

Artificial Intelligence in Design'02

Artificial Intelligence in Design '02

Edited by

John S Gero

*Key Centre of Design Computing and Cognition
University of Sydney
Australia*



SPRINGER-SCIENCE+BUSINESS MEDIA, B.V.

A C.I.P Catalogue record for this book is available from the Library of Congress

ISBN 978-90-481-6059-4 ISBN 978-94-017-0795-4 (eBook)
DOI 10.1007/978-94-017-0795-4

Printed on acid free paper

All rights reserved

© 2002 Springer Science+Business Media Dordrecht

Originally published by Kluwer Academic Publishers in 2002

No part of the material protected by this copyright notice may be reproduced or utilised in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage and retrieval system, without written permission from the copyright owner.

TABLE OF CONTENTS

Preface	ix
Design Synthesis	1
<i>Synthesis in designing</i>	3
Tim Smithers	
<i>A graphical notation for mixed-initiative dialogue with generative design systems</i>	25
Sambit Datta and Robert F Woodbury	
<i>Web-based configuration of virtual private networks with multiple suppliers</i>	41
Alexander Felfernig, Gerhard Friedrich, Dietmar Jannach and Markus Zanker	
Frameworks for Design	63
<i>Constructing design worlds</i>	65
Mihaly Lenart and Ana Pasztor	
<i>The situated function–behaviour–structure framework</i>	89
John S Gero and Udo Kannengiesser	
<i>Representational flexibility for design</i>	105
Rudi Stouffs and Ramesh Krishnamurti	
Spatial Synthesis and Analysis	129
<i>Analysis of architectural space composition using inductive logic programming</i>	131
Noritoshi Sugiura and Shigeyuki Okazaki	
<i>Towards an architectural design system based on generic representations</i>	153
Sviataslav Pranovich, Henri Achten and Jarke J Van Wijk	
<i>Digital sandbox</i>	165
Ellen Yi-Luen Do	
Conceptual Knowledge in Design	189
<i>MMforTED: A cognitive tool fostering the acquisition of conceptual knowledge about design products</i>	191
Elio Toppano	
<i>From concept to embodiment: challenge and strategy</i>	215
Zhi Gang Xu, Ming Xi Tang and John Hamilton Frazer	
<i>Artificial intelligence for the design and grading of precious stones</i>	237
Tony Holden and Matee Serearuno	
Learning from Human Designers	259
<i>Using protocol analysis to investigate collective learning in design</i>	261
Zhichao Wu and Alex Duffy	

<i>5. 8 analogies per hour</i>	285
Pierre Leclercq and Ann Heylighen	
<i>Towards computational tools for supporting the reflective team</i>	305
Andrew W Hill, Andy Dong and Alice M Agogino	
Evolutionary Approaches in Design	327
<i>An evolutionary approach to the inverse problem in rule-based design representations</i>	329
Stephan Rudolph and Rolf Alber	
<i>Evolving three-dimensional architecture form</i>	351
Luisa Caldas	
<i>Strategic shape design</i>	371
Toshiharu Taura, Takayuki Shiose and Ryohei Ishida	
<i>An evolutionary framework for enhancing design</i>	383
Kwai Hung Chan, John Hamilton Frazer and Ming-Xi Tang	
Knowledge Support for Design	405
<i>Knowledge support for customer-based design for mass customization</i>	407
Xuan Fang Zha and Wen F Lu	
<i>Elucidating the design requirement for conventional and automated conceptual design</i>	431
Mansur Darlington and Stephen J Culley	
<i>Case-based design facilitated by the design exemplar</i>	453
Joshua J Summers, Zoe Lacroix and Jami J Shah	
Systemic Support	477
<i>Connectivity as a key to supporting design</i>	479
Claudia Eckert and P John Clarkson	
<i>Automated (re-)design of software agents</i>	503
Frances MT Brazier and Niek JE Wijngaards	
<i>Automated toolset selection for feature manufacturing</i>	521
Farsad Badjgholi, Burkhard Kittl and Markus Stumptner	
Components in Design and Design Models	545
<i>Requirements specification and automated evaluation of dynamic properties of a component-based design</i>	547
Catholijn M Jonker, Jan Treur and Wouter C A Wijngaards	

<i>Identifying component modules</i>	571
Robert I Whitfield, Joanne S Smith and Alex B Duffy	
<i>Perspectors</i>	593
John Haymaker, Martin Fischer and John Kunz	
<i>Product data exchange using ontologies</i>	617
Christel Dartigues and Parisa Ghodous	
Author Index	639
Contact Authors' Email Addresses	641

International Review Panel

- Henri Achten, Eindhoven University of Technology,
The Netherlands
- Tomasz Arciszewski, George Mason University, USA
Can Baykan, Middle East Technical University,
Turkey
- Peter Bentley, University College London, UK
Joao Bento, Instituto Superior Tecnico, Portugal
Bill Birmingham, University of Michigan, USA
Frances Brazier, Vrije Universiteit, The Netherlands
David Brown, Worcester Polytechnic Institute, USA
Ken Brown, University of Aberdeen, UK
Scott Chase, University of Strathclyde, UK
Mao-Lin Chiu, National Cheng Kung University,
Taiwan
- Dave Corne, University of Reading, UK
Alex Duffy, University of Strathclyde, UK
Steven Fenves, NIST, USA
Susan Finger, Carnegie Mellon University, USA
Ulrich Flemming, Carnegie Mellon University, USA
Haruyuki Fujii, Tokyo Institute of Technology,
Japan
- John Gero, University of Sydney, Australia
Alberto Giretti, University of Ancona, Italy
David Gunaratnam, University of Sydney, Australia
John Haymaker, Stanford University, USA
Jeff Heisserman, Terabeam Corporation, USA
Ann Heylighen, KU Leuven, Belgium
Tony Holden, University of Cambridge, UK
Leo Joskowicz, Hebrew University of Jerusalem, Israel
Richard Junge, Technical University Munich,
Germany
- Vladimir Kazakov, University of Sydney, Australia
Mark Klein, MIT, USA
Rudiger Klein, Daimler-Chrysler, Germany
Janet Kolodner, Georgia Institute of Technology,
USA
- Alex Koutamanis, Delft University, Netherlands
Ramesh Krishnamurti, Carnegie Mellon University,
USA
- Bimal Kumar, University of Strathclyde, UK
John Lee, University of Edinburgh, UK
Mark Lee, University of Wales, UK
Mihaly Lenart, University of Kassel, Germany
Udo Lindemann, Technische Universität München,
Germany
- Hod Lipson, Brandies University, USA
Ray McCall, University of Colorado, USA
Mary Lou Maher, University of Sydney, Australia
Peter Matthews, Cambridge University, UK
Hari Narayanan, Auburn University, USA
Rivka Oxman, Technion Israel Institute of
Technology, Israel
- Ian Parmee, Plymouth University, UK
Joan Peckham, University of Rhode Island, USA
Feniosky Pena-Mora, MIT, USA
Pearl Pu, EPFL, Switzerland
Terry Purcell, University of Sydney, Australia
Rabbee Reffat, University of Sydney, Australia
Yoram Reich, Tel Aviv University, Israel
Michael Rosenman, University of Sydney,
Australia
- Stephan Rudolf, University of Stuttgart, Germany
Linda Schmidt, University of Maryland, USA
Thorsten Schnier, University of Birmingham, UK
Mark Schwabacher, NIST, USA
Stephen Scrivener, University of Derby, UK
Simeon Simoff, University of Technology,
Sydney, Australia
- Ian Smith, EPFL, Switzerland
Tim Smithers, VICOMTech, Spain
Ram Sriram, NIST, USA
Louis Steinberg, Rutgers University, USA
George Stiny, MIT, USA
Rudi Stouffs, Delft University of Technology,
Netherlands
- Markus Stumptner, University of South Australia,
Australia
- Masaki Suwa, Chukyo University, Japan
Hideaki Takeda, Nara Institute of Science and
Technology, Japan
- Hsien-Hui Tang, University of Sydney, Australia
Toshiharu Taura, Kobe University, Japan
Wade Troxell, Colorado State University, USA
Ziga Turk, University of Ljubljana, Slovenia
George Turkiyyah, University of Washington,
USA
- Angi Voss, GMD, Germany
Rob Woodbury, Technical University of British
Columbia, Canada

PREFACE

One of the foundations for change in our society comes from designing. Its genesis is the notion that the world around us either is unsuited to our needs or can be improved. The need for designing is driven by a society's view that it can improve or add value to human existence well beyond simple subsistence. As a consequence of designing the world which we inhabit is increasingly a designed rather than a naturally occurring one. In that sense it is an "artificial" world. Designing is a fundamental precursor to manufacturing, fabrication, construction or implementation. Design research aims to develop an understanding of designing and to produce models of designing that can be used to aid designing.

Artificial intelligence has provided an environmental paradigm within which design research based on computational constructions, can be carried out.

Design research can be carried out in variety of ways. It can be viewed as largely an empirical endeavour in which experiments are designed and executed in order to test some hypothesis about some design phenomenon or design behaviour. This is the approach adopted in cognitive science. It often manifests itself through the use of protocol studies of designers. The results of such research form the basis of a computational model. A second view is that design research can be carried out by positing axioms and then deriving consequences from them. If the axioms can be mapped onto design situations then the consequences should follow. This is the approach adopted in mathematics and logic and forms the basis of a small but powerful area in design research. A third view, and the most common one in the computational domain is that design research can be carried out by conjecturing design processes and constructing computational models of those processes and then examining the behaviours of the resulting computational systems. Artificial intelligence in design research utilises all three approaches.

The papers in this volume are from the Seventh International Conference on Artificial Intelligence in Design (*AID'02*) held in Cambridge University, UK. They represent the state-of-the-art and the cutting edge of research and development in artificial intelligence in design. They are of particular interest to researchers, developers and users of advanced computation in design.

In these proceedings the papers are grouped under the following nine headings, describing both advances in theory and application and

PREFACE

demonstrating the depth and breadth of the artificial intelligence paradigm in design:

- Design Synthesis
- Frameworks for Design
- Spatial Synthesis and Analysis
- Conceptual Knowledge in Design
- Learning from Human Designers
- Evolutionary Approaches in Design
- Knowledge Support for Design
- Systemic Support
- Components in Design and Design Models

All papers were extensively reviewed by three referees drawn from the large international panel of referees listed earlier. Thanks go to them, for the quality of these papers depends on their efforts. The reviewers' recommendations were then assessed before the final recommendation was made.

Anne Christian deserves particular thanks for it was she who took what should have been consistently formatted submissions but were not, and turned them into a coherent whole – no mean effort. The final quality of the manuscript bears her mark.

John S. Gero
University of Sydney
and
Massachusetts Institute of Technology
April 2002

DESIGN SYNTHESIS

Synthesis in design
Tim Smithers

*A graphical notation for mixed-initiative dialogue
with generative design systems*
Sambit Datta and Robert F Woodbury

Web-based configuration of virtual private networks with multiple suppliers
Alexander Felfernig, Gerhard Friedrich, Dietmar Jannach and Markus Zanker

SYNTHESIS IN DESIGNING

TIM SMITHERS

*VICOMTech and Mondragon Unibertsitatea
Spain*

Abstract: Synthesis has mostly been presumed to be an important aspect of designing. Few people in design research have felt the need to explain why it is. Synthesis has usually been presumed to be concerned with the generation of design solutions. Little or no design research has questions this assumption, or seriously proposed anything else. Here a Knowledge Level theory of designing KLD_v^E ¹ is used to support a theoretical investigation of synthesis in designing, in terms of the knowledge used and generated. First, the concept of knowledge used is introduced. *Designing as exploration* is then briefly explained, and, KLD_v^E ¹ which embodies this view of the nature of designing, is set out. Then, using the different kinds of knowledge, and their roles and relationships, defined by KLD_v^E ¹ different kinds of synthesis activities are identified and discussed. The motivation for doing this is the belief that, knowing what knowledge is used and generated by the different synthesis activities in designing is one way of understanding the nature and role of synthesis in designing, and how it might be supported effectively using knowledge based systems.

1. Introduction

Synthesis is the putting together of things, to build up something to form a whole out of parts or elements¹. In trying to understand the nature and role of synthesis in designing, we must first be clear about what is being synthesised. From this can follow a proper identification of what is put together; what is used to form the whole. Knowing what is synthesised, and from what, then makes possible the identification of what kinds of designing knowledge are involved in—used and created by—the synthesising. Knowing what knowledge is used and created is one way of usefully understanding the nature and role of synthesis in designing, and

¹ Chemical synthesis, in which specific molecules are made by combining particular atoms, is, perhaps, the archetypal example of a synthesis activity or process.

how it might be supported effectively using knowledge based systems. There are two ways in which what is synthesised in designing, and from what, can be established: empirical studies of some particular designing; or by appealing to a theory of designing. The first, empirical studies, could yield useful results for the kinds of designing studied, but would leave the generalisation of this understanding difficult or unsupported. A theoretical starting point could lead to a general and deeper understanding of synthesis in designing, if the theory used is general enough and good enough. This paper uses KLD_V^E , a Knowledge Level theory of designing, to identify different kinds of synthesis activities in designing in general. An important result of this analysis is that synthesis in designing cannot be properly understood as just one kind of activity. It is shown to be a multiple and interrelated set of distinct knowledge using and knowledge creating activities. In the next section the concept of knowledge used in KLD_V^E is briefly introduced. Section 3 introduces *Designing as Exploration*, the foundation of KLD_V^E as a general theory of designing. In Section 4, the basic contents and structure of KLD_V^E are introduced. Section 5 then presents an analysis of the different kinds of synthesis that can be identified in designing using KLD_V^E . The paper ends with a discussion of design synthesis, as it is identified in other models and theories of designing, and how they relate to the multiple-synthesis activities identified in KLD_V^E .

2. Knowledge and the Knowledge Level

Newell (1982)² presents the *Knowledge Level* as a new level of abstraction in computer and cognitive systems. He placed it directly above the Symbol Level, and argued that it is needed to properly understand and specify the problem solving behaviour of an intelligent agent. For Newell and Simon, this Knowledge Level formed an important component of their theory of cognition in both human and artificial agents—since, for Newell and Simon, human agents are intelligent because they are examples of physical symbol systems, just as computers can be when programmed appropriately (Newell, et al. 1958; Newell and Simon 1976; Newell 1990).

In Newell's original presentation, at the Knowledge Level (KL), an intelligent agent is composed of goals, actions, and a body. The medium at the KL, the composition of the body, is knowledge (what the agent

² Though Alan Newell was the sole author of this paper, which is based upon his AAAI Presidential address of 1981, it is widely accepted that the ideas he presented are both his and those of Herbert Simon.

knows), and the law of behaviour, at the KL, is the principle of rationality: that the agent uses its knowledge to select one or more of its actions to achieve its goals. As Newell (1982) observes, from this definition of the Knowledge Level, it follows that knowledge is intimately linked with rationality, so that we can say that systems that are observed to act rationally can be said to have knowledge, and to have knowledge is to have a capacity to act rationally. In other words, the concept of knowledge that underlies, and is fundamental to, the Knowledge Level, is a competence notion. Knowledge, according to Newell, is a capacity for rational action.

In presenting the Knowledge Level, Newell did not think he was presenting anything new to the Artificial Intelligence (AI) community. Rather he referred to its presentation as a rational reconstruction, a making explicit, of a concept of knowledge that had been developed and used since the earliest days of AI, albeit tacitly or implicitly, and for some, at least, unknowingly. Knowledge as a capacity for rational action is thus to be understood as *the* concept of knowledge adopted and used in AI. It is, however, important to note that though widely adopted and used in AI, this concept of knowledge is radically different from the more conventional, or Classical, concept of knowledge we have from the field of Epistemology and philosophers of knowledge. These characterise knowledge as *justified true belief*, a concept we find first in Plato's (c.428– c.348 BC) "Meno, Phaedo, and Theaetetus" (one of Plato's so called Early Dialogues featuring Socrates).

Newell's conception of knowledge, has two big advantages over the Classical view. First, that knowledge as a capacity for rational action is a practical concept which is not hard to use. The second advantage is that it escapes from the problems that epistemologists and philosophers continue to struggle with concerning applicability of the concept of knowledge as justified true belief, and its logical adequacy. See the so called "Gettier counter-examples," (Gettier 1967) and see (Quine 1987).

The practicality of knowledge as a capacity for rational action has been well used and well demonstrated in recent years with the development of modern Knowledge Engineering (KE) methods, such as CommonKADS, for example, (Schreiber et al. 1999). These all take Newell's concept of knowledge as a common starting point. They have, however, modified Newell's original view of the Knowledge Level, to make it more useful in supporting the Knowledge Level modelling of expert behaviour. These KE developments of the KL are important, and can be summarised as involving three basic changes to Newell's original proposal:

1. All cognitive connotations and implications are dropped.
2. The KL is decoupled from the Symbol Level and taken to exist independently of the Symbol Level and all other system levels below it.
3. Different types of knowledge are defined, which play different roles in the (modelled) expert behaviour, in contrast to knowledge being one amorphous undistinguished body, as Newell defined it.

The first two of these changes allow the concept of the Knowledge Level to be used as a useful level of abstraction, without necessarily committing to cognition being a property of a physical symbol system, as Newell and Simon (and others) believe. They remove a much debated aspect, but leave a powerful abstraction level. The third change, makes the KL much more useful in modelling real expert or human problem solving behaviour. It allows KL models to have structure, which, in turn, can be used to model and specify important aspects of real knowledge-based systems. In CommonKADS, for example, we have domain knowledge, task knowledge, and inference knowledge. The effective application of these different types of knowledge, the roles they play, and the ways they relate, are embodied in a set of principles for the KL modelling of expertise, (Akkermans et al. 1994) which make knowledge engineering a well defined modelling activity similar to other software engineering methods and practices.

3. Designing as Exploration

We design things when there is a need or a desire for some part or aspect of our world to be different, and we cannot immediately specify how it should or could be changed.

Designing is thus not properly understood as problem solving, since it does *not* start with a problem to be solved. Problems, to be well specified, must, amongst other things, specify what can properly be a solution, the solution space. Needs and desires do *not* specify what can satisfy them, they simply identify what we would like to be different, or what we need to be different. Designing does involve problem solving, as we will see, but this only accounts for a certain part of the overall process, not the process as a whole or in general.

Designing is also not planning since to plan something we first need to have a specification for what is to be achieved by executing the plan. The needs or desires that motivate any designing are not specifications of what is to be designed; they do not identify what it will or could take to satisfy them. A final design is what specifies what must be realised or

implemented to satisfy the needs or desires. We may then plan its implementation, and we often do.

What makes designing a particular kind of activity, distinct from problem solving and planning, and other human activities, is that designing must start with something that neither specifies what is required nor defines a problem to be solved, yet it must arrive at a design—a specification—for something that, when realised or implemented, will satisfy the motivating needs or desires: the realised design should remove the need or desire for something to be different.

For example, the need for a new font for use in minimalist overhead slide designs which is distinctive and not too formal, does not specify anything directly about what the new font should look like, its design. However, if you know anything about font designing, it does suggest certain requirements quite directly, such as a sans serif font type, (Smithers 2000). This stated need also does not, in and of itself, define a problem to be solved. It does not specify or otherwise identify any criteria or conditions that must be satisfied by a solution—another necessary component of any properly posed problem.

Thus designing must finish with realisable specifications, designs, without starting with anything that can be properly understood as a problem for which the final design is a solution. This apparent paradox—arriving at a kind of solution without starting with a problem—is what makes designing different from other activities. It is the characteristic feature of designing.

Designing resolves this paradox by actively constructing the problem or problems whose solution or solutions can form a design or parts of a design. Designing is thus puzzle making and puzzle solving, to borrow a term from Archea (1987). Usually the problem forming—puzzle making—and solution finding—puzzle solving—are tightly integrated, incremental activities which are driven by reflection on what is happening. Furthermore, the problem forming aspects are often tacit and never made explicit in the work of the designers.

This characterisation of designing as integrating problem forming and problem solving is reflected in the writings of numerous other design researchers. Jones (1992) writes, for example,

If, as is likely, the act of tracing out the immediate steps exposes unforeseen difficulties or suggests better objectives, the pattern of the original problem may change so drastically that the designers are thrown back to square one. It is as if, during a game of chess, one could chose to switch, or be obliged to switch, to a game of snakes and ladders. This instability of the problem is what makes designing so much more difficult and

more fascinating than it may appear to someone who has not tried it. (page 10).

Lawson (1990) identifies the same strong interaction between problems and solutions in designing when he writes,

In this respect designing is rather like devising a crossword.

Change the letters of one word and several other words will need altering necessitating further changes. (page 45).

Lawson called this as ‘analysis through synthesis,’ (Lawson 1990), and Getzels and Csikszentmihalyi (1976) presented examples of this kind of ‘problem- finding’ behaviour in a study of art students.

The essential situated and reflective nature of this integrated and incremental problem forming and problem solving (or, as sometimes happens, solution identifying and problem forming) is an example of what Schön has investigated and written about as reflection in action, (Schön 1983; Schön 1985; Schön 1992). It constitutes a kind of exploration³ of what problems and solutions can be devised and developed to support the construction of a design that satisfies the motivating needs or desires.

At the centre of designing is thus a combination of problem forming and solution finding activities. But this cannot be all there is. Problem solutions need well defined problems, and well defined problems need to embody explicit conditions on the possible solutions: well defined problems must define the space of possible solutions.

The conditions used to define the solution space are derived from identified criteria or requirements: they are operationalisations of criteria that are identified as requiring to be met in order for a design to satisfy the motivating needs or desires. The devising and forming of these requirements is thus also an integral, and necessary, part of designing. They are what a designer uses to frame a client’s needs and desires in such a way that designing can get started, and they are what a designer needs to discover and introduce for the designing to progress towards an acceptable final design that satisfies the motivating needs and desires, (Smithers and Troxell 1990; Smithers 1992; Smithers 1998). See also (Suwa et al. 2000) for a recent study of requirement discovery and synthesis in designing.

³ We use the term *exploration*, rather than *search* here to capture the idea that this incremental and integrated problem forming and problem solving must result in an effective uncovering, discovery, of what is possible, and a thorough understanding of why, how, and in what way it is. To search means to look for something in particular. It is what we might do to solve well defined problems, but it is not useful when we don’t know much or anything about what we will or need to find by an exploration.

Solutions to well defined problems do not, however, necessarily meet all or any of the requirements. This depends upon how well or how completely the problem operationalises the current set of requirements. Before solutions can be considered as possible designs, or as parts of a possible design, they need to be evaluated with respect to the criteria operationalised in the problem they are a solution to. In general, this requirements forming, problem defining, solution finding, and solution evaluation, do not occur in some simple or linear fashion. Rather, they are all combined in an incremental and integrated process. They form an essential and symbiotic set of activities which change and develop together at the core of any and all designing: the requirements, problems, solutions and their evaluations co-evolve as the designing progresses.

In summary then, *Designing as Exploration* characterises designing as the *exploration of the problems that can be devised whose solutions can be shown to satisfy the needs or desires that motivate the designing*.

4. **KLD_V^E₁ : A Knowledge Level Theory of Designing**

The idea, presented in the previous section, that, at its core, all designing involves the incremental and symbiotic forming of requirements, defining of problems, finding of solutions, and evaluation of those solutions, describes the basic mechanism of designing; a mechanism for exploring what is possible. How this basic mechanism is realised, what it looks like in practice, depends very much upon the details of the designing being considered. It can, and does, look quite different across different examples of designing, even within the same domain. It also often looks quite different across different designers designing the same kinds of things.

Developing a general theory of Designing as Exploration that tries to account for and to explain all this natural and normal variation in the actual practice of designing, is far too difficult. It would require a much more complete theory of cognition and a much better understanding of the psychology and sociology of artistic, personal, and professional practices, (Smithers 1996).

Given that all designing is a kind of intelligent behaviour. It can be understood as a kind of knowledge applying and creating process, if knowledge is taken to be a competence notion, as introduced in section two above. The Knowledge Level thus offers a suitable and appropriate level of abstraction at which we can seek to develop a general theory of designing: a theory about what kinds of knowledge are necessary and sufficient for designing, what roles these different kinds of knowledge

play in the process, and what the relationships are between them in designing. It offers both a practical level at which to build a general theory of designing, and a way of building useful theories, which can support the knowledge engineering of design support systems, (Smithers 1996), and the specification of knowledge management infrastructures, for example, (Smithers 1988).

$K.D_v^E_1$ is an attempt to develop a KL theory of Designing as Exploration. It is supposed to define the necessary and sufficient kinds of knowledge involved in all and any kind of designing, the roles they play and the relationships between them.

This section sets out the basic kinds of knowledge defined in $K.D_v^E_1$, and the roles and relations these have in designing. It does not, however, present the complete theory.⁴.

4.1 THE KNOWLEDGE STRUCTURE OF $K.D_v^E_1$

First we will introduce and briefly explain the different kinds of knowledge defined by $K.D_v^E_1$ and the different roles and relationships they can have: the basic knowledge structure of $K.D_v^E_1$.

$K.D_v^E_1$ defines different *kinds* of knowledge because, as we have seen, Designing as Exploration involves knowing about different kinds of things. If we take knowledge to be a capacity for rational action, then knowing about X means a capacity for rational action with respect to X . Furthermore, taking knowledge as a capacity for rational action allows us to think of different kinds of knowledge as being different kinds of actors, with a capacity to act in different ways and with respect to different things. In other words, the different kinds of knowledge can have different *roles* in designing. Finally, to form a coherent and effective overall process, the different kinds of knowledge, with their different roles, must be properly related. This results in the need for different kinds of relationships between the different kinds of knowledge and their roles.

According to $K.D_v^E_1$ knowledge is both used and constructed during designing. In the case of the knowledge that is used, there are three basic kinds:

1. General context knowledge, K.gc;
2. Exploration knowledge, K.ex; and
3. Design knowledge, K.dk.

Each of these basic kinds of knowledge that are used in designing embed other kinds of knowledge that are necessary for designing. The

⁴ This would require a proper definition of all the terms and concepts used.

different kinds of constructed knowledge are presented below. The basic roles that the different kinds of knowledge in KLD_v^E can have are:

- *A Supporting role;*
- *A Constructing role; and*
- *A State maintaining role.*

The relations that are used in KLD_v^E to combine the basic kinds of knowledge into a Knowledge Level version of the Design as Exploration mechanisms presented in Section 3, are:

- *Embedded in;*
- *Supports;*
- *Used in (the construction of); and*
- *Increments and/or modifies.*

4.2 KLD_v^E KNOWLEDGE KINDS, THEIR ROLES AND RELATIONS

Each of the three basic kinds of used knowledge have a series of *embedded component* kinds of knowledge.

4.2.1 General context knowledge

All designing necessarily takes place in a wider context formed by the prevailing cultural, political, socioeconomic, and technological conditions. Designing is always carried out with respect to this wider context and is always relative to it. Knowledge of this wider context is thus a necessary knowledge component of designing, which, in KLD_v^E , is called *General Context Knowledge*, or $K.gc$ for short⁵.

The general context knowledge, $K.gc$, also embeds:

- $K.dm$, knowledge of the domain or domains;
- $K.cc$, knowledge of the customer/client context;
- $K.nd$, knowledge of the needs and desires that motivate the designing; and
- $K.pc$, knowledge of the design practice context; business, artistic, personal, institutional, or social styles, customs, or cultures.

Within the wider context, designing is usually located within some particular domain (or domains) such as font design, or VLSI design, or the design of schools, for example. Knowledge of the domain, $K.dm$, in which the designing is located, is necessary for designing. It is *embedded* in

⁵ For a nice example illustrating the importance of an awareness and proper understanding of technological issues and developments taking place in the general context of a design domain, see the paper Graham (1974) on programming language design, in which she shows how the debate on the use of the ‘go-to’ statement complicated the design of new languages at the time.

K.gc. It plays a supporting role in designing, and it has a supports relation to the exploration knowledge, K.ex, and to the design knowledge, K.dk.

Another important part of the knowledge of the wider context, is knowledge of the customer or client context, K.cc. This kind of knowledge must shape the designing so that the resulting design satisfies the customer or client's needs and desires in a way that fits with his or her or their situation and context. It is embedded in the general context knowledge. It has a supporting role in designing, and it has a supports relation to the exploration knowledge, K.ex, and to the design knowledge, K.dk.

Knowing the needs and desires of the client or customer is also a necessary kind of knowledge for designing. this $K.D_v^E$ is called *K.nd*. Although designing necessarily requires knowledge of the motivating needs or desires, it is important to understand that the actual needs or desires always remain the property of the customer or client. This is why it is embedded in K.gc. The needs and desires can never be changed by the designing (by the designer or designers) though they can be, and sometimes are, changed by the client or customer as a result of interaction with the ongoing designing. How close the relationship is between the motivating needs and desires and the actual designing varies considerably from domain to domain. In architecture, for example, there can be a close relationship and considerable interaction between the needs and desires of the client and the designing that is motivated by them. In product design, however, there may be no direct contact with the customers; the people for whom the product is intended. Designing can also be motivated by a presumed need or desire. The Sony Walkman™ is a good example of a product being designed before anybody actually thought they needed or wanted one. Knowledge of the needs and desires plays a supporting role in designing, and it has a supports relation to the exploration knowledge, K.ex, and to the design knowledge, K.dk.

Designing is not usually a one-off activity. The knowledge of the culture, style, methods, and customs that is developed by, and that develop around any designing practice, thus also forms an important kind of knowledge that is embedded in the general context knowledge of any designing. $K.D_v^E$ this is called K.pc and it plays a supporting role in designing. K.pc has a supports relation to the exploration knowledge, K.ex, and to the design knowledge, K.dk.

4.2.2 *Exploration knowledge*

At the core of all designing, according to Design as Exploration, is a process that combines and integrates requirements forming, problem

defining, solution finding, and solution evaluation. The knowledge used by this exploration process is called K.ex, and each of the core aspects depend upon particular kinds of knowledge that are embedded in K.ex.

The exploration knowledge, K.ex, thus embeds:

- K.r, knowledge of requirements formation, recognition, and development;
- K.p, knowledge of well formed problem definition, modification, and revision;
- K.s, knowledge of solution finding; and
- K.e, knowledge of solution evaluation, which itself can embed:
 - K.an, analysis knowledge;
 - K.sm, simulation knowledge;
 - K.an, prototyping knowledge; and
- K.lp, knowledge of local plan formation.

Requirements specify the criteria that are used to evaluate, judge, and decide if a solution to a problem could form a possible design, or a part of a possible design, that can satisfy the needs and desires. They do not typically or necessarily have any direct or unique correspondence to the needs and desires, but they will, in part be derived from or be implied by them. Furthermore, in general it is not possible to identify, at the start of designing, all the criteria that will be needed to complete a final design: the motivating needs and desires do not typically identify or directly imply all the criteria that will be needed. Nor is it typically possible to start with a consistent set of requirements. Requirements, more typically, and in general, start by being incomplete and inconsistent, and they may also be imprecise, ambiguous, and impossible. One of the necessary aspects of designing is to incrementally discover and develop a complete, consistent, precise, unambiguous set of criteria that can be used to show that a final design is both possible and can satisfy the motivating needs and desires in an acceptable way—acceptable to the customer or client, and to the wider context in which the designing takes place. This incremental construction and development depends upon the defining of well formed problems and finding solutions to them, which are then evaluated with respect to some or all of the current requirements. It is an aspect of designing that depends upon knowledge of how to form and develop requirements. This kind of knowledge is called K.r. It is embedded in the exploration knowledge, and it has a constructing role within K.D_{y^E}¹.

Well defined problems must specify the conditions that any solution must satisfy: a well defined problem must define a solution space. These conditions are obtained by some kind of operationalisation of some or all

of the requirements in the current set of design requirements. This operationalisation basically means identifying or defining parameters and the values or value ranges that they can take. Often there can be more than one way of operationalising a requirement. So for any requirement or set of requirements, it may be possible to devise more than one problem definition. The failure to be able to define a well formed problem from the current set of requirements, and the various attempts to operationalise them, are important in the process of identifying any inadequacies in the current set of design requirements: any incompleteness, inconsistency, imprecision, ambiguity, or impossibility. The knowledge needed to form and to further develop well defined problems in this way is called K.p, and it is embedded in the exploration knowledge. It has a *constructing* role within K.D_v^E¹.

No matter how well defined a problem is, finding one or more solutions to it may be more or less difficult. Obtaining an exact solution also may not be possible, for practical and/or theoretical reasons. A sufficiently close solution may then be selected. The problem definition may then be changed so that the selected solution becomes an exact solution of the new problem. In this way problem solving can and does drive problem definition or re-definition in designing. At other times, the exploration may start with a solution (from some previous designing), which is then modified or adapted, and a well formed problem for it then developed. These too, are ways in which new requirements might be discovered or identified, or inadequacies in the current set of requirements established. The knowledge needed to solve the problems that are formed is called K.s and it is embedded in the exploration knowledge of designing. This kind of knowledge also has a *constructing* role within K.D_v^E¹.

Establishing if the solution to a well defined problem can form a design, or a part of a design, that satisfies the needs or desires requires evaluation. This evaluation is carried out with respect to the criteria from the requirements that have been operationalised in the problem that produces the solution in question. The process of exploration thus needs knowledge of how to carry out adequate evaluations, called K.e in K.D_v^E¹. The evaluation of problem solutions is sometimes a direct process, but may also need an analysis of the solution first. In this case, we also have knowledge of how to do an appropriate analysis of the solutions, K.an, which is embedded in K.e. Similarly, an evaluation may involve the development and running of a simulation, in which case we would also have K.sm embedded in K.e. Or an evaluation may require the constructing and testing of a physical prototype, in which case the designing will involve K.pr embedded in K.e. The evaluation knowledge,

K.e, together with K.an, K.sm, and K.pr all play constructing roles within the exploration process in designing.

Since any problem defining, solution finding, or solution evaluation may result in the identification of some need to change or modify the current set of design requirements, or to define a new problem, the exploration process—of problem definition, solution finding, solution evaluation, and requirements identification, modification and development—typically needs to be coordinated, so that it remains coherent and locally effective. This results in the need to devise and maintain what are called local plans. These are essentially lists of things to do: define a new problem or modify an existing one; find a new solution; evaluate another solution, add a new requirement to the current set, etc. An important property of these local plans is that they are often not completed, and sometimes never even started: the list of things to do that they identify are never completed or not even attempted. This simply reflects the contingent and reflective nature of the exploration process in designing. The knowledge needed to form and maintain local plans is called K.lp and it is embedded in K.ex. It has a constructing role within the exploration process in designing.

4.2.3 Design Knowledge

The design knowledge, K.dk, embeds:

- K.dd, knowledge of design description formation and modification;
- K.dc, knowledge of design documentation, which can embed:
 - K.dr, knowledge of design rational maintenance; and
- K.dq, knowledge of design presentation preparation.

Solutions to well formed problems do not directly constitute design descriptions, or designs, at least, not in general. Proper design descriptions will usually be represented in a different way, and may also require further information to be included. And design descriptions may be compiled from a number of problem solutions. Forming proper design descriptions, even partial design descriptions, thus requires knowledge of how to do this, called K.dd. This kind of knowledge is embedded in the design knowledge, K.dk. It has a constructing role within the exploration process in K_D^E ¹.

Documenting the designing is a domain dependent activity in designing, but in some designs it is also a formal requirement. Maintaining an explicit record of the design rational, for example, would form part of the documentation activities in designing, which requires knowledge of how to do this, K.dc. This is embedded in the design knowledge, K.dk, and it plays a *constructing* role within the exploration process in K_D^E ¹. K.dc

can also embed knowledge of design rational development and maintenance, K.dr, which also has a constructing roles within KLD_v^E ¹.

The presentation of designs, intermediate designs, or partial designs may also be a formal or normal part of designing. This requires knowledge of how to prepare and make effective presentations of designs, K.dq. Again, this is a kind of knowledge embedded in the design knowledge, K.dk, and it plays a constructing role within the exploration process in KLD_v^E ¹.

4.2.4 Constructed knowledge

In the case of the knowledge that is constructed during designing, KLD_v^E ¹ identifies the following basic kinds:

- C.r, knowledge of the current set of requirements and their status;
- C.p, knowledge of the problems so far defined, their status, and organisation;
- C.s, knowledge of the problem solutions so far generated, their status and organisation;
- C.e, knowledge of the solution evaluations produced so far, which can embed:
 - C.an, knowledge of the analyses produced so far;
 - C.sm, knowledge of the simulations made so far; and
 - C.pr, knowledge of the prototypes built so far;
- C.lp, knowledge of the local plans developed so far, their status, and organisation;
- C.dd, knowledge of the design descriptions developed so far, their status, and organisation;
- C.de, knowledge of the design documentation produced so far, which can embed:
 - C.dr, knowledge of the design rational maintained so far;
- C.dq, knowledge of the design presentations prepared so far;
- C.dh, knowledge of the history of the designing so far.

The requirements are a basic and necessary construction of designing, and knowledge of the requirements so far devised, and their status and organisation, is a necessary kind of knowledge constructed by designing. In KLD_v^E ¹ this knowledge of the current set, or sets, of requirements and their status and organisation, is called *C.r*.

The knowledge of the problems that have so far been defined, their current status, and the way they are related and organised, is also a construction of the designing. In KLD_v^E ¹ this kind of constructed knowledge is called *C.p*.

Similarly, the knowledge of the problem solutions that have been found, and how these are related to the current state of the exploration,

is another basic kind of knowledge constructed by designing. In $K.D_v^E$ this is called C.s.

Evaluating problem solutions, to see if they can form possible designs or parts of designs, also constructs a basic kind on knowledge, which is called C.e. Depending on what is involved in doing the evaluating, this kind of knowledge can also embed knowledge created as a result of doing any analyses, C.an, knowledge created as a result of doing any simulations, C.sm, and/or knowledge created as a result of making any prototypes, C.pr.

The development and use of the local plans during the exploration process also results in the construction of a basic kind of knowledge in, $K.D_v^E$, which is called C.lp.

These first four kinds of constructed knowledge (with their embedded kinds), as their names imply, are related to the basic kinds of knowledge used in the exploration process. The next three basic kinds of constructed knowledge are related to the kinds of design knowledge used. These are: the knowledge that results from forming and developing design descriptions, or partial design descriptions, C.dd; the knowledge that results from documenting and recording the designing, C.dc, which may embed the knowledge of the design rational that is being maintained, C.dr; and the knowledge that results from presenting intermediate or partial designs and the final design, C.dq. A final basic kind of knowledge that is constructed as a result of designing is what is called the designing history knowledge in $K.D_v^E$, C.dh. Often this is not formally recorded, but it is a kind of knowledge that designers have of the designing they are involved in, and it can be important in influencing and forming any subsequent designing.

All these different kinds of knowledge have an axiomatic status as basic components of $K.D_v^E$; they are defined to be components of designing, rather than derived from more basic elements.

5. Synthesis in $K.D_v^E$

With the presentation of the basic knowledge components of $K.D_v^E$, and their roles and relationships, we can now use $K.D_v^E$ to identify where synthesis activities occur in designing. As we stated at the outset, synthesis is the putting together of things to build up something and so form a whole out of parts or elements. This putting together might be done all at once, if all the parts or elements to be used are all available, or it might be done incrementally as parts or elements are identified, become available, or are otherwise created or produced. With this

understanding of synthesis, using $K.D_v^E$, we can identify four distinct but related kinds of synthesis in designing, as it is presented here:

1. the synthesis of requirements;
2. the synthesis of well defined problems;
3. the synthesis of local plans; and
4. the synthesis of design descriptions.

Each of these different kinds of synthesis will be considered in a more detail in the following subsections.

5.1 SYNTHESIS OF REQUIREMENTS

From section 4, we can see that the knowledge, $C.r$, of the formation and revision of the requirements that occurs during designing is related to the general context knowledge, $K.gc$, and domain knowledge, $K.dm$, of the designing, knowledge of the motivating needs and desires, $K.nd$, knowledge of requirements formation and definition, $K.r$, and knowledge of any well formed problems, $C.p$, solutions, $C.s$, and their evaluations, $C.e$.

The background and domain knowledge are used to identify possible criteria to be adopted as requirements, or of possible modifications to already adopted criteria. The knowledge of the motivating needs and desires is used to select from the possibilities, and the knowledge of requirements formation is used to properly define and organise the requirements that are chosen, or currently adopted. The knowledge of any existing well formed problems, solutions, and evaluations, is used to identify any inconsistency, incompleteness, ambiguity, imprecision, or impossibility in the existing set of requirements, and results in the discovery of the need for new criteria.

We can see from this Knowledge Level view of requirements formation and modification, according to $K.D_v^E$, that requirements are not simply derived directly from the needs and desires. Nor are they the result of an analysis of the needs and desires. They are the result of a discovery and synthesis process that involves a range of different kinds of knowledge, each of which plays a different role in designing.

5.2 SYNTHESIS OF WELL DEFINED PROBLEMS

The formation and development of well defined problems is also a synthesis process $K.D_v^E$: well formed problems are synthesised from the attempts to operationalise some or all of the current set of design requirements. From the section 4, we can see that the knowledge, $C.p$, of this problem definition and development aspect of the exploration that is at the core of designing, depends upon the general context knowledge,

K.gc, domain knowledge, K.dm, and knowledge of the motivating needs and desires, K.nd, for the knowledge of how design requirements might be operationalised; knowledge of problem defining, K.p, and knowledge of the problems already formed, their status and organisation, C.p; knowledge of the solutions so far generated, C.s, and knowledge of their evaluations, C.e.

The other embedded kinds of general context knowledge, K.cc (knowledge of the customer/client context), K.pc (knowledge of the designing practice context), will also bear on the problem definition. They can be thought of as 'shaping' the synthesis of the well formed problems.

5.3 SYNTHESIS OF LOCAL PLANS

The third kind of synthesis process that can be identified in designing, according $KLD_v^{E_1}$, is the synthesis of the local plans. These are built, either all at once, or incrementally, from the identification (the knowledge) of things that need to be done, changed, or tried as a result of the requirements formation, problem defining, solution finding, and solution evaluations aspects of exploration. The synthesis of the local plans thus depends upon knowledge of all these aspects. It will also be influenced by the design history knowledge, C.dh, as well as supported by the domain knowledge, K.dm, embedded in the general context knowledge.

The synthesis of local plans is a different kind of synthesis activity form the first two, since the synthesised local plans are not necessarily used or executed. They are needed but are not necessarily instrumental in progressing the designing. Their synthesis can, however, provide the basis of useful reflection on the state and form of the ongoing designing. So, even if a particular local plan is not completely executed, or not executed at all, its synthesis can, and often does, play an important role in the designing that produces it.

5.4 SYNTHESIS OF DESIGN DESCRIPTIONS

A necessary outcome of any successful designing must be a design description, or design descriptions. These, as we have seen, are, according to $KLD_v^{E_1}$, built form the solutions to some (but not usually all) of the evaluated solutions of well formed problems. The formation of design descriptions is thus a synthesis process, and it is the kind of synthesis in $KLD_v^{E_1}$ that most closely matches what is called synthesis in designing in other models or theories of designing which only identify one kind of

synthesis activity. It depends upon knowledge of how to form proper design descriptions, K.dd; knowledge of the problems solutions, C.s, the problems they are solutions of, C.p, that can form all or part of an acceptable design that satisfies the motivating needs and desires, together with knowledge of their respective evaluations, C.e. The domain knowledge, K.dm, customer/client context knowledge, K.cc, and design practice context knowledge, K.pc, that are embedded in the general context knowledge, are also important components of the knowledge needed to synthesise proper design descriptions.

6. Discussion

So called 'early models' or 'linear models' of designing divided the design process in to two or three basic sequential phases called analysis, synthesis, and (when included) evaluation. Alexander's model, (Alexander 1964), is perhaps the best known example of this type of model of designing. Here, the analysis is of the design problem. It consists of decomposing the problem into a hierarchical set of subproblems. Alexander's synthesis phase consists of identifying solutions to each of the subproblems, and of putting these component solutions together in a way defined by the hierarchical structure of the subproblems that comes from the analysis. Archer (1963), for example, presented a similar kind of model, which explicitly included evaluation. Apart from this, it too characterises designing as essentially consisting of an analysis stage followed by a synthesis and evaluation stage. Asimow (1962) offered another three phase linear model, in which the last, evaluation, stage was broken down into optimization, revision, and implementation. Still, synthesis was clearly presumed to be a basic and important part of designing, albeit only in one stage.

These early models, now largely discredited and discarded,⁶ gave way to what we can call the cyclic models of designing. These derive from the linear models, but introduce so called 'feedback' paths from later stages back to earlier stages, thus allowing for later designing to give rise to the need to reconsider and possible re-do some activity of an earlier stage. Page (1963) was one of the first to argue against the reality of the linear models and to identify the 'cycling round' nature of designing. Though considered to be more realistic, these cyclic models still presumed synthesis to be one single stage in the overall process of designing.

⁶ Even Alexander later acknowledged the inadequacy of his model, and of the possibility of decomposing design problems in particular, (Alexander 1971).

Despite Broadbent's criticism of these early liner and cyclic models, as mistakenly identifying the design process with the decision sequence, see (Broadbent 1973), another influential model of designing that does essentially this is due to Simon. Simon (1973) argued that design problems are typical examples of what he called "ill structured problems", problems whose structure lacks definition in some respect. He then set out to show that despite this apparent lack of definition, ill structured problems can be solved by a process of decomposition in to a set of well structured problems that are only weakly coupled. Simon's aim, and the conclusion of his paper, was to show that so called ill structured problems, such as design problems (so he said) could be solved using the problem solving methods and techniques being developed in Artificial Intelligence.

Despite appearing to accept, and to be able to treat, the supposed ill structured natured of design problems, Simon's model of designing does not in fact identify explicitly any kind of synthesis task or stage. Following Alexander's model, which also depends upon a problem decomposition stage, we can, however, identify synthesis in Simon's ill structured problem solving with the (re)composition of the solutions obtained to the weakly couple well structured (sub)problems. Simon's concept of the ill structured problem has strongly influenced most work on designing in Artificial Intelligence (AI). It essentially legitimised the idea that designing starts with a problem that needs to be solved, and encouraged researchers in AI to believe that their AI methods and symbolic computation techniques could be made to solve these problems. This is why so called design problems have received quite a lot of attention in AI, but why little if any attention has been given to what is synthesis. In Simon's view of design problem solving there is nothing called synthesis. Most work in AI in design has been concerned with problem solving techniques of one kind or another, either to solve problems completely or to provide some kind of problem solving support or aid to human designers. One of the few exception to this is the work of Chakrabarti and Bligh on a functional synthesis system, (Chakrabarti et al. 1996; Chakrabarti and Tang 1996), though here still, synthesis is essentially treated as a kind of problem solving.

Horst Rittel's characterisation of design problems as "wicked problems" (Rittel and Webber 1972)⁷ and Bazjanac's promotion of this idea in design theory (Bazjanac 1974), forms the basis of a fundamentally

⁷ Rittel first presented his characterisation of wicked problems in the 1960s, but seems not to have published anything on this until his paper with Webber in 1972.

different conception of the nature of designing, and of the kind of situation with which designing has to deal. When dealing with wicked problems one cannot, as Bazjanac puts it, “first define the problem, and then the solution—solutions are generated all the time as problems are formulated” (page 10). Rittel’s model of designing says that the designer continuously goes through alternating sequences of generation of variety (ie, synthesis) and reduction of variety (ie, evaluation). Rittel’s and Bazjanaca’s work both form important parts of the background to the development of Designing as Exploration, so it is not surprising that we can see more of the wicked type of problem solving in Designing as Exploration.

What KLD_v^E does, however, is to make clearer that there is not just one kind of synthesis that must go on in designing, but four different but interrelated kinds of synthesis activities. Designing involves synthesising the design requirements, well defined problems, local plans, and design descriptions, according to KLD_v^E .

Acknowledgments

Hans Akkermans, Morgens Andreasen, Amaia Bernaras, Luciene Blessing, Stephen Buswell, Amaresh Chakrabarti, Nigel Cross, Alex Duffy, Susan Finger, John Gero, Ken Ping Hew, Leslie Kaelbling, Ralf Lossack, Ken Mac- Callum, Mary Lou Maher, Lawrence Mandow, Terry Purcell, Yoram Reich, Norbert Roozenberg, Michael Rosenman, Leon Sterling, George Stiny, Ming Xi Tang, and Tetsuo Tomiyama have all been involved in helpful discussions and criticism of the work reported here. I am grateful and indebted for their time and efforts, but remain responsible for all remaining errors and confusions.

References

- Akkermans, JM; van de Velde, W, Wielinga, BJ and Schreiber, GAT: 1994, Rational: principles underlying expertise modelling, in BJ Wielinga (ed.), *Expertise Model Definition Document*, KADS-II project document KADS-II/M2/UvA/026/5.0, pp. 5-9.
- Alexander, C: 1964, *Note on the Synthesis of Form*, Harvard University Press, Cambridge, MA.
- Alexander, C: 1971, The state of the art in design methodology, in an interview published in *Design Methods Group Newsletter* 5(3): 3-7.
- Archea, J: 1987, Puzzle-making: what architects do when no one is looking, in YE Kalay, (ed.), *Principles of Computer-Aided Design: Computability of Design*, John-Wiley and Sons, NewYork, pp 37-52.
- Archer, LB: 1963, Systematic method for designers, *Design*, pp. 172–188.
- Asimow, M: 1962, *Introduction to Design*, Prentice-Hall, London.
- Bazjanac, V: 1974, Architectural design theory: models of the design process, in WR Spillers (ed.), *Basic Questions of Design Theory*, North-Holland, Amsterdam, pp 3-20.
- Broadbent, G: 1973, *Design in Architecture*, John-Wiley and Sons, NewYork.

- Chakrabarti, A and Bligh, TP: 1996, An approach to functional synthesis of mechanical design concepts: Theory, applications, and emerging research issues, *Artificial Intelligence for Engineering Design, Analysis and Manufacturing* 10: 313–331.
- Chakrabarti, A and Tang, MX: 1996, Generating conceptual solutions on FuncSION: evolution of a functional synthesiser, in JS Gero and F Sudweeks, (eds), *Artificial Intelligence in Design '96*, Kluwer, Dordrecht, pp. 603–622.
- Gettier, EL: 1967, Is justified true belief knowledge? in A Phillips Griffiths (ed.), *Knowledge and Belief*, Oxford University Press, Oxford, pp. 144–146.
- Getzels, JW and Csikszentmihalyi, M: 1976, *The Creative Vision: A Longitudinal Study of Problem Finding in Art*, Wiley and Sons, New York.
- Graham, SL: 1974, The design of programming languages, in WR Spillers (ed.), *Basic Questions of Design Theory*, North-Holland, Amsterdam, pp. 371-381.
- Jones, JC: 1992, *Design Methods*, 2nd Ed., Van Nostrand Reinhold, New York.
- Lawson, B: 1990, *How Designers Think: The Design Process Demystified*, 2nd Ed., Butterworth, London.
- Newell, A, Shaw, JC and Simon, HA: 1958, Elements of a theory of human problem solving, *Psychological Review* 65: 151–166.
- Newell, A and Simon, HA: 1976, Computer science as empirical enquiry: symbols and search, *Communications of the ACM* 19(3): 113-126.
- Newell, A: 1982, The knowledge level, *Artificial Intelligence* 18(1): 87-127.
- Newell, A: 1990, *Unified Theories of Cognition*, Harvard University Press, Cambridge, MA.
- Page, JK: 1963, A review of the papers presented at the conference, in JC Jones (ed.), *Conference on Design Methods*, Pergamon, London, pp. 3-8.
- Quine, WV: 1987, *Quiddities: An Intermittently Philosophical Dictionary*, The Belknap Press of Harvard University, Cambridge, MA.
- Rittel, HWJ and Webber MR: 1972, Dilemmas in a general theory of planning, *Working Paper*, Institute of Urban and Regional Development, University of California, Berkeley, November.
- Schön, DA: 1983, *The Reflective Practitioner, How Professionals Think in Action*, Basic Books, London.
- Schön, DA: 1985, *The Design Studio, An Exploration of its Traditions and Potential*, RIBA Publications, London.
- Schön, DA: 1992, Designing as a reflective conversation with materials of a design situation, *Knowledge Based Systems* 5(1): 3-14.
- Schreiber, G, Akkermans, H, Anjewiesden, A, de Hoog, R, Shadbolt, N, Van de Velde, W and Wielinga, B: 1999, *Knowledge Engineering and Management: The CommonKADS methodology*, MIT Press, Cambridge, MA.
- Simon, H: 1973, The structure of ill structured problems, *Artificial Intelligence* 4, 181-201.
- Smithers, T: 1988, Product creation: An appropriate coupling of human and artificial intelligence, *AI & Society* 2(1): 341-353.
- Smithers, T and Troxwell, WO: 1990, Design is intelligent behaviour, but what's the formalism?, *AI EDAM* 4(2): 89-98.
- Smithers, T: 1992, Design as exploration: Puzzle-making and puzzle solving, *Workshop on Exploration-Based Models Of Design and Search-Based Models of Design*, AI in Design '92, Carnegie Mellon University, Pittsburgh.

- Smithers, T: 1996, On knowledge level theories of design process, *in* JS Gero and F Sudweeks (eds), *Artificial Intelligence in Design '96*, Kluwer, Dordrecht, pp. 561-579.
- Smithers, T: 1998, Towards a knowledge level theory of design process, *in* JS Gero and F Sudweeks (eds), *Artificial Intelligence in Design '98*, Kluwer, Dordrecht, pp 3-22.
- Smithers, T: 2000, Designing a font to test a theory, *in* JS Gero and (ed.), *Artificial Intelligence in Design '00*, Kluwer, Dordrecht, pp 3-22.
- Smithers, T: 2002, On knowledge level theories and the knowledge management of designing, submitted to *DESIGN 2002, Dubrovnik, Croatia, May 2002*.
- Suwa, M, Gero, JS and Purcell, T: 2000, Unexpected discoveries and S-inventions of design requirements: important vehicles for a design process, *Design Studies* **21**(6): 539-567.

A GRAPHICAL NOTATION FOR MIXED-INITIATIVE DIALOGUE WITH GENERATIVE DESIGN SYSTEMS

SAMBIT DATTA

*Deakin University
Australia*

AND

ROBERT F WOODBURY

*Technical University of British Columbia
Canada*

Abstract. In generative design systems, the interaction between the designer and the generative formalism is typically transactional. The system responds to a single query or utterance with one or more results defining the transaction. A neat conceptual separation exists between user tasks and system tasks, rendering the modelling of joint responsibility for generation difficult. To address this problem, a robust basis for combining design interaction with automated or intelligent systems is necessary. We propose a mixed-initiative model of interaction that supports forms of dialogue where both the designer and the formalism share responsibility over the same task. Mixed-initiative enables the role of both user and formalism to be integrated into an incremental model of transaction, where the user and the formalism can share and input data through a common shared resource, on a common shared task. In this paper, we report on the development of a shared graphical notation to support mixed-initiative dialogue situations in generative design systems. We demonstrate the use of this notation in design space exploration and discuss its implications for interaction with generative design systems based on our ongoing work.

1. Introduction

In generative design systems, the interaction between the designer and the generative formalism is typically transactional. The system responds to a single query or utterance with one or more results, defining the transaction. A neat conceptual separation exists between user tasks and system tasks, rendering the modelling of joint responsibility for

generation difficult. A mixed-initiative model of interaction is necessary to support forms of dialogue where both the designer and the formalism may share responsibility over the same task. Mixed-initiative provides a robust basis for combining design interaction with automated or intelligent systems. In such systems, the role of both user and formalism are integrated into an incremental-based model, where the user and the formalism can share and input data through a common shared resource, on a common shared task.

Generative design systems provide formal representations that aid designers in exploring spaces of designs. Woodbury et al (Burrow 1999; Woodbury and Burrow 1999) develop a formal representation of design spaces and unification-based algorithms for generation in the specification and implementation of design space explorers, computer programs for constructing, navigating and exploring spaces of design problems using feature structure theory (Carpenter 1992). They adopt the unification view of incremental generation and provide a representation for the constraints on a design space, relate those constraints to the generation of partial designs and supports intermediate representations of design states. Extending this work with exploration systems, we report on the development of a graphical notation to support mixed-initiative dialogue in generative design systems. We demonstrate the use of this notation in design space exploration and discuss its implications for interaction with generative design systems.

1.1 DESIGN SPACE EXPLORATION

The exploration formalism (Woodbury et al. 1999) is made out of three sets of components: types T, structures F and descriptions D, put into a representation scheme within the building design domain. The types comprising T stand for expressed knowledge of the domain of interest, in this case building design. Structures from F represent models of particular designs, in this case, buildings and/or their components, either physical or conceptual. Structures from F are expressed in terms of the information expressed in T. Descriptions from D are utterances in a formal textual language and correspond to sets of structures from F in the representation scheme.

The primary representational device for design space exploration is provided by the typed feature structure. Symbols are augmented with types and recursively defined attributes called features. Symbol rewriting is based upon the incremental unification of typed feature structures. The problems associated with symbol matching in earlier systems such as combinatorial explosion are resolved by using feature unification in the

generative process. Unification is an efficient algorithm to determine whether one view is more specific than another. The formal generation is incremental and supports stepwise refinement rather than global search for solutions. The use of constraints facilitates the view of domain objects at varying levels of resolution subject to satisfaction. The representation of intermediate states facilitates a collection of partial views of a single domain object at varying levels of specificity. The representation of two structures that possibly represent the same object is recognisable eliminating redundancy in the generative process.

1.2. MIXED-INITIATIVE DIALOGUE

The term mixed initiative refers broadly to methods that explicitly support an efficient, natural interleaving of control by users and automated services aimed at converging on solutions to problems (Allen 1999). Mixed-initiative dialogue provides a robust and well understood basis for addressing the problem of interaction with generative design systems. In this dialogue view of interaction, instructions are entered via spoken input, typed commands or the direct manipulation of graphical symbols. All modalities of input can be interpreted in a common symbolic representation. The same applies to different modes of output, whether generated speech, natural language explanations or graphical visualisation. Mixed-initiative dialogue involves the exchange of initiative in a flexible and opportunistic manner, shifts in focus of attention to meet user needs and the maintenance of shared contexts. This approach has been developed, applied and tested in the areas of AI planning (Ferguson and Allen 1994; Burstein and McDermott 1996), simulation (Amant and Cohen 1997; Cohen et al. 1997), knowledge engineering (Tecuci and Lee 1999), scheduling (Ferguson and Miller 1996) and computational dialogue systems (Guinn 1993).

In order to support mixed-initiative dialogue in design space exploration, we develop a unified notation that enables the designer and the formalism to exchange initiative, maintain shared context and supports control, coordination and communication during exploration. The visual notation is based on the following assumptions,

Typed Feature structures as a representation.

The typed feature structure is a common basis for representing the components of design space exploration as feature nodes. Feature structures provide a rigorous and well-understood basis for representing the objects and entities in a design space as feature nodes.

Integration of input and output modalities.

The typed feature structure can support the multiple input and output modalities arising out of interaction between the user and the generative system. Feature nodes provide and maintain a shared context for dialogue interactions between the user and the formalism. The unification of typed feature structures integrates both generative and interactive actions of exploration. Given this conception of a feature node and its attributes and mixed-initiative dialogue, the components of the visual notation are examined in the next section.

2. The Visual Notation

In this section, we present the visual notation for representing dialogue using feature structures. We employ this visual form of feature structures as a means for encoding dialogue in mixed-initiative systems. First, we describe the straightforward adaptation of feature structure notation to represent feature nodes. The representation is then extended with direct manipulation to support interaction dialogue between the designer and the generative formalism. All communication between the designer and the generative formalism is based on graphical interaction with the notation. This paper examines the development of this visual notation and its support for mixed-initiative dialogue between the designer and the generative formalism.

2.1. FEATURE STRUCTURES AS A REPRESENTATION

The analogy between feature structures and frame-based representations provides a more standard graphical notation for large collections of feature structures. By interpreting each node as a frame, the features on arcs can represent slot labels, and the arcs themselves point to the slot fillers. The only difference being the natural enforcement of unique-value restrictions on slot values in feature structures, since the arcs are modelled by using a partial function $_$. This frame-based interpretation is the standard notation for feature structures used in the description of linguistic fragments modelled by feature structures. The frame-like notation is called attribute-value matrix or "avm" notation. In this notation, each node is represented with the frame delimiters "[" and "]". The frame is annotated with the type of the node, as shown in Figure 1. Re-entrancy, or structure sharing is indicated by tags such as 4. The slots are the features and the values are written next to them, as shown in Figure 2. We develop this "avm" notation to represent the feature nodes and their attributes. First, the elements of a typed feature structure, types, features and their values are mapped to a visual representation of

feature structures using elements of the "avm" notation. The visual representation comprises the representation of types, features, their values and co-references. The smallest element of the visual feature structure notation is the feature-value pair, comprising the relation between a feature and its value, which may be atomic, complex or another feature structure. For example, the feature-value pair, represents the functional relation between the feature, MASS_EL and its a value, which is minimally the type, **massing**.

(1) [MASS_EL : **massing**]

The value of MASS_EL may also be complex, such as a query description, resolution step or function application or external complex datatype. The value of MASS_EL may also be another feature structure. The representation of the value of a feature f, is given by an element called a feature-value map which specifies the relation between a feature and the feature structure that is its value.

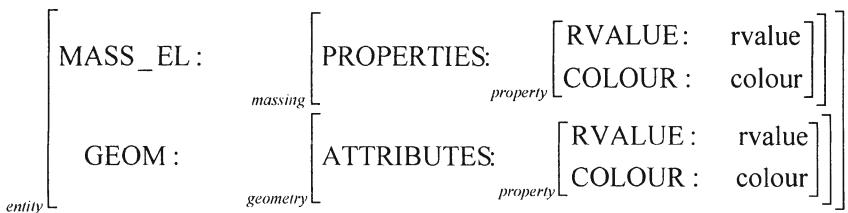


Figure 1. Feature structure in AVM notation. The feature structure entity has the features, mass_el and geom. These features have two substructures of type massing and type geometry.

The feature-value map shown above represents the functional relationship between the type geom and its value. The feature-value map is enclosed by the delimiters "[" and "]" and annotated by its type, geometry. Since feature structures are recursive, the values of feature structures may be another feature structure; the attribute-value notation is easily adopted for a visual representation of the feature-value map. The feature-value map can be conceptualised as a recursive container of entities of type feature-value pair.

Finally, in a visual feature structure, two or more features can share the same information. Each shared structure is represented by co-references, also called tags. The co-reference n denotes an index value and the identity of the node that is shared between on or more feature structures. Co-references and their denotation by indices are a straightforward adaptation from the avm notation.

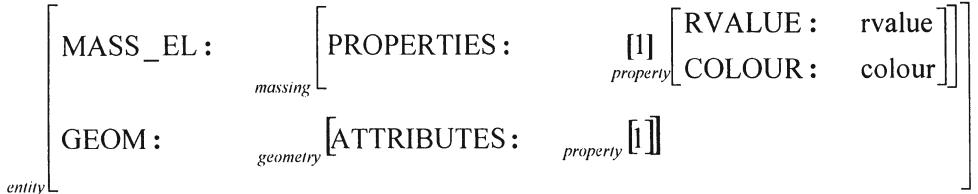


Figure 2. A folded feature structure with co-reference notation. The shared feature structure of type **property** is indicated by the co-reference tag.

2.2. INTERACTION WITH FEATURE NODES

The formal properties of visual feature nodes provide a representation for combining human interaction with the generative process. In this section, we extend the representational properties of feature nodes to incorporate graphical interaction and direct manipulation of feature nodes. The functionality of the visual components of feature nodes can be extended by interaction logic to incorporate complex behaviour attributes into the representation. The feature node representation is extended with an interaction logic providing the ability to unfold feature nodes through zooming and imploding substructures, interaction with conjuncts, disjuncts, functions and commands.

2.2.1 Zooming and imploding substructures

Feature structures can be nested recursively upto arbitrary levels, and can contain many recursively nested substructures. It is necessary to provide a mechanism for folding the nested substructures of a feature-value map to hide their underlying notation and for unfolding the imploded structure, when it is necessary to see the details of a feature value map. In the visual representation, this is realised by incorporating interaction logic on a feature-value map. This is described in Figure 3. The feature-value map can be in one of two modes, either open or closed. This is represented visually by annotating the feature-value map with the symbol, "+".

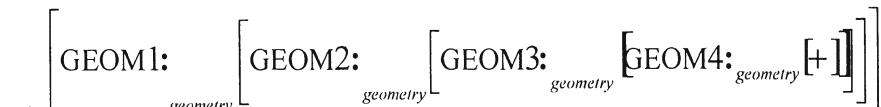


Figure 3. An example of an automatically imploded feature structure during deep containment. The symbol indicates that the feature-value map of type **entity** is hidden automatically after three levels of nesting. User interaction on this node is necessary to unfold these structures, but on regeneration will resume their default behaviour.

The unfolding symbol, "+" shows up in two different situations. A restriction may be placed on the depth of display of a feature-value map. Any substructure in a feature-value map that exceeds that depth is represented by the symbol, "+". This is automatically managed by the dialogue layer and the nesting levels set through preferences. The user can also manipulate the feature-value map interactively. In this situation, the feature-value map will be shown as folded, until it is explicitly unfolded. Thus, the "+" symbols on the feature-value map coming from depth restriction are generated and removed dynamically while the user navigates a feature structure. In contrast, the maps that are unfolded manually need explicit interaction to change their mode. This enables the user to control the level of detail, shown in Figure 4, while zooming and imploding substructures.

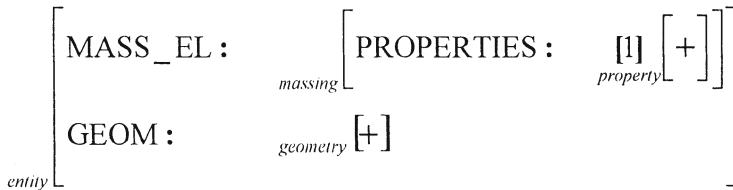


Figure 4. An example of an imploded feature structure. The symbol indicates that the feature-value map of type **property** and **geometry** contains nested substructures which can be unfolded by user interaction.

Co-reference tags in the visual representation indicate structure sharing. They are used in two ways. Firstly, a co-reference is used to annotate a feature-value pair that structure-shares a feature-value map. Secondly, it is used to simplify the visual representation of feature structures, by simple substitution of the shared feature-value map by the co-reference tag, denoted by n. This is shown in Figure 5. The co-reference tag can be substituted by the feature structure it denotes by user interaction. If the feature structure is represented, the co-reference appears outside the feature-value map, as shown in the value of properties. If the co-reference is used to nest the feature structure, it appears inside the feature-value map as shown in the value of attributes.

2.2.3 Interaction with functions and commands

The operations of the description language, conjunctions, disjunctions, implications, lists and function applications, need to be incorporated into the graphical representation of feature structures. Such an interface, must allow for the following, namely,

- _ Extensions to the graphical interaction to enable the display of conjunctions and disjunctions,
- _ Provide the interaction logic for integrating implications, resolution steps and function applications.

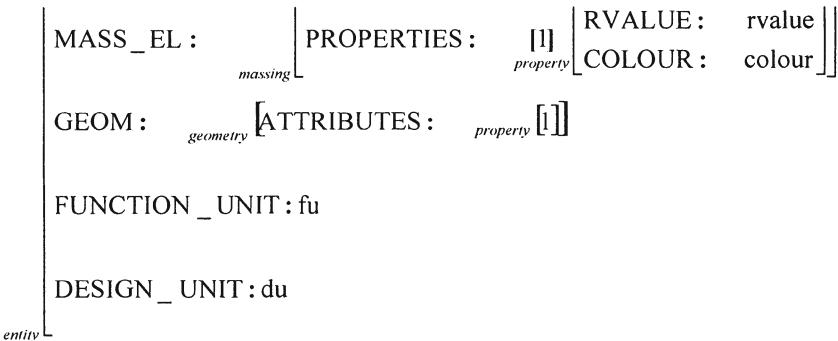


Figure 5. An example of substitution of a feature-value map with a coreference tag,

1. The coreference tag is an index to the nested typed feature structure of type **property** that is shared by the features, PROPERTIES and ATTRIBUTES.

2.2.4 Interaction with conjunctions and disjunctions

Conjuncts and disjuncts can be represented using visual feature structures using the same notation as feature value pairs. In place of the feature labels, the labels *conjunct* and *disjunct* are used, with the values as feature structures. This common representation can be scaled to represent the conjunction of disjuncts and the disjunction of conjuncts. In linugistic attribute-value formalisms, conjuncts and disjuncts are denoted by special delimiters, such as "[" and "]" and their edge names are either omitted, suppressed or obscured. This is not necessary in the interactive representation outlined above. An example of a conjunct of disjunctive feature nodes is shown in Figure 6. User interaction is necessary to resolve the structures associated with the features *mass_el* and *geom* of the feature structure of type *entity*.

The nodes *conjunct 1* and *conjunct 2* are represented as a feature-value map. Each feature-value map has a type *conjunct n*, where *n* is an index over conjunct nodes. Each conjunct feature-value map has four possible disjuncts, each of which are represented as a feature-value pair, whose features are defined by *disjunct n* where *n* is an index over disjuncts.

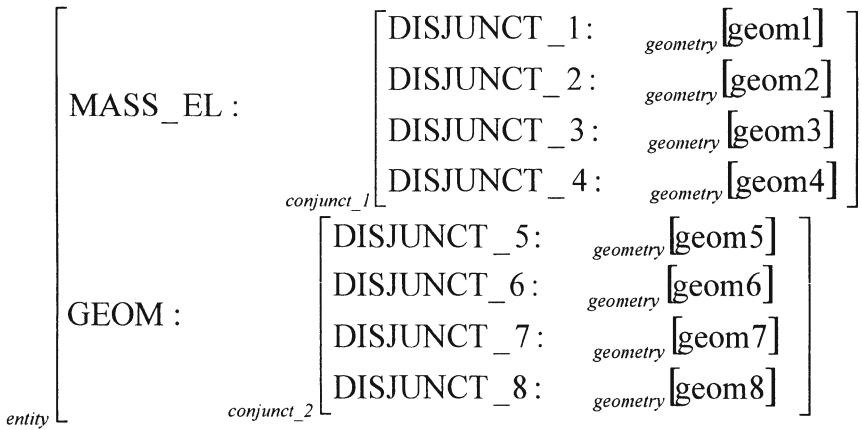


Figure 6. An example of a conjunct of disjunctive feature nodes. The conjunct nodes `conjunct_1` and `conjunct_2` are represented as a feature-value map and each disjunct is represented as a feature-value pair, whose features are defined by `disjunct_n` where `n` is an index over disjuncts.

2.2.5 Incorporating resolution steps

There are two ways to visualise functions for mixed-initiative interaction. Functions can be encoded within the representation such that the functor annotates the feature-value map and the arguments are features. An example of the duality of a function and its arguments with a feature node representation is shown in Figure 7. This representation allows the feature structure to encode the traditional commands found in geometry-based design systems. The specification of a command or function then returns a value, which can be atomic, complex or a feature structure. The use of feature structures to encode functions can also be used to pass commands. The expressiveness of feature structure command representations need to address the possibility of cyclic feature structures and structure sharing. Cyclic feature structures present challenges for the use of feature structures as a representation of visual commands. A cycle arises when following a non-empty sequence of features out of a node leads back to that node, which is a useful property in state-space models and the finite modelling of knowledge (Carpenter 1992). A recognition mechanism is necessary to interrupt infinite loops in the commands.

The restriction on structure sharing is that it is not considered to be a valid part of the command syntax. If co-references occur, the structures they represent are copied uniquely within each command. The second way of visualising functions within feature structures is to encode the functional definition as the value of a feature-value pair. In this scheme,

for a function $\text{append}(X, Y)$ with two arguments, there exists a type function which introduces a feature function, such that its value is a function definition with the syntax, $\text{append}(X, Y)$.

$$\text{append}(X, Y) \Leftrightarrow \underset{\text{append}}{\left[\begin{array}{l} \text{ARG1 : } \underset{\text{arg}}{[X]} \\ \text{ARG2 : } \underset{\text{arg}}{[Y]} \end{array} \right]}$$

Figure 7. Encoding a function as a feature-value map. The function $\text{append}(X, Y)$ which concatenates values can be represented as the feature-value map of type, append and the two features arg1 and arg2 . The features, arg1 and arg2 encode the values X and Y as two feature value pairs.

An example of the feature-value pair representation of a procedural function and is shown in Figure 8. User interaction on this node involves three possible behaviours, the application of the function to an appropriate node results in a new feature structure, consistent with the application.

The unification of a functional node with an appropriate feature structure, results in a new feature structure, following the unification of typed feature structures. Finally, the function can be unfolded into its constituent subparts following the standard interaction and its values subject to exploration.

An example of the latter is shown in Figure 9. The unfolding of a functional representation shows the feature structure dual of the function, $\text{translate}(a, b, c)$ is of type **translate** and the arguments are the three feature value pairs, TX , TY and TZ .

$$\underset{\text{design_unit}}{\left[\begin{array}{l} \text{GEOM : geom1} \\ \text{TRANSFORM : } \langle \text{translate}(a, b, c) \rangle \end{array} \right]}$$

Figure 8. The function $\text{translate}(a, b, c)$ can be represented a feature-value pair and contained within a visual feature structure, with feature transform and value, $\text{translate}(\text{geom})$.

This representation of functions, commands and their arguments extends the visual feature structure notation for user interaction. The behaviour of functions and commands during interaction, namely function unfolding, function application and function unification can be added to the interaction logic necessary for exploration.

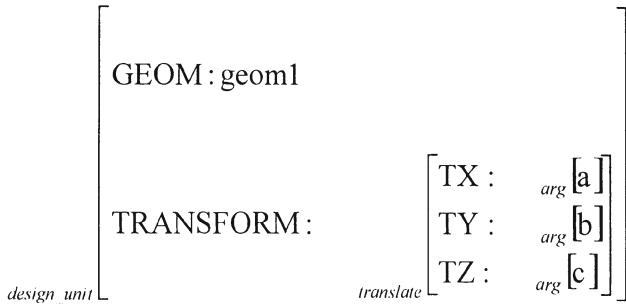


Figure 9. An unfolding of a functional representation shows the feature structure notation of the function, $\text{translate}(a,b,c)$. The type of the function is translate . The arguments are unfolded into the three feature value pairs, TX, TY, TZ.

3. Mixed-Initiative Dialogue

The exposition thus far concentrates on the generic attributes of a visual notation for enabling mixed-initiative dialogue in generative design systems.

The combination of typed feature structure unification, the recursive containment of feature nodes and the interaction behaviour of the notation for feature nodes provides a sound representation for combining human interaction with the generative process. In this section we provide an example of the behaviour of the notation in a mixed-initiative interface implemented in an experimental mixed-initiative system, FOLDS. The visual representation of a description query and translation of the visual representation into an interactive feature node in FOLDS is shown in Figure 10. User interaction in FOLDS involves three possible behaviours, the application of an unfolding operation to an appropriate node results in a new feature structure, consistent with the application. The unification of a type with an appropriate feature structure, results in a new feature structure, following the laws of unification for typed feature structures. Finally, the feature node can be unfolded into its constituent subparts following the standard interaction and its values subject to exploration.

In this example, a description query returns a partial satisfier of type sfc comprising three feature-value pairs, living, porch, dining and their most general substructure nodes. The elements can be expanded and imploded using the triangular arrows, while the selected feature-value pair, size of type vector can be subject to exploration through mixed-initiative. Note that the interactive visual feature node interface as well

as the description interpreter in the background enables communication with the generative formalism. FOLDS comprise a graphical view of the satisfier space and the feature value nodes that it contains, shown in Figure 11. The visual representation of the current state of the dialogue facilitates the growth and traversal of the design space along the attributes of a feature node.

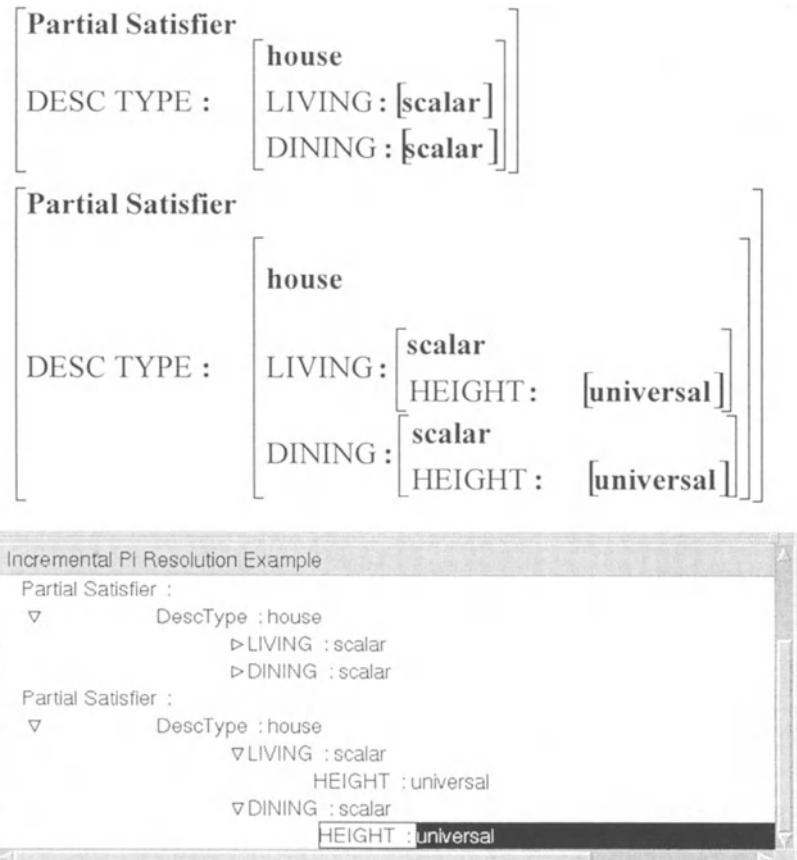


Figure 10. The visual representation of a description query and translation of the visual representation into an interactive feature node in FOLDS .

3.1 RELATED WORK

Linguistic formalisms as well as constraint programming languages use feature structures as a basic data structure. FEGRAMED (Kiefer and Fettig 1995) develops a visual tool that is built to cope with the complexity of

feature structures in constraint-based systems. St. Amant et al (Amant and Cohen 1997) combine the dialogue view of mixed-initiative with direct manipulation techniques in the domain of abstract force simulation and exploratory data analysis. They focus on the ability of an interactive environment to constrain and guide both automated agent behaviour as well as human effort. Rich and Sidner (Rich and Sidner 1997; Rich and Sidner 1998) discuss the design of a collaborative interface agent, which works on a plan with its user and communicates via mixed-initiative dialogue. Hartrum et al (Hartrum and DeLoach 1999) proposes a mixed-initiative system in which humans interact directly with software agents in a collaborative framework for problem-solving. A language processing architecture, based on typed feature structure unification, which supports both speech and gesture is proposed in (Johnston et al. 1997). This architecture allows simultaneous input from speech and gesture recognition interfaces. It is implemented in the QuickSet system (Cohen et al., 1997), a multimodal pen/voice system that enables users to set up and control distributive interactive simulations. The ice, Incremental Configuration Engine project (Zeller and Snelting 1995; Zeller 1997) applies feature logic to the problem of incremental configuration management. Feature logic is used as a unifying formalism for the description of variants and revisions, where sets of versions rather than single versions are the basic units of reasoning. This approach allows for a configuration thread to be constructed by adding or modifying configuration constraints until either a complete configuration or an inconsistency can be deduced. Chien (Chien and Flemming 1996; Chien 1998) addresses the problem of design exploration in interactive contexts and employs the navigation and wayfinding metaphors arising in physical and information spaces to address the problem of interaction with generative design systems. Schulte (Schulte 1997) presents a novel visual constraint programming tool, Oz Explorer, intended to support the exploration, visualization and development of constraint programs. It uses a user-driven and interactive search tree of a constraint problem as its central metaphor.

4. Discussion

In conclusion, the notation we have described in this paper has potential for incorporating mixed-initiative dialogue in generative design systems. The notation extends the modelling of incomplete or partial situations in design space exploration. It enables the user and the formalism to communicate and exchange initiative through interaction.

The notation addresses the requirements for exchange of initiative during exploration. Incrementality provides the basis for developing a model of turn-taking between the formal resolution process and the user during exploration. As the steps of exploration change the design space dynamically, the user and the resolution process can shift their focus of attention dynamically, synchronise their paths to meet the shifts in exploration and provide context for the next step of exploration. The notation imposes a consistent and shared model of dialogue against which dynamic change may occur.

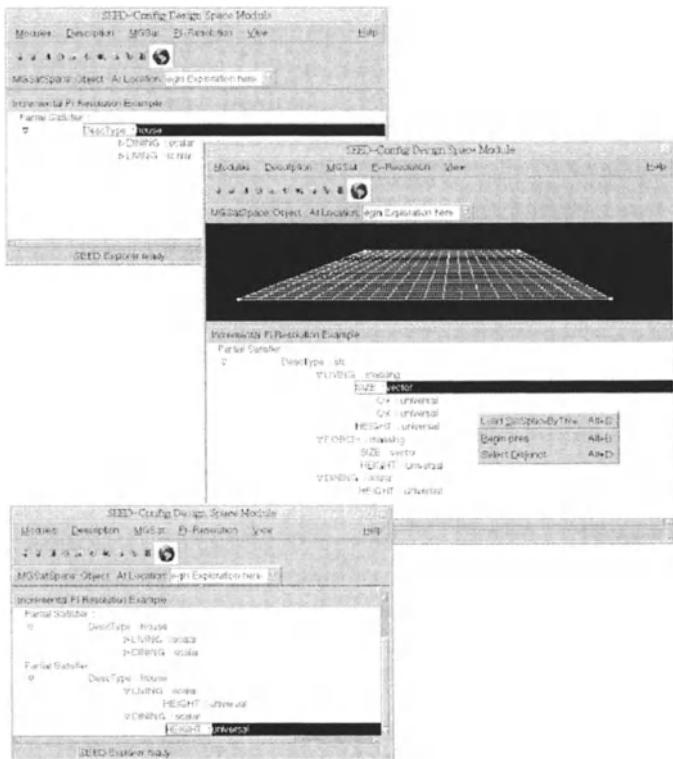


Figure 11. Implementation of interactive unfolding of a feature-value map using the visual representation of a feature node in FOLDS .

The notation schema is computable from linguistic forms of generation based on constraint unification and it is independent from the surface syntactic structure of generative formalisms. While it is based on a logical form, it is not limited to supporting a specific generative formalism. In the example application, logic plays an important role as

an inference mechanism, but the notation itself avoids this regimentation. While supporting dialogue in a principled way, it remains open to multiple modalities of input and output, including voice and gesture. While the work reported in this paper emerged out of research into supporting more intuitive forms of input based on a logical description language, extending this work through research on visual formalisms provides a possible direction of future work.

Acknowledgments

The authors gratefully acknowledge the contributions of Andrew Burrow, Technical University of British Columbia, Vancouver, Canada and Teng-wen Chang, Ming Chuan University, Taiwan.

References

- Allen, JF: 1999, Mixed initiative interaction, *IEEE Intelligent Systems* **14**(6): 14-23.
- Amant, RS and Cohen PR: 1997, Interaction with a mixed-initiative system for exploratory data analysis, *Proceedings of Intelligent User Interfaces*, pp. 15-22.
- Burrow, AL: 1999, *Computational Design and Formal Languages*, Ph.D. thesis, Department of Computer Science. Adelaide University, Australia.
- Burstein, M and McDermott, D: 1996, Cognitive technology: in search of a humane interface, *Issues in the Development of Human-Computer Mixed-Initiative Planning*, Elsevier Science, Amsterdam, pp. 285-303.
- Carpenter, B: 1992, *The Logic of Typed Feature Structures with Applications to Unification Grammars, Logic Programs and Constraint Resolution*, Cambridge Tracts in Theoretical Computer Science, Cambridge University Press, Cambridge.
- Chien, SF: 1998, *Supporting Information Navigation in Generative Design Systems*, Ph.D. thesis, School of Architecture, Carnegie-Mellon University, PA.
- Chien, SF and Flemming, U: 1996, Design space navigation: an annotated bibliography, *Technical Report EDRC 48-37-96*, Engineering Design Research Center, Carnegie-Mellon University, Pittsburgh, PA, USA.
- Cohen, PR, et al.: 1997, QuickSet: Multi-modal interaction for distributed applications, *Proceedings of the Fifth International Multimedia Conference*, ACM Press, pp. 31-40.
- Ferguson, G and Allen, JF: 1994, Arguing about plans: plan representation and reasoning for mixed-initiative planning, *Proceedings of the 2nd International Conference on AI Planning Systems* Chicago, IL, pp. 43-48.
- Ferguson, G and Allen, J and Miller, B: 1996, TRAINS-95: Towards a mixed-initiative planning assistant, *Proceedings of the 3rd International Conference on AI Planning Systems*, Edinburgh, Scotland, pp. 70-77.
- Guinn, C: 1993, A computational model of dialogue initiative in collaborative discourse, *AAAI93 Fall Symposium on Human-Computer Collaboration*, Raleigh, NC, pp. 32-39.
- Hartrum, T and DeLoach, SA: 1999, Design Issues for Mixed-Initiative Agent Systems'. In: Proceedings of the AAAI-99 Workshop on Mixed-Initiative Intelligence. Orlando Fl, pp

- Johnston, M, et al.: 1997, Unification-based Multimodal Integration, *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics*, Madrid, Spain, pp. 281-288.
- Kiefer, B and Fettig, T: 1995, FEGRAMED: An interactive graphics editor for feature structures. *Research Report RR-95-06*, German Research Center for Artificial Intelligence (DFKI), Saarbrucken, Germany.
- Rich, C and Sidner, CL: 1997, COLLAGEN: When agents collaborate with people, in WL Johnson and B Hayes-Roth (eds.), *Proceedings of the First International Conference on Autonomous Agents (Agents'97)*, ACM Press, New York, pp. 284-291.
- Rich, C and Sidner, CL: 1998, COLLAGEN: A collaboration manager for software interface agents, *User Modeling and User-Adapted Interaction* 8(3-4): 315--350.
- Schulte, C: 1997, Oz explorer: a visual constraint programming tool, in L. Naish (ed.), *Proceedings of the Fourteenth International Conference on Logic Programming*, The MIT Press, Cambridge, MA, pp. 286-300.
- Tecuci, GM and Boicu, KW and Lee, S: 1999, Mixed-initiative development of knowledge bases, Proceedings of the AAAI-99 Workshop on Mixed-Initiative Intelligence. Orlando FL.
- Woodbury, R, Burrow, A, Datta, S and Chang, T: 1999, Typed feature structures and design space exploration, *Artificial Intelligence in Design, Engineering and Manufacturing* 13(4): 287-302.
- Woodbury, RF and Burrow, AL: 1999, -resolution and design space exploration, in C Eastman and G Augenbroe (eds), CAAD Futures'99: Computers in Building, Kluwer, Dordrecht, pp 291-308.
- Zeller, A: 1997, *Configuration Management with Version Sets - A Unified Software Versioning Model and its Applications*, Ph.D thesis, TU Braunschweig.
- Zeller, A and Snelting, G: 1995, Handling version sets through feature logic, in W Schfer and P.Botella (eds), *Proceedings of the 5th European Software Engineering Conference (ESEC)*, Vol. 989 of Lecture Notes in Computer Science. pp. 191-204.

WEB-BASED CONFIGURATION OF VIRTUAL PRIVATE NETWORKS WITH MULTIPLE SUPPLIERS

ALEXANDER FELFERNIG, GERHARD FRIEDRICH, DIETMAR JANNACH AND MARKUS ZANKER

*University of Klagenfurt
Austria*

Abstract. An Internet-based Virtual Private Network (IP-VPN) uses the open, distributed infrastructure of the Internet to transmit data between corporate sites. The configuration (network design) for a specific customer network typically requires the selection of network access lines and backbone sections that are provisioned by different organizations in a supply chain. Moreover, when configuring such a network, the given customer requirements (e.g., minimal bandwidth) have to be observed. Within this paper, we show how the (sales-) configuration process for these networks is supported within a framework for distributed configuration. Beside the implementation of an adequate distributed problem solving mechanism based on Constraint Satisfaction with commercial tools, we address the problem of supplier selection and knowledge integration in a Web-based environment for eCommerce: Based on common ontological commitments on representation concepts for the configuration domain, the suppliers can advertise their products and services, whereby the distributed problem solving process involves locating and executing the supplier's configuration service using an open XML-based protocol. We present the architecture of the implemented prototype framework and show the relation of our work to emerging approaches in the fields of Distributed Problem Solving and Semantic Web Services.

1. Introduction

In today's networked economy, effective communication has become a necessity, when remote users (e.g., sales people or distant offices) need easy access to the corporate network and secure connections with the business partners are required. Virtual Private Networks (VPN) extend the company's intranet and are capable of providing such services at reduced cost using the worldwide IP network services and dedicated service provider IP backbones

(Infonetics Research 1997). VPN infrastructures are designed to be flexible and configurable in order to be able to cope with a rich variety of possible customer requirements. Therefore, the establishment of a concrete VPN involves different steps after determination of customer requirements like locations to be connected or specification of required bandwidth: selection of adequate access facilities from the customer site to some entry point to the VPN backbone, reservation of bandwidth within the backbone, as well as configuration of routing hardware and additional services like installation support. Note, that it is very unlikely that all these products and services needed for the implementation of the VPN can be supplied by one single organization but are in general made available by specialized solution providers, e.g., Internet Service Providers, telecommunication companies or hardware manufacturers, Figure 1. Therefore, VPNs are typically marketed by specialized resellers (or telecommunication companies like two of our application partners) that integrate the services of individual suppliers and offer complete VPN solutions to their customers.

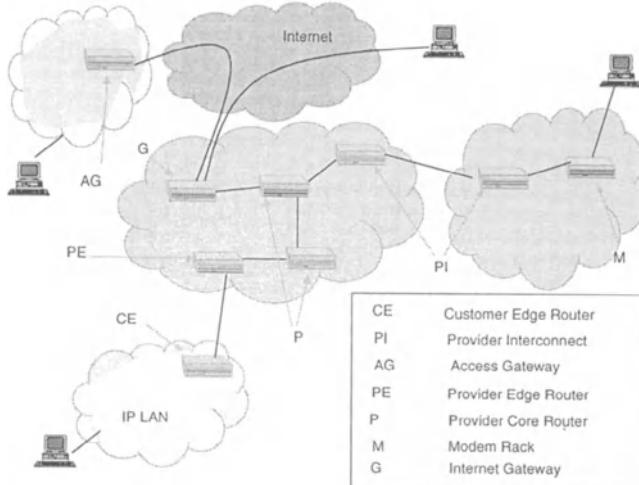


Figure 1. IP-VPN sketch

Efficient sales processes for such complex and configurable products and services require specialized support by sales-force automation tools like product configurators with advanced problem-solving capabilities (Fleischandler et al 1998; Mailharro 1998). Consequently, product configuration has become an important application area for Artificial Intelligence-based techniques in industry and nearly all vendors of Enterprise Resource Planning or Business-to-Business eCommerce systems

have integrated such technologies in their products (Haag 1998; Yu and Skovgaard 1998). However, while there are several commercial product configuration tools on the market, there are some specific requirements in the described application domain that are not addressed adequately by nowadays systems:

- *Distributed configuration*: due to the permanent physical restructuring of the network infrastructure, appearing and disappearing suppliers, and corporate privacy reasons, an approach with one single centralized knowledge base and problem solver is not possible. Furthermore, means must be provided in order to locate the appropriate suppliers and initiate the provided configuration service.
- *Heterogeneity*: For the detailed configuration of the network, the suppliers may employ some specialized software performing complex computational methods for network routing or are using legacy configuration systems. These systems must interact in order to cooperatively solve the overall configuration task.
- *Knowledge integration*: Resellers and network suppliers may use different concepts and terminology for the configuration task. Therefore a common ontology as well as a knowledge exchange mechanism for the application domain is needed.

Exactly these issues are addressed as part of the EU-funded CAWICOMS¹ project and will be discussed in the rest of the paper: First, we describe how standard configuration technology can be extended to cope with Distributed Problem Solving requirements and sketch the implementation framework developed within CAWICOMS. Later on, we discuss how web-based eCommerce between the involved suppliers is done based on information-providing (semantic) web services (McIlraith et al. 2001).

2. Configuration of Virtual Private Networks

At a first glance, the problem of finding routes through a network that connect several given access points and observe specific constraints (e.g., on bandwidth) does not fit to the widely adopted component-connection oriented definition of product configuration from (Mittal and Frayman 1989) which is the basis for most configuration tools.

However, the sales process for VPNs comprises several stages: first, a high-level, abstract design of the VPN is performed, i.e., for each of the

¹ CAWICOMS is the acronym for *Customer-adaptive Web interface for the configuration of products and services with multiple suppliers*. This work was partly funded by the EU through the IST Programme under contract IST-1999-10688. (www.cawicoms.org)

customer sites to be connected we have to select a way to connect the site to some entry point in the IP-backbone network. Typically there are several choices, e.g., on the type of the backbone access (which entry point, which supplier, required protocols etc.), and there are specific constraints (e.g., on compatibility of protocols) that have to be observed. In addition, we have to assure that there is a route within the backbone that connects all the chosen entry points.

The result of this first phase is a coarse network layout and a price which is used to generate an offer for the customer. Once this offer is accepted, the detailed configuration of the VPN can be performed, i.e., computation of low-level technical details like IP-addresses or router configuration parameters. Note that this step requires the usage of (existing) specialized routing or configuration software which is for instance capable of taking into account the current network load of some supplier network.

The following simplified example, Figure 2, will illustrate the rationale of hierarchical configuration of VPNs. The sales engineer for VPNs interacts with the configuration system and interactively enters the customer requirements: The customer sites in *London*, *Paris*, and *Milan* have to be connected, whereby for each connection, he can enter requirements on e.g., bandwidth or latency.

In a first step, the configurator determines a set of adequate access lines from all the available lines that were advertised by the suppliers. Furthermore, a route in the backbone network is computed that interconnects the selected access lines. Note, that this computation is done with the standard functionalities of our configuration software, which also allows for optimization. Optimization can be done according to some objective function like price or number of needed suppliers. Moreover, the search can be guided by user preferences (e.g., according to some business goals) (Junker 2001).

This search process results in the high-level network marked with bold lines in Figure 2 and determines the set of needed suppliers (*BTT*, *TELCO Paris*). In a next step the configurators of these suppliers are contacted in parallel, whereby information about the required components of the network is handed over in the format defined for knowledge sharing. At the supplier sites, details of the network layout are computed (or simply read from a catalog), whereby this typically involves specialized, existing software modules. Those parts of these computations that are relevant for the reseller are transformed back to the common format (according to the ontology) and returned to the main configurator.

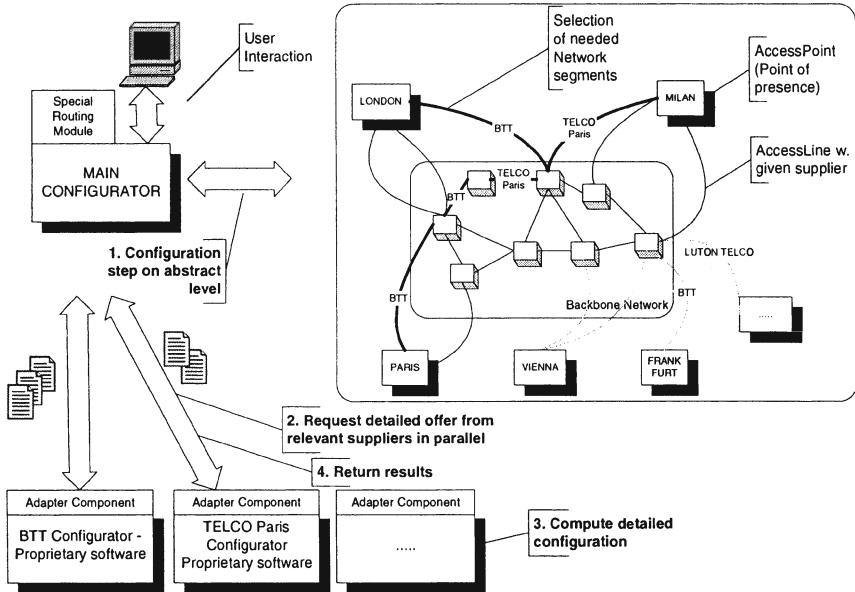


Figure 2. Example scenario

3. Knowledge Sharing

In order to allow communication among several systems, the CAWICOMS framework must be able to support these given business processes and implement that sort of hierarchical configuration process, whereby the integration of existing specialized software modules is a crucial factor. Figure 3 illustrates the overall rationale of knowledge sharing in our framework.

In a first step, the participating companies have to agree on a shared ontology (and terminology) for the domain of IP-VPNs. While this problem is far from being solved in the general case, it was already shown that for specific application domains (eg. *RosettaNet* for Electronic Components; (RosettaNet 2001) such standardization is possible. Moreover, in our application domain the companies involved in the supply chain typically rely on long-term business relations and negotiation phases, which alleviates these integration steps.

The integration of the different product models is supported in the CAWICOMS framework as follows: We use UML as a domain-independent graphical modeling language for the design of the common product model (Felfernig et al. 2001; Rumbaugh et al. 1998). This method has the advantage of being in wide-spread use and comprehensible for domain

experts and is expressive enough for the configuration domain. Moreover, this representation mechanism is independent from the proprietary notation of specific configuration tools. Finally, the models acquired in UML (*Unified Modeling Language*) can be automatically transformed both into the representation of the configuration engine, in our case ILOGs JConfigurator; (Ilog, 2001) as well as into other ontology description mechanisms like DAML+OIL (Fensel et al. 2001). Figure 4 shows a fragment of the VPN product model implemented in the current prototype:

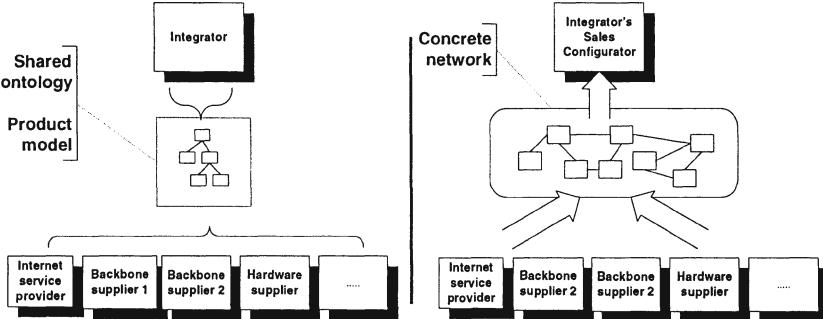


Figure 3. Shared ontology and concrete networks and catalogs

In general, an IP-VPN consists of a set of *AccessPoints* which are connected to *BackboneSections* via *AccessLines* whereby these classes have configurable attributes. The inner IP-backbone network is given by interconnections between the backbone sections (*ProviderInterconnect*). Furthermore, additional routing hardware etc. will be part of the generic product model. This model represents the common ontology for all the companies that are involved in the supply chain.

Once a common understanding of the problem domain is established between the involved parties, knowledge about concrete offers by the suppliers has to be exchanged and integrated. Within our framework, this is done by explicit advertisement of specific instances or subtypes of ontological concepts: As an example, think of a supplier that offers some specific means of access to some backbone section. In order to do that, it publishes the available service to the configuration system of the *integrator* including some concrete values (e.g., *I can provide a managed firewall connection from London to BackboneSection b1 with guaranteed bandwidth of 500kb/s at price X etc.*). Note that this advertisement has to be done automatically by registering the offer to an integration agent at the reseller's site, because of changes in the range of products product and available

resources at the supplier. These offers then form the concrete available network (as well as other services like installation support) for the integrator, which is depicted on the right hand side of Figure 3. As a knowledge representation format for these offerings, we can utilize an XML representation for UML (instance) diagrams. The usage of the emerging DAML+OIL standard is currently evaluated and will be included in future implementations. Note however, that the supplier systems, which can be fully-fledged product configurators, specialized routing software modules, or mere product catalogs, do not have to rely on the same internal knowledge representation or problem solving mechanism like the configurator at the reseller's site. The only requirement at that stage is that the published portions of the e.g., catalog information can be transformed to the common ontology. Typically, the computation of details on the network at the supplier site will include additional reasoning mechanisms or data from other sources like current network load which is not relevant for the integrator in the first place.

4. Distributed Configuration

In recent years, some major advances have been made in the field of Distributed Artificial Intelligence and in particular with respect to techniques for Distributed Constraint Satisfaction (Yokoo 2001) and Multi-Agent Systems. In addition, the field is continuously pushed forward by the rapid growth of world-wide Business-to-Business eCommerce and the need for seamless supply-chain integration. Having discussed the problem of knowledge integration and –exchange in the previous section, we will now focus on the techniques and algorithms employed for distributed problem solving in the CAWICOMS framework with respect to specific requirements for our application domain. We have to state in advance that the general conditions of our domain-independent framework for distributed configuration include both (re-)use of existing technologies as well as openness for integration of other systems². Therefore, two quite different application domains were chosen for evaluation, whereby one addresses configuration of VPNs as described here and the other the more traditional product configuration of telecommunication switches.

We base our algorithms on Constraint Satisfaction techniques, which have shown to be adequate for solving configuration problems with regard to expressiveness, efficient problem solving, and declarative knowledge representation (Fleischanderl et al. 1998; Mailharro 1998). While there are already algorithms available for Distributed Constraint Satisfaction (Silaghi

² For an overview of the complete project, please refer to (Ardissono et al. 2001).

et al. 2000; Yokoo 2001), these approaches have some characteristics that do not fit into our framework too well:

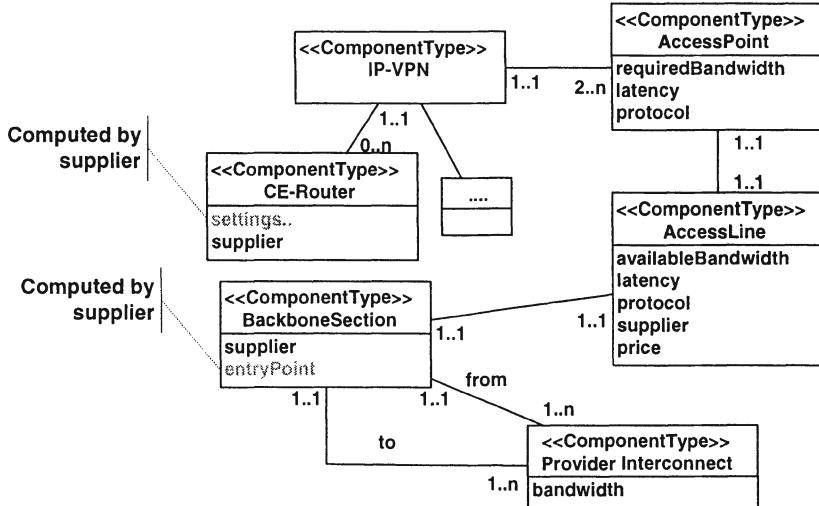


Figure 4. Product model fragment for VPNs in UML

- they require the usage of specialized search mechanisms like Asynchronous Backtracking or Weak-Commitment Search not provided by commercial tools.
- in many cases, the problem is simplified to agents that only handle binary constraints and one single local variable (although, in general, these approaches can be extended).
- they do not take the given supply chain structure and existing business processes into account for (configuration) problem solving.

The approach taken in the CAWICOMS framework takes these considerations into account by allowing the integration of several configurators in the supply chain, whereby – compared with other approaches – there is some predefined, typically tree-structured order and a client-server relation between the involved systems; at the inner nodes of the tree-structured supply chain setting, dedicated configurators facilitate the integration of serving configuration systems. At the top of the supply chain a *main configurator* communicates with the user via an interaction module.

The main idea of integrating the involved systems relies on sharing of variables in terms of Constraint Satisfaction and sharing of components in the sense of (Mittal and Frayman 1989). Configurators may share parts of their configuration knowledge with neighboring systems; more specifically, configurators may publish relevant parts of their knowledge to others that

are at the next higher level in the supply chain. This knowledge is then incorporated (including the definition of inter-agent constraints) into the product model at the next higher level and we markup those "imported" chunks of knowledge at the higher level with information, which supplier is *responsible* for finding a solution for that sub-problem. During the configuration process, search starts at the top level of the supply chain, e.g., using an extended standard CSP search algorithm like implemented ILOG JConfigurator (Ilog 2001); during the search for solutions for the local configuration problem at one node we may encounter a sub-problem which references to a suppliers knowledge base.

Figure 4 depicts this situation, where some of the attributes of the integrated product model are shaded, i.e., we know that the values for these variables have to be determined by some supplier configuration systems. In the example, these are technical details that cannot/should not be computed by the reseller's configurator, because the rationale of the computation is both complex (needs specific algorithms) and confidential to the supplier.

Obviously, it is not sufficient that each of the involved configurators finds a solution to its local sub-problem but we have to find an overall solution that satisfies both the user requirements as well as all the given constraints between the involved configurators. In Felfernig et al. (2001a), the overall conditions for finding a globally consistent solution to such a distributed problem are described in terms of a logical model of distributed configuration. In general, means of conflict resolution have to be provided in cases, where a solution that is consistent with the local configuration knowledge is inconsistent with other partial solutions of configurators that share some variables. As an example, in Felfernig et al. (2001b) we describe such a sound and complete algorithm for distributed configuration of telecommunication switches, which is based on synchronous backtracking to ensure global consistency. This algorithm was implemented by extending to the standard forward-checking backtracking search procedure of ILOG JConfigurator and takes the mostly sequential nature of the task into account³.

In principle we can use that algorithm also for the configuration of VPNs, according to our intention of being domain-independent with respect to our solving techniques: we start selecting lines in the abstract network using the configurator at the reseller's site, request a detailization of the solution from the supplier, integrate the results and continue with the next

³ Note, that *parallel* computation (like it is in Distributed CSPs) is not a natural way of problem solving in some domains.

access line. In case of inconsistencies that may arise during integration of the detailed results we backtrack and search for another solution.

In the case of VPN configuration, however, we can apply a hierarchical and parallel approach, where we compute the complete (optimal) solution at the abstract level and then request the configuration details from all the involved suppliers in parallel. This can be done under the assumption that the suppliers will always find a solution given the parameters for their subproblem and that the returned results do not violate any constraints when integrated into the overall solution, which is a reasonable assumption according to the requirements of our application partners.

A further requirement for the application domain can be tackled with that approach: When requesting a solution (or an offer) from a supplier it may be the case that the configuration process at the supplier side requires some (longer-lasting) human interaction because e.g., some internal business processes have to be initiated. This implies that the overall configuration can only be completed after all the results from all involved suppliers are returned. Consequently, after sending out the requests for solutions to the suppliers in parallel, we have to wait until the last solution is replied before notifying for instance the sales representative that the configuration is completed. This requirement is addressed in our framework by supporting long-lasting configuration sessions. In some cases, however, it may also be sufficient to present the user the results of the high-level network for the offering phase.

5. Implementation

The CAWICOMS framework for distributed configuration is implemented on Sun's JAVA-based J2EE platform that supports component-oriented development and portability and provides basic standard functionalities for Internet-based programming like naming services or load balancing. *Java Server Pages* are used for the generation of adaptive interfaces which is part of the project but out of the scope of this paper.

Reasoning. The core reasoning mechanism for distributed configuration is implemented by extending the commercial domain-independent configuration engine *JConfigurator* from ILOG⁴. Note however, that these extensions were done without changing the core mechanisms of the configurator engine but only by using the built-in extensibility features. This was done in order to keep the solution as independent as possible from specific vendors. The main requirement for a configurator to be usable in our framework is that one can perform user-defined procedures at certain

⁴ www.ilog.com

points (e.g., when trying to solve a goal) in the search process. So whenever the configurator (or simply a constraint solver) tries to configure a certain component instance, one module of our framework checks whether the component has to be configured by a supplier by querying a database. If so, we have to look up the supplier and request a solution for the sub-problem and integrate the results in the local solution space. Finally, we added some generic interface to plug-in specialized domain-specific algorithms for finding a route within the backbone network, in our case a variant of the minimum spanning tree algorithm.

Knowledge Acquisition & Representation. The CAWICOMS workbench includes a *Knowledge Acquisition Workbench* which must be general enough to model a wide range of configuration problems. As already described we made excellent experiences by using UML (and the built-in *Object Constraint Language*) as a language for expressing configuration problems on a conceptual level and we have shown that it can also be used to model problems like in the VPN domain that do not fit the classical scheme of component-connection oriented configuration in the first place. In the current state of the project, we take the XMI representation of UML (which can be generated by most commercial UML tools like *Rational Rose*) and transform it automatically into an intermediate XML representation which is then used to incorporate additional knowledge both about potential suppliers as well as personalization information for the user-interface generation task. Again, depending on the specific configuration tool that is used for the application domain, we have to write adapter components to transform the knowledge into the representation of a specific tool, Figure 5. Future steps include the usage of DAML+OIL (Fensel et al. 2001) as knowledge representation format.

Knowledge exchange during the configuration process. During the distributed configuration process, the configuration agents have to exchange information about the current state of the search process. On the one hand, when requesting a solution for a sub-problem from a supplier, the requesting agent has to inform the supplier both of the actual requirements as well as of the intermediate results of the inference process (e.g., domain reductions in CSP terminology).

On the other hand, the results computed by the supplier configurators have to be returned to their clients. Note, that in the configuration domain these results will be complex data structures that reflect the computed configuration including instantiated components, attribute values (or domains) as well as connections between the individual components. In UML terminology, we have to exchange instance models (containing individual objects rather than classes) whereby these instance models have

to conform the structure defined in the product model, i.e., the static structure diagram like Figure 4.

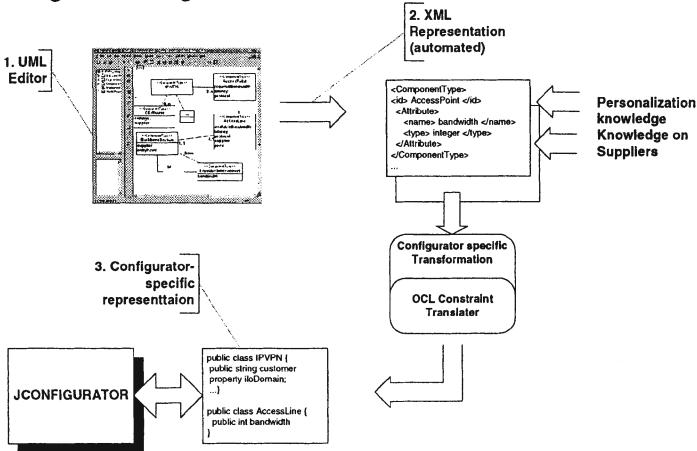


Figure 5. Schematic knowledge acquisition process

Finally, these pieces of information also have to be communicated via the User Interface component, through which customer requirements can be stated and results will be retrieved and presented. From a low-level technical point of view on communication in the Internet, there are several techniques available that allow communication between processes over the network. The most prominent ones are CORBA, see e.g., (Mowbray and Ruh 1998), Microsoft's DCOM, or Java-based Remote Method Invocation (RMI). These approaches allow the exchange of complex data structures and CORBA and DCOM allow for interoperability between different programming languages. However, these techniques are still programming approaches that offer remote computation but do not cope with the requirements of offering semantic services, that can be located and accessed on basis of their functionality. Moreover, these techniques are often rather complex to use and rely on low-level TCP/IP communication which may contravene a company's security policy.

With the development of SOAP (Simple Object Access Protocol – see <http://www.w3c.org>) these limitations are overcome by defining a XML-based protocol for information exchange over the Internet. This protocol aims at providing means for describing what is in a message and how to process it, conventions for remote procedure calls as well as encoding rules for application-defined data types. The goals therefore include the possibility to add semantics to messages as well as a platform-independent communication mechanism over HTTP.

The approach taken in the CAWICOMS framework relies on the same mechanisms, whereby some specific extensions for the domain are incorporated, Figure 6. Communication between configurators as well as with the user interface application is based on *ILOGs WebConnector* protocol. This protocol (and toolkit) was basically designed as a generic protocol to publish and edit complex data structures over the Web, and is not limited to the domain of product configuration. It defines an API to manipulate complex data structures whereby the calls to the API can be done in a SOAP-compliant XML format and the results are again returned in XML (i.e., XML-Schema) which can be further transformed for e.g., presentation to the user. ILOGs JConfigurator was integrated with the WebConnector toolkit, which required the definition of how manipulations to the data structures on the abstract layer (visible for the clients) are mapped to the internal object model of the configurator. Thus, the configurator can be accessed via the WebConnector protocol using XML messages and all the transformations from and to the XML format are performed automatically. In our setting of distributed configuration we use this protocol for communication not only with the user interface but also for communication between individual configuration systems.

The built-in extensibility features of the WebConnector toolkit were used to add product configuration-specific methods and transaction management. Finally, a module was developed that automatically adapts the contents of the messages according to the views on the product model of the involved configurators: This is done because according to our notion of sharing knowledge, the cooperating configurators have only restricted knowledge of the product model of each other.

Another design goal that should be reached with this approach is openness to different configuration tools or legacy systems. As depicted in Figure 6, supplier configurators (or existing systems) do only have to support the open WebConnector protocol in order to participate in the distributed (network) configuration process. However, in these cases, adapter components have to be developed that map the XML-messages onto the internal representations of these systems.

6. Distributed Configuration (of VPNs) as Semantic Web Service

The enormous growth of the Internet and related businesses that operate on top of the Web causes an increasing demand for attaching meta-data to the information and services provided, once we do not solely view the Web as a large repository for text and images (McIlraith et al. 2001). Such annotations – that have to exceed the descriptive capabilities of simple

keywords – aim at describing the semantics of the pieces of information available on the Web, thus giving us the possibility to perform more sophisticated operations, i.e., high-level queries on data as well as identification, location and execution of available services.

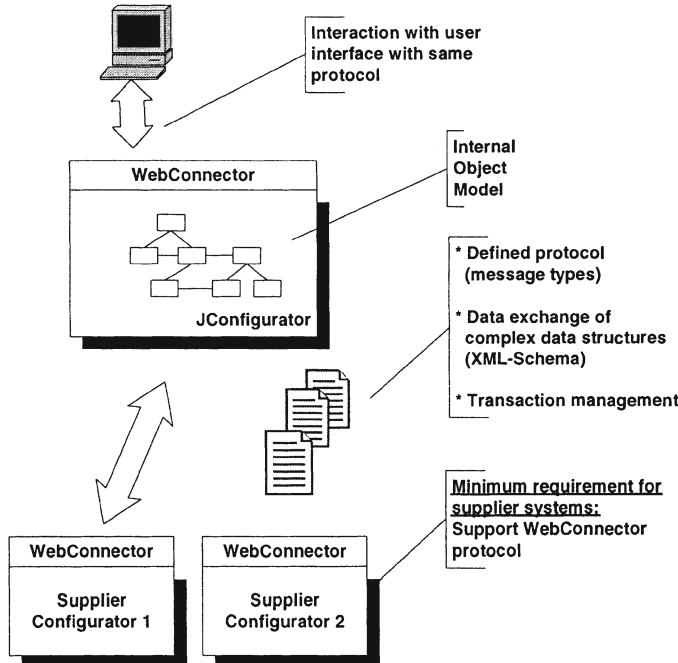


Figure 6. XML-based information exchange

Especially for the case of services that are provided online (e.g., flight reservation or travel planning) we face the problem that these services are only accessible via custom-made Web interfaces and the usage of the services is limited to humans that interact with the systems. Moreover, once one has solved the problem of finding such a service, in general, no standard way of interacting (protocol) or a common terminology is available for these services.

However, with the rapid emergence of Business-to-Business (B2B) eCommerce and electronic market places, the automation of the interaction between the involved systems has become inevitable. This automation involves both the definition of a standard ontology containing the concepts of the business domain as well as the definition of services that are to be provided in the domain. In recent years, a lot of XML-based (pseudo-) standards like Commerce XML (cXML) or Common Business Library

(CBL) for B2B applications have evolved that support communication on the basis of standardized terminologies, data exchange formats and a set of predefined operations like e.g., order placement or other typical business transactions. So while these standard operations are quite well understood and supported in today's systems, i.e., the meaning of an purchase order is basically the same for most businesses, there are no mechanisms available to describe the semantics of non-standard and domain-specific world-altering operations like reservation of an airline ticket like described in (McIlraith et al. 2001). The same holds for the domain of configurable products where several interaction steps like the definition of requirements or search for a suitable configuration may be required. This problem is even harder in cases where configuration of products may occur on several occasions in a supply chain and different (configuration) services are required to compute a satisfying solution.

The further development of the CAWICOMS framework aims at providing distributed configuration services in supply chain settings based on an open Semantic Web environment. While much of the work done in the emerging Semantic Web community is still based on theoretical considerations, the application domain of distributed product and service configuration can serve as a test bed for adding intelligence to the Web: Firstly, there is real industrial demand and business opportunities in this domain, and secondly, the domain is sufficiently restricted and understood in order to allow the implementation of viable solutions. However, the transformation of the distributed configuration process into a Web Service in the sense of (McIlraith et al. 2001) requires more than just the adoption of emerging standards on the technical level, like the usage of SOAP-conforming messages and DAML+OIL for knowledge representation or techniques for service localization. It rather requires a common understanding of

- a) the configuration problem itself, i.e., the semantics of the concepts used to describe configuration problems,
- b) the services required to solve a configuration problem as well as the semantics and consequences of service execution,
- c) the means of exchanging information during the configuration process, i.e., how do we represent complex data structures and what is the meaning of individual pieces of information.

In recent years, both the academic (Mittal and Frayman 1989; Peltonen et al. 1998) and the industrial communities (Fleischanderl et al. 1998; Junker 2001; Mailharro 1998) involved in the product configuration domain have made significant advances in establishing a common component-connection

oriented view of the configuration task. The theoretical foundations in the CAWICOMS project are based on a logic theory of the (distributed) configuration task. This allows both for precise semantics for the employed concepts as well as independence from specific knowledge representation and reasoning mechanisms⁵. In general, a configuration problem consists of a domain description that describes the available component types, their attributes and connection points and some specific user requirements for the actual configuration task. The configuration result can be described by grounded literals, whereby we define the set of predicate symbols that describe a configuration result as CONL, (Felfernig et al. 2000).

Definition (Configuration problem): A configuration problem is described by a triple $(DD, SRS, CONL)$, where DD and SRS are sets of logical sentences and $CONL$ is a set of predicate symbols. DD represents the domain description, SRS the system requirements specification for a configuration problem instance. A configuration $CONF$ is described by a set of positive ground literals whose predicate symbols are in $CONL$. \square

Based on this definition, we can now describe in which situation $CONF$ is a solution for the configuration problem:

Definition (Consistent configuration): Given a configuration problem $(DD, SRS, CONL)$, a configuration $CONF$ is consistent iff $DD \cup SRS \cup CONF$ is satisfiable, \square

To ensure the completeness of a configuration, additional formulae for each symbol in $CONL$ have to be introduced to $CONF$. We denote the configuration $CONF$ extended by these axioms with \overline{CONF}).

Definition (Valid configuration): Let $(DD, SRS, CONL)$ be a configuration problem. A configuration $CONF$ is valid iff $DD \cup SRS \cup \overline{CONF}$ is satisfiable. \square

In Felfernig et al. (2001a) this logical theory was extended to the case where several configuration agents solve a distributed configuration problem, whereby the individual agents solve sub-problems of the overall problem and rely on local knowledge bases (domain descriptions).

Definition (Distributed configuration problem): A distributed configuration problem for n different configuration agents is described by a triple $(DD_{set}, SRS_{set}, CONL)$ where

$$DD_{set} = \{DD_1 \dots DD_n\} \text{ and}$$

$$SRS_{set} = \{SRS_1 \dots SRS_n\}.$$

Each element of DD_{set} and of SRS_{set} is a set of logical sentences and $CONL$ is

⁵ Note, that this does not require the involved configurators to rely on such a representation mechanism.

a set of predicate symbols. For $k \in \{1 \dots n\}$, DD_k corresponds to the domain description of the configuration system k and SRS_k specifies its system requirements. A configuration $CONF$ is described by a set of positive ground literals whose predicate symbols are in $CONL$. \square

Definition (Valid solution to a distributed configuration problem): Given a distributed configuration problem $(DD_{set}, SRS_{set}, CONL)$, a configuration $CONF$ is valid iff $DD_k \cup SRS_k \cup \overline{CONF}$ is satisfiable $\forall k \in \{1 \dots n\}$ \square

In principle, the involved configurators can be seen as independent modules that are able to solve their individual configuration tasks. We define a property called defined interfacing, where all involved configuration systems employ disjoint sets of predicate symbols except for those contained in $CONL$. This way configurators can exchange (partial) configuration results based on shared predicate symbols in $CONL$. Based on this property, we can define precisely under which circumstances the distributed solving of the configuration task generates equivalent solutions to a centralized approach.

Theorem: Let $(DD, SRS, CONL)$ be a configuration problem and $(DD_{set}, SRS_{set}, CONL)$ a distributed configuration problem with defined interfacing where

$$DD = \bigcup_{dd \in DD_{set}} dd \text{ and } SRS = \bigcup_{srs \in SRS_{set}} srs.$$

$CONF$ is a valid configuration for $(DD, SRS, CONL)$ iff $CONF$ is a valid solution for the distributed configuration problem $(DD_{set}, SRS_{set}, CONL)$. \square

For the **Proof** and further details, see Felfernig et al. (2001a).

Finally, when solving the distributed configuration problem, agents exchange partial solutions to come to an overall configuration. During the search process it could be discovered that some of the exchanged partial solutions are in conflict with the local knowledge base and conflict resolution among agents must be initiated.

This general definition of the (distributed) configuration task serves two purposes in our framework. Firstly, we can extend the formalism by introducing the extensible set of predicate symbols $CONL$ for component-connection oriented configuration, i.e., $CONL=\{type/2, conn/4, val/3\}$ for describing component instances, connections and attribute valuations as the common interface for exchanging (partial) configurations. Other representations (e.g., those of commercial tools) can be mapped to that logical representation. Secondly, if we standardize the way the components (and their attributes) itself are to be described in DD , we can automatically transform several other representation mechanisms for product models (like *UML*) into the logical framework thus yielding precise semantics for the

employed modeling concepts (Felfernig et al. 2000). In addition, the logical framework does not explicitly require some specific reasoning mechanism because it rather describes the conditions under which a configuration instance is a valid solution for a distributed problem. The reasoning task can e.g., be accomplished – as in the CAWICOMS framework – by some specialized constraint solver.

Finally, we need to address the description of the actual service a configurator can offer to its clients in a distributed environment, whereby two different pieces of information have to be provided:

1. *which products* can be configured by the configurator?
2. *what capabilities* (services) does the configurator offer?

The first point is related to the advertisement of the range of products the configurator is capable to configure. While there are approaches emerging to define a unique world-wide catalog and categorization of products like UNSPSC or eCl@ss in B2B eCommerce (e.g. Fensel et al. 2001b), up to now there is no agreed-upon industrial standard. Moreover, most of these classification schemes do not support products that are parametrizable which is typical for configurable artifacts. The approach taken in CAWICOMS, which relies on shared ontological commitments for configuration domain specific representation concepts and advertisement of services, was already described in Section 3. Future work in CAWICOMS will therefore include an approach to integrate these techniques with forthcoming classification standards. However, the way of ontology and knowledge integration between the involved partners strongly depends on the type of businesses: For short-term businesses of standardized (low-priced) products (like purchasing an airline ticket, (McIlraith et al. 2001), the usage of world-wide classification standards will be a must in order to allow the participation of a large number of possible providers of such products. In contrast, in highly complex domains like the provision of Virtual Private Networks, supply chain integration will typically rely on longer lasting business relations among the partners; therefore, we suppose that ontologies for specific application domains or even only for the involved partners can be established. Nonetheless, better techniques and tools for ontology construction and integration will be needed and within the scope of future development of CAWICOMS.

Another point targeting service description relates to the individual capabilities of the involved configuration systems. Note, that we do not want to restrict our framework to some specific tool or specific constraint solving algorithms, because openness towards legacy configurators is an important prerequisite. In the framework described in Felfernig et al. (2001a) we base distributed problem solving on exchanging partial

configurations and conflicts. Furthermore, we define basic communicational capabilities a configurator has to provide in order to participate in a distributed configuration process. Note, that the very semantics of these method calls have to be described precisely in order to allow successful cooperation. Therefore, we rely on a logic theory of configuration, that describes the semantics of the required services unambiguously. For a simple distributed algorithm using a facilitating agent (Felfernig et al. 2001a) we need some basic message types exchanged between configuration agents, e.g.:

- *requesting a solution*: As an input, a partial configuration (containing the user requirements) is given. The return value of the call is either a configuration that is consistent with the local knowledge base or else a notification if no solution can be found and ideally the conflict.
- *adding a conflict*: In order to avoid infinite processing loops, conflicts are exchanged among agents and incorporated into their view of the overall problem solving status.

Furthermore, there will be some additional methods for agent initialization as well as additional parameters to control the search process, since – as opposed to simple web services where the individual calls to some agent will be independent – the distributed configuration process will involve a series of subsequent calls to an agent (e.g., in case of backtracking) in a configuration session.

7. Conclusions

We have presented a framework for distributed (sales) configuration and subsequent offer generation that is currently being developed in the EU-funded CAWICOMS project. While the general framework is designed to be applicable to many different domains, we have focused on the configuration of Virtual Private Networks because of its specific requirements on the configuration process and its industrial importance.

After describing the specific properties of the application domain, we sketched an approach for knowledge sharing among configurators in a supply chain based on a common ontological commitments on concepts for problem representation and advertisement of configuration capabilities. After discussing the requirements for distributed and cooperative problem solving in a web-based environment, we presented the current implementation of the CAWICOMS framework that is built using state-of-the art component technology. Distributed configuration is performed by extending an industrial-strength constraint solver for the problem solving process and by using an open, XML-based knowledge exchange mechanism.

In the final section, the relation of the problem setting to the emerging field of the *Semantic Web* is emphasized. Based on a formal definition of the distributed configuration problem with precise semantics, we outlined the future extension of the framework in order to allow the provision of configuration services in an open web-based environment.

References

- Ardissimo, L, Felfernig, A, Friedrich, G, Jannach, D, Schaefer, R, and Zanker, M: 2002, Intelligent interfaces for distributed web-based product and service configuration, N Zhong, Y Yao, J Liu, and S Ohsuga, (eds), *Proceedings of Web Intelligence (WI-2001), Lecture Notes in Artificial Intelligence*, Springer Verlag, Berlin, pp. 184-188.
- Felfernig, A, Friedrich, G and Jannach, D: 2000, UML as domain specific language for the construction of knowledge-based configuration systems, *International Journal of Software Engineering and Knowledge Engineering* **10**(4):449-470.
- Felfernig, A, Friedrich, G and Jannach, D and Zanker, M: 2001a, Towards distributed configuration, in F Baader, G Brewka, and T Eiter, (eds), *Proceedings of KI-2001, Joint German/Austrian Conference on AI*, Lecture Notes in AI, Springer Verlag, Vienna, pp. 198-212.
- Felfernig, A, Friedrich, G, Jannach, D, Russ, C and Zanker, M.: 2002, Multi-site product configuration of telecommunication switches Proceedings of 20th IASTED Conference on Applied Informatics, IASTED, Innsbruck, to appear.
- Fensel, D, Horrocks, I, van Harmelen, F, McGuinness, DL and Patel-Schneider, P: 2001, OIL: an ontology infrastructure for the semantic web, *IEEE Intelligent Systems* **16**(2): 38-45.
- Fensel, D, Ding, Y, Omelayenko, B, Schulten, E, Botquin, G, Brown, M and Flett, A: 2001, Product data integration in B2B E-commerce, *IEEE Intelligent Systems* **16**(4): 54-59.
- Fleischanderl, G, Friedrich, G, Haselböck, A, Schreiner, H, and Stumptner, M: 1998, Configuring large systems using generative constraint satisfaction, *IEEE Intelligent Systems* **13**(4): 59-68.
- Haag, A: 1998, Sales configuration in business processes, *IEEE Intelligent Systems* **13**(4): 78-85.
- Infonetics Research: 1997, *Virtual Private Networks White Paper*, <http://www.infonetics.com>.
- ILOG JConfigurator User Manual, 2001, ILOG S.A. France, 2001, <http://www.ilog.com>.
- Junker, U: 2001, Preference-programming for configuration, *Working Notes of IJCAI'01 - Configuration Workshop*, Seattle, Washington, pp. 50-56.
- Mailharro, D: 1998, A classification and constraint-based framework for configuration, *AI EDAM*, Vol. 12(98), Cambridge University Press, Cambridge, pp. 383-397.
- McIlraith, S, Son, TC and Zeng, H: 2001, Semantic web services, *IEEE Intelligent Systems* **16**(2): 46-53.
- Mittal, S and Frayman, F: 1989, Towards a generic model of configuration tasks, NS Sridharan, (ed.), *Proceedings of IJCAI'89*, Morgan Kaufmann, San Mateo, CA, pp. 1395-1401.
- Mowbray, TJ and Ruh, WA: 1998, *Inside CORBA*, Addison-Wesley, Reading, MA.
- Peltonen, H, Männistö, T, Soininen, T, Tiilonen, J, Martio, A and Sulonen, R: 1998, Concepts for modeling configurable products, *Proceedings of European Conference*

- Product Data Technology Days 1998*, Quality Marketing Services, Sandhurst, pp. 189-196.
- RosettaNet WebPages: 2001, <http://www.rosettanet.org>.
- Rumbaugh, J., Jacobson, I. and Booch, G.: 1998, *The Unified Modeling Language Reference Manual*, Addison-Wesley, Boston, MA.
- Silaghi, M., Sam-Haroud, D. and Faltings, B.: 2000, Asynchronous search with aggregations, *Proceedings of the 17th National Conference on Artificial Intelligence*, AAAI Press/MIT Press, Austin, Texas, pp. 917-922.
- Yokoo, M.: 2001, *Distributed Constraint Satisfaction - Foundations of Cooperation in Multi-Agent Systems*, Springer Verlag, Berlin.
- Yu, B. and Skovgaard, HJ: 1998, A configuration tool to increase product competitiveness, *IEEE Intelligent Systems* 13(4): 34-41.

FRAMEWORKS FOR DESIGN

Constructing design worlds

Mihaly Lenart and Ana Pasztor

The situated function–behaviour–structure framework

John S Gero and Udo Kannengiesser

Representational flexibility for design

Rudi Stouffs and Ramesh Krishnamurti

CONSTRUCTING DESIGN WORLDS

Changing paradigms

MIHALY LENART
University of Kassel
Germany

AND

ANA PASZTOR
Florida International University
USA

Abstract. Since its early days in the 1950's, design has been one of the main topics of Artificial Intelligence (AI). Starting at the turn of the twentieth century, a paradigm shift has been taking place in the sciences and humanities. This had an impact on design, as well as on AI, albeit an indirect and implicit one. In design, a more human oriented postmodern style started gaining territory, and the focus of the design process shifted from technology more toward the user. From the 1980's on, user-centered and participatory design has become widely accepted and utilized. Presently, a new AI is emerging that is interested in anthropocentric models. These models are based on new principles, such as situatedness, embodiment, emotions, and social interactions. However, it seems that the paradigm shift has neither been recognized nor acknowledged in design nor in AI. Moreover, there is a lack of explicit and systematic philosophical foundations to provide a general program and directions in design. It is even more so in design-related AI for classical and neo-classical AI still dominate this field. The purpose of this paper is to outline such foundations. First we give a historical background of paradigm shifts in both science and design. We then contrast the positivist and the constructivist paradigms and discuss what they mean for design and AI.

1. Design Philosophy

The driving force of any new development in any domain is the prevailing worldview or philosophy in that domain. "Science, research, and technology

are all by-products of our philosophical reflections" (Gaarder 1996:463). Since design is an integral part of science, research, and technology, it is also a by-product of our "philosophical reflections." Any design object is the result of a creative process driven by the designer's views and convictions. It embodies the designer's philosophy. A classic example for expressing philosophical views through design is Jeremy Bentham's Panopticon. Bentham, the father of utilitarianism, published his plans for this semi-circle shaped penitentiary in 1791. Although never built, the building embodies the utilitarian concept of calculated and measurable efficiency and control. The Panopticon also epitomized and symbolized the views of the eighteenth-century French philosophes seeking radical social and cultural changes. Foucault (1979:170) connects the Panopticon with the social and cultural values of modernism and Lyon (1994:57-80) with the perils of today's information society.

2. The Paradigm Shift

Thomas Kuhn (1962) introduced the notion of paradigm shift. He defined paradigm as "a constellation of achievements—concepts, values, techniques, etc.—shared by a scientific community and used by that community to define legitimate problems and solutions." The term "paradigm" refers to a systematic set of assumptions or beliefs about fundamental aspects of reality. "Paradigms represent what we *think* about the world (but cannot prove). Our actions in the world, including the actions we take as inquirers, cannot occur without reference to those paradigms: 'As we think, so do we act.'" (Lincoln and Guba 1985:15).

Kuhn also described how paradigms drive scientific inquiries and developments and how paradigm shifts occur discontinuously and cause radical changes or revolutions throughout the history of science. Paradigm shifts replace one scientific paradigm by another incommensurable one. They move across scientific disciplines and eventually impact the entire society.

Although science and design play different roles in human development, they are intimately connected (Schön 1984:168-203), so that paradigm shifts equally affect both. While scientific paradigms have become a focus of interest in the philosophy of science, paradigms have received little attention in design. In this paper we would like to shed light on a recent paradigm shift that is rapidly capturing the scientific world and influences design as well.

Modern science and technology are based on 19th century positivist paradigms, but these paradigms have also set the trends and determined the predominant views and styles in design. New discoveries in particle physics and the exponential increase of computational power allowing complex computer simulations (from the 1980's on) lead to contradictions and raised

questions about the adequacy of these paradigms. The quest for new foundations gave rise to postmodern, in particular to constructivist paradigms. While the positivist philosophy assumes that there exists an objective reality independent from the observer, according to the constructivist view, we are constrained by our perception so that we cannot access “reality”—whether it exists or not—and so our “reality” can only be constructed. The observer and the observed cannot be separated and “reality” is co-constructed by people in a social context through a dynamic process.

While the paradigm shift from modernism or positivism to postmodernism in general and to constructivism in particular is taking place across many disciplines, it has remained largely unnoticed as such and has initiated changes in design only sporadically. The paradigm shift is recognizable by the emergence of new concepts, such as participatory and user centered design (Norman and Draper 1986; “Participatory”, 1998; Gill 1991), but there is a lack of foundation and orchestrated effort to make this shift explicit. In particular, we are not aware of any attempt to study the impact of new paradigms on design by establishing a connection between science and design. Even studies about the impact of postmodern philosophies on design (such as Coyne 1995), are rare with the notable exception of the educational field (Black and McClintock 1995; Wilson 1997) and systems engineering (Crowe et al. 1996). As a consequence of the lack of foundations there is also a lack of explicit goals and directions in design. Such goals and directions seem particularly relevant for computer-aided design and design automation. Based on a comparative study in cognitive science (Hale-Haniff and Pasztor 1999), in this paper we contrast the positivist and the constructivist view and analyze how the latter impacts design in general and design automation in particular. We argue for making the paradigm shift explicit and consequent across the whole design spectrum. We will focus on new anthropocentric AI paradigms that—we think—need to be applied and consequently extended to design computation. Although the paradigm shift has been taking place in AI, an *explicit* study of constructivism in AI is still missing. Therefore this paper is not only a contribution to AI related design in particular, but also to AI in general.

3. Constructivist Paradigm

Schwartz and Ogilvy (1979) provided an analysis of the concepts that were, at the time, emerging in a variety of disciplines and research areas. From their analysis, they synthesized seven major characteristics of an alternative naturalistic or constructivist paradigm, that stand in contrast to those of the still-dominant positivist paradigm (Lincoln and Guba 1985): Movement 1. from simple to complex realities, 2. from hierarchic to heterarchic concepts of order, 3. from mechanical to holographic image, 4. from determinacy to

indeterminacy, 5. from linear toward mutual causality, 6. from assembly to morphogenesis, and 7. from objective to perspectival views.

We note that a sharp distinction between the positivist and the constructivist view is often impossible as similar to constructivism positivist philosophy is also inhomogeneous. Although the danger that a *black-and-white* or *either-or* characterization might provide an inaccurate or even misleading picture, such a simplification helps to identify constructivist ideas and changes implied by them through the contrast of constructivist with traditional views.

In spite of the differences, we want to place particular emphasis on the fact that we do not see these two paradigms as antagonistic, but rather complementary. In any context where subjective experience is not the main issue, positivist or quantitative research methods can be applied very successfully. Such contexts involve what Paul Waczlawick calls “first order realities.” This is “the universe of ‘facts’ which can be established objectively in as much as the repetition of the same experiment yields the same result independently of by whom, and where the experiment is being carried out” (Waczlawick 1984:236). However, positivist research methods fail in contexts that involve “second order realities” or “[t]he aspect of reality in the framework of which meaning, significance, and value are attributed [...]. While it is sensible in the area of first order reality to examine, in the cases of differences of opinion, whose opinions do justice to the concrete facts and who is wrong, in the sphere of second order reality *it is senseless to argue about scientifically established ‘truth’ or to claim to have found it* [italics ours]” (ibid:237–238).

4. Emerging Design Paradigms

Design intentions are intimately related to social and cultural *values* that are the core of design paradigms. Design paradigms are more obscure than scientific ones, for the philosophy behind intentions is seldom explicit or well documented. With the advent of the Industrial Revolution and the emergence of a distinct design discipline, it became necessary to think about the *mission* of design, its nature and place in human development.

The development of new technologies and the industrialized production conquering the world lead to new paradigms that probably the Chicago architect, Louis Sullivan formulated first. His famous dictum, “form follows function” became a maxim for architects and designers of the modernist movement and a leitmotif of the Bauhaus. The Bauhaus of the 1920s and the New Bauhaus (later, Institute of Design) of the 1940s embraced the “*Zeitgeist*” by focusing on technology and industrial production. After WWII, while Moholy-Nagy was working on the legacy of the Bauhaus in Chicago, another alumnus of the Bauhaus, Max Bill, founded the Hochschule für

Gestaltung (HfG) in Ulm, Germany. The HfG under Max Bill's successor, Tomás Moldonado, ventured beyond the limitation of the Bauhaus by seeking scientific foundations and developing systematics for a new *science of design*. Although this approach was based on the same positivist paradigms as the Bauhaus, it also embraced the emerging systems theory and cybernetics.

In the second half of the twentieth century, designers turned to traditional forms and styles, especially to Egyptian, Greek, and Roman, as a reaction to the sparse and expedient designs of the previous era. The bizarre mixture of new and old styles became known as postmodern design. Experiencing with a hodge-podge of techniques and styles was, however, more than just a swing of fashion. Designers found that the functionalist or modernist approach concentrates exclusively on economic and fails to acknowledge human values. Without having any relation to postmodern design, system designers came to the same conclusion. In particular, they recognized that in the fastest growing field of information technology, computer systems have been developed mainly under economic considerations. Originating in the Scandinavian countries, from the 1980's on, participatory and user-centered design became an integral part of system design (Schuler and Namioka 1993; Bailey 1996; Noyes and Baber 1999). Also, the design of user interfaces has changed significantly: Participatory methods and human factors such as ergonomics have come to the forefront. Human factors play a central role in other areas of system design as well. Donald Norman (1993:253) writes: "Remember the motto of the 1933 Chicago World's Fair [...]: 'Science Finds, Industry Applies, Man Conforms'? That was the machine-centered view of the world—unashamedly, proudly machine-centered. It is time to revolt. We can't conform. Moreover, we shouldn't have to. It is science and technology—and thereby, industry—that should do the conforming. The slogan of the 1930s has been with us long enough. Now, as we enter the twenty-first century, it is time for a person-centered motto, one that puts the emphasis right: People Propose, Science Studies, Technology Conforms."

5. Emerging Scientific Paradigms

As in design, in the first half of the twentieth century interests and views in science and philosophy shifted as well. The triggering event was probably Planck's discovery that the second law of thermodynamics was a statistical rather than an absolute law. The implications of Planck's quantum theory (published in 1900) that we can have only statistical knowledge of certain phenomena and—according to the Copenhagen interpretation—the observer cannot be separated from the observed phenomena changed the dominant Cartesian world-view. Science ceased to be the ultimate objectivity to ex-

plore nature, or as Heisenberg formulated: “Natural science does not simply describe and explain nature; it is part of the interplay between nature and ourselves; it describes nature as exposed to our method of questioning.”

The other major impact on the emerging paradigm was Einstein’s special and general theories of relativity. A significant aspect of Einstein’s general (gravitational) theory is its nonlinearity, which implies that the whole is greater than the sum of its parts. Both, the quantum and relativity theory raised questions not just about earlier methods, but also about the focus of scientific inquiries. These earlier methods were based on Descartes’ reductionist views. According to these views, the world is an assembly of simpler physical entities and therefore one can successfully study complex phenomena by breaking them apart and analyzing the simple parts separately, mainly by mathematical modeling. This idea has been particularly successful in the so-called ‘hard sciences’ leading to useful theories and valuable applications, in particular a series of technological break-throughs. The most important one was the development of computers. Ironically, however, computers also helped to undermine reductionism by playing a central role in the emergence of cybernetics and systems theory. Cybernetics and systems theory studied complex systems and soon recognized that any reductionist approach was doomed to fail. “The great shock of twentieth-century science has been” writes Fritjof Capra (1996:29) “that systems cannot be understood by analysis. The properties of the parts are not intrinsic properties but can be understood only within the context of the larger whole.” This came as a surprise to the entire scientific community because suddenly scientific interest turned to complex phenomena and found that systems are everywhere, not just in social and life sciences, but in hard sciences as well. In fact, it was discoveries in physics and chemistry, such as the Bénard cells or the laser that raised worldwide interest in complex systems. By studying complex systems, researchers recognized that similar nonlinear mathematical models could describe systems as diverse as the weather and the stock market for their behavioral patterns are similar at some abstract level.

At the same time researchers also recognized that understanding the role of *humans* in scientific inquiries, or the cognitive aspects of scientific endeavors is fundamental to understanding complex systems. The result was a paradigm shift across disciplines that also triggered a large number of inter- and cross-disciplinary researches. “No compelling reason has ever been offered why the same [holistic] strategy should not work to unite the natural sciences with the social sciences and humanities” (Wilson 1998:267). Neither seems to be a compelling reason for treating design differently from natural sciences or forgo applying the same methods and strategies that promise “to unite the natural sciences with the social sciences and humanities.”

6. Contrasting Positivist and Constructivist Paradigms

Lincoln and Guba (1985:37-38) list five basic presuppositions (axioms) of both the positivist and naturalistic paradigms. They also note that these five basic characteristics of the alternative emerging paradigm, “have a synergistic relationship to one another—none could stand alone” (*ibid*:56).

Axiom 1: The nature of reality (ontology). *Positivist paradigm*: There is a single tangible reality “out there” fragmentable into independent variables and processes, any of which may be studied independently of the others; inquiry can converge onto that reality until, finally, it can be predicted and controlled. *Naturalistic paradigm*: There are multiple constructed realities that may be studied only holistically; inquiry into these multiple realities will inevitably diverge (each inquiry raises more questions than it answers) so that prediction and control are unlikely outcomes, although some level of understanding may be achieved.

Axiom 2: The relationship of knower to known (epistemology). *Positivist*: The inquirer and the object of inquiry are independent; the knower and the known constitute a discrete dualism. *Naturalistic*: The inquirer and the “object” of inquiry interact to influence one another; knower and known are inseparable.

Axiom 3: The possibility of generalization. *Positivist*: The aim of inquiry is to develop a nomothetic body of knowledge in the form of generalizations that are truth statements both time- and context-free; they must be true anywhere at any time. *Naturalistic*: The aim of inquiry is to develop an idiographic body of knowledge in the form of “working hypotheses” that describe the individual case. Inductive learning leads to grounded theories that are context-dependent generalizations.

Axiom 4: The possibility of causal linkages. *Positivist*: Every action can be explained as the result (effect) of a real cause that precedes the effect temporally. *Naturalistic*: All entities are in a state of mutual, simultaneous shaping so that it is impossible to distinguish causes from effects.

Axiom 5: The role of values in inquiry. *Positivist*: Inquiry is value-free and can be guaranteed to be so by virtue of the objective methodology employed. *Naturalistic*: Inquiry is value-bound in at least in the following five ways:

Corollary 1: Inquiries are influenced by *inquirer* values as expressed in the choice of a problem, evaluand, or policy option, and in the framing, bounding, and focusing of that problem, evaluand, or policy option.

Corollary 2: Inquiry is influenced by the choice of the *paradigm* that guides the investigation into the problem.

Corollary 3: Inquiry is influenced by the choice of the *substantive theory* utilized to guide the collection and analysis of data and in the interpretation of findings.

Corollary 4: Inquiry is influenced by the values that inhere in the *context*.

Corollary 5: With respect to corollaries 1 through 4 above, inquiry is either *value-resonant* (reinforcing or congruent) or *value-dissonant* (conflicting). Problem, evaluand, or policy option, paradigm, theory, and context must exhibit *congruence* (value-resonance) if the inquiry is to produce meaningful results.

Objections can be raised for these axioms fail to reflect the diversity of the many views; instead they lump these views together as either positivist or constructivist. Nevertheless, these axioms provide a useful framework for characterizing the paradigm change that is taking place in design and AI. Is there a need for such a characterization? We think so. Currently designers and AI practitioners live with contradicting paradigms either without being aware of or ignoring them as well as their consequences. A recent issue of the AI Magazine (Vol. 22, No.3, Fall 2001) dedicated to the topic of creativity illustrates this point quite convincingly. On its last pages reports about the AAAI 2000 Fall Symposiums describe increased interest in situated and embodied agents. In connection with socially intelligent agents, AI-researchers talk about the importance and relevance of human intentions, emotions and beliefs. This seems to be at odds with most of the creative AI-systems described on pages 13-28 by one of the AI-pioneers, Bruce Buchanan. Creativity is at the heart of design and we all know from experience that it is a social concept that requires a stimulating environment and bodily experience. How can anyone be creative without having clear intentions, emotions and beliefs? And yet so far mainstream AI seems to overlook or dismiss entirely this aspect.

7. AI in Perspective

After WWII the controversy between the positivist and constructivist views split the cybernetics movement and lead to the emergence of AI. Also the fathers of the cybernetics movement, Norbert Wiener and John von Neumann parted ways. Wiener with Claude Shanon developed information theory “to understanding the mind as a systems phenomenon and became the first successful attempt in science to understand the Cartesian division between mind and body” (Capra 1996:55). Von Neumann, on the other hand, developed the foundations of today’s computer technology and used it as the conceptual basis for the scientific study of the mind. His “invention of the computer and his analogy between computer and brain functioning are so closely intertwined that it is difficult to know which came first” (Capra 1996:66). For the rest of the twentieth century this analogy became the prevalent view—called cognitivism—in cognitive science and the foundation of the emerging field of AI.

Since the mid 1950s the defining paradigm of AI has been the Cartesian division between hard- and software and the belief that the human mind is a problem solving machine operating on symbols. Although this paradigm resulted in a number of successful applications and provided valuable insight into the functioning of the mind, it is currently under attack from all sides as it runs against new developments and trends in science and philosophy (c.f. Dreyfus and Haugeland in Baumgartner and Payr 1995:72 and 106, resp.).

Problem-solving as symbol processing has been particularly relevant for design related AI providing successful applications, in particularly expert systems. However, they were successful only in solving specific problems within a limited domain but not in understanding and modeling the designers' mind. Thus the success was more in the realm of economics than of cognitive modeling. According to John Searle, the inherent problem with symbolic computation is that “[o]nly an observer, user, or agent can assign a symbolic interpretation to something, which becomes a symbol only by the act of interpreting it as symbol. So computation exists only relative to some agent or observer who imposes a computational interpretation on some phenomenon. [...] But if the question is whether consciousness is intrinsically computation, then the answer is that nothing is intrinsically computational; it is observer-relative” (Baumgartner and Payr 1995:210). Searle concludes: “without the computer, there would have been no discipline of cognitive science as we know it. However, one of the most important results of the development of cognitive science has been the refutation of the computational model” (*ibid*:205). Since design requires complex skills, cognitivism or the symbol processing AI paradigm is inadequate for capturing any of the relevant cognitive aspects of design.

The AI community largely ignored, rejected and dismissed such external critics that “often viewed AI models through philosophical lenses and found scandalously bad. AI people [...] often do not recognize their methods in the interpretations of the critics, and as a result they have sometimes regarded [them] as practically insane” (Agre 1997:xi). This is currently changing as a growing number of AI-practitioners share Agre’s view “that the substantive analysis of human experience in the main traditions of AI research is profoundly mistaken” and “that people are intimately involved in the world around them and that the epistemological isolation that Descartes took for granted is untenable” (*ibid*).

It seems that after the split in cybernetics, the maturing AI came back full circle, as many of the original ideas of cybernetics are being now rediscovered and pursued vigorously in AI. This and other developments indicate that cognitivism is on the decline and neural based cognitive models are on the rise in AI. We believe that cognitive aspects of design and their relation to the positivist and constructivist views—as it will be discussed below—is a fundamental part of this development.

8. Contrasting Intelligent Design Systems Guided by Positivist and Constructivist Approaches

In this section we will use Lincoln and Guba’s (1985) above described five axioms as an interactive framework to contrast how the positivist and constructivist philosophies influence design.

8.1 THE NATURE OF REALITY

Objectivity

According to the positivist view, there is only one single, true, discoverable, knowable, and fragmentable reality. However, “[t]he concept of reality seems to demand [...] the idea of an absolute conception of the world” (Trigg 1993:103) or a “God’s-Eye View.” This is what Descartes envisioned as a determinate picture of the world, “independent of any knowledge or representation in thought” (Williams 1978:65). The designer shapes reality to satisfy the user. Experts (e.g. the designer) have access to reality. Access also means that reality can be described by adequate means, such as symbols. The language of mathematics provides the most abstract symbolic representation of reality. “The central idea of a mathematical model is somehow to mirror the observable quantities of the real world in the abstract structures making up the world of mathematics” (Casti 1990:31). Quantitative data reinforce trust in objectivity: What can be measured is independent of our interpretation.

However, we have no way of knowing whether there exists reality independent of our interpretation, much less of telling whether and how a given view relates to it. By referring to Maturana (1980), Capra writes: “[T]he nervous system is not only self-organizing but also continually self-referencing, so that perception cannot be viewed as the representation of an external reality but must be understood as the continual creation of new relationships within the neural network” (Capra 1996:96). A prime example is color vision (c.f. Wilson 1998:159-162, Varela et al. 1999:181-183). “Reality” deceives and in design it is particularly easy to find examples for claims to describe realities that crumble under scrutiny. Just think of economic, traffic, population, etc. data that are claimed to be objective but are, in fact, just quantified expressions of subjective views. As the Scandinavian computer scientist, Arne Kjellman (2000) put it: “Reality has the same status to technology and the materialistic sciences as God has to the true believer—namely the utmost answer. Unfortunately this is too an easy and even misleading way to walk.”

The constructivist view is that there are multiple versions of holistic co-constructed, invented realities. Anything we experience is through our perception. “Reality” is co-created by communicating our experiences. Since any process of co-constructing “reality” is unique, it is impossible to exactly reproduce this process. Each (re-)constructed “reality” is different. Also, “reality” created in a complex environment through complex communication is holistic in nature, and reducing or filtering aspects of the construction would mean interfering with the process and thus distorting “reality.”

It is now widely recognized that the success of a product depends to a large degree on whether the user’s needs, thoughts, and behaviors have been

studied and incorporated into the design process. This has given rise first to user centered and then to participatory design. “Participatory Design (PD) is a set of diverse ways of thinking, planning, and acting through which people make their work, technologies, and social institutions more responsive to human needs. [...] PD enables users, stakeholders, and other interested parties to play powerful roles in shaping technological and work outcomes to reflect their interests” (“Participatory” 1998a). System analysis of user behavior revealed significant discrepancies between the participants’ “realities.” In his book ‘Aramis,’ Bruno Latour (1996) describes a case study of design “realities.” Aramis was a revolutionary but failed mega-project (1970-1988) to solve urban public transportation problems of France. Latour, who co-developed Actor-Network Theory, lets the “actors” (including Aramis itself) describe their “realities,” their role in this project and how each of them invariably sees different reasons for the same failures.

Design is an iterative process to elicit and construct relevant conditions to be satisfied by the design object. These conditions or requirements and the knowledge of how to satisfy them are the foundations of the process. Requirement or knowledge elicitation traditionally is a one-directional linear process where the elicitor (designer) extracts information (facts and knowledge) from the domain expert. The process is driven by the elicitor’s model of the world that is assumed to reflect “reality” (Jirotka and Gougen, 1994). In contrast, co-creation of realities is a dynamic, multidirectional process. It is also non-authoritarian and democratic that seeks agreements. Constructivist models need to be dynamic to incorporate evolutionary changes, and they need to be holistic to capture the complexity of the environment

Notion of ‘Self’

The positivist view is that there is one knowable and true self to be discovered and understood. Any agent, be it the designer or the user, has one single self. This self is the identity of the agent that can be elicited and described in objective terms. Modeling the agent’s behavior is based on the understanding of the agent’s self. This self is time and context independent. The Cartesian dualist view is that the experience of the self is radically different from any other sensory or mental experience. Such a transcendental nature of the self is inconsistent with positivist philosophies so that the self is usually left out intentionally from cognitive models. However, “without the language of the self—of our internal states, processes, and characteristics—social life would be virtually unrecognizable” (Gergen 1991:6). Although most design systems don’t have a notion of self or don’t address problems related to the self of an agent explicitly, it can usually be inferred from the behavior, beliefs, and values of the agent. The self is usually an abstract disembodied concept subordinated to the purpose of the system.

There is something mystical about the nature of the self as it is elusive unlike any other ordinary human experience: “We envision each person as having a mind that is centered upon something we call a Self—but we don’t agree on what Selves might be, and we’re even less clear about what Selves do” (Minsky 2002). According to the constructivist view, multiple versions of “self” are possible. Similar to “reality,” the “self” is an invented construct. Brain research provides supporting evidence: “The most obvious fact about existence is your sense of being a single, unified self ‘in charge’ of your destiny; so obvious, in fact, that you rarely pause to think about it. And yet [experiments …] suggest that there is in fact another being inside you that goes about his or her business without your knowledge or awareness. And, as it turns out, there is not just one such zombie but a multitude of them inhabiting your brain. If so, your concept of a single ‘I’ or ‘self’ inhabiting your brain may be simply an illusion” (Ramachandran and Blakeslee 1998:83-84). How the “self” is constructed depends not only on the personality of the agent, but also on its context. It is important in the context of design to make the “self” explicit and to understand its ontology.

With changing environment and context the self and its manifestations also change. Accordingly, the self of an agent must co-evolve with its social, cultural, and physical environment. As for the concept of a disembodied self, AI has long ignored “the most obvious [...] aspect of human intelligence [...] that it is embodied” (Brooks et al. 1998). To trust them, intelligent agents need to be believable. To be believable, they need to have human characteristics, in particular personality. Interface agents, such as Erin the bartender (Isbister and Hayes-Roth 1998) or Melvin and Sluggo, the kids from the playground (Reilly 1996), all have their own personalities. Since personality is a function of the “self,” agents need to have a “self” in the first place.

Emotions

According to the positivist view, human experience is fragmented, with rational aspects being emphasized. Emotions are treated as separate from the rest of the human experience. Cognitive linguist, Mark Johnson (1987:xxiv) lists four main characteristics of the positivist view of rationality: “1. Reasoning is a rule-governed manipulation of connections among symbols. [...] 2. The core of rationality is formal logic. [...] 3. [...] Rationality is essentially disembodied; it consists of pure abstract logical relations and operations independent of subjective processes in the reasoner’s mind. [...] 4. [...] It is assumed that human beings can somehow ‘plug into’ a transcendent, autonomous reality that stands beyond all historical developments.” Western philosophy has taken a route in which pure reason is believed to be able to transcend the constraints of the human body that is trapped in the confines of perception and emotion. In traditional Western philosophy

mathematics has been seen as the purest example of reason: “purely abstract, transcendental, culture-free, unemotional, universal, decontextualized, disembodied, and hence formal” (Lakoff and Nuñez 1997:22). However, new gaining recognition challenges this view: “Mathematics is part of human culture and history, which is rooted in our biological nature and our physical and biological surroundings. Our mathematical ideas in general match our world for the same reason that our lungs match earth’s atmosphere” (Hersch 1997:17).

According to the positivist view, design decisions should be driven by rationality. To be rational, design arguments often make use of mathematics, in particular of formal logic. Designers often fail to recognize that rationality might be an important tool for argumentation, but it is emotions not rationality that is central to decision making. According to Antonio Damasio, a leading neuroscientist, “feelings have a say on how the rest of the brain and cognition go about their business” (Damasio 1994:159–160). His patients with frontal lobe disorders provide ample evidence that the lack of emotions have disastrous consequences for the subject’s decision-making ability. For the rationalist, the most important aspect of design is functionality. The widely held assumption is that there must be a logical reason for all design functions and for all solutions satisfying the functions. However, this rational has changed. According to Susan Yelavich (Cooper-Hewitt National Museum), “[f]unction now embraces psychology and emotion” (Gibney and Luscombe 2000).

Creativity is fundamental to design and it also plays a central role in design related AI (Gero 1992; Brown 1993; Goel 1997). There is plenty of psychological data showing that emotions have an impact on creativity. Famous artists, designers, and scientist have reported the effect of mood change and chemical substances that impact the nervous system on their own creativity. The same holds for memory: “Memory retrieval is largely a mystery, but there is increasing evidence that emotions play a role in its function. Memory may be the chief mechanism through which emotions enter into the mental associations active in analogical thinking and creativity” (Picard 1997:41).

Regarding emotions, AI is not much different from design: “Traditional Artificial Intelligence has tried to tackle the problem of building artificially intelligent systems from the top down. It tackled intelligence through the notion of thought and reason” (Brooks, 1991:1). Recognizing the relevance of emotions to our cognitive functions in general and to AI in particular lead to the development of the new AI-field of *affective computing*. “Affective computing includes implementing emotions, and therefore can aid the development and testing of [...] emotional theories” (Picard 1997:3).

While lifelike interface agents with emotions have left the research labs and now enter the commercial arena (Ball et al. 1997; Ball and Breese 1998,

“Extempo”, 1999; “Firefly” 1999), design agents still lack emotions. Trust and user acceptance (Maes 1994; Mitchel et al. 1994) require believability of the agents. “Emotion is one of the primary means to achieve this believability, this illusion of life, because it helps us know that characters really care about what happens in the world, that they truly have desires” (Bates 1994). Emotions turned out to be also relevant for robotics (Kaplan 2000) and captured the interest of prominent AI researchers (Minsky 2002).

Constructivists believe that human experience is holistic and holographic. Emotional experience is ongoing and simultaneous with the rest of experience. Emotions are continuous support factors affecting all human experience. The great importance of the emotional component of mental states, especially of the physical aspects of this component, is probably best explained by Damasio’s (1994:170) “somatic-marker hypothesis.” Since decisions are influenced by emotions, believable and trustworthy design agents need emotions.

Concept of ‘Truth’

In the positivist world reality is divided into right or true and wrong or false versions. The quest for the truth is at the heart of all sciences. However, as von Glaserfeld (1976:4) points out, while trying to access reality, we have been caught in an age long dilemma: On one hand truth is (traditionally) defined as “the perfect match, the flawless representation” of reality, but on the other hand, we *all* live in a world of genetic, social and cultural constraints, some of which we cannot ever “escape.” *Who then, is to judge the match?*

To answer this question, Western positivist philosophy has taken a route in which, given the right tools, pure reason is believed to be able to transcend all constraints and the confines of the human body, including those of perception and emotion. The traditional scientist is out to find objective truth. In doing so, he or she is trained to be value-neutral in order to be able to objectively judge “the perfect match” with reality. In practice, however, there is a direct “relationship between claims to truth and the distribution of power in society” (Gergen 1991:95). Mathematics as an objective, value-free way to study nature plays a central role in discovering the truth. However, as we discussed earlier, mathematics tells us more about ourselves, the functioning of our mind than the physical world.

According to the positivist view, design solutions are also right or wrong and design models true or false. There are objective measures (statistics, scales, benchmarks, norms, etc.) to test the adequacy of a solution. Based on these it can be decided whether something is right or wrong, true or false. How misleading this position is, illustrates the recent controversy about the superiority of Microsoft’s NT and Linux. The battle is apparently about benchmarks and other measurable differences in performance as a proof of

which one is better. Both parties use the *same* data to prove their point but they draw *different* conclusions and believe in *different* ‘truths.’

Constructivists counter the quest for truth by claiming that many interpretations are possible. Thus, no one is any more right or wrong than any others. Nobody is holding the truth. Neither the designer’s nor the user’s or anybody else’s model of the world is correct. Neither the designer nor the user is right or wrong, has the right or the wrong answer or solution. Instead, the question is whether the answer or solution proposed by either party is *useful*. However, usefulness is not an absolute term; it is relative to the agent, purpose, and context. In other words, for evaluating a design solution, one needs to know not only whether it is *useful*, but also who is going to benefit from it, what its purpose is, and under which circumstances it will be used.

8.2 THE RELATIONSHIP BETWEEN THE KNOWER AND THE KNOWN

Authority

The positivist position is that any viable system operates in a hierarchical and authoritarian fashion. The designer and user are separate: The designer is the “objective” expert who makes decisions in the user’s interest. Expert and knowledge based systems exemplify this attitude. These systems are built on and driven by the expert’s knowledge. The expert has the authority and the power to make decisions. The user is asked for information but rarely for opinion. The user doesn’t contribute to the solution.

The main problem with this position is not as much the potential misuse of authority as (perhaps inadvertently) ignoring inherent social issues and the lack of systemic considerations. Unilateral decision-making is often the cause of faulty designs (Norman 1988, 1993) and ignoring human factors might have grave consequences (Pool 1997).

Authority is two-directional: Decisions and opinions are handed down in an authoritarian manner, but they are also need to be accepted as unquestionable authority at the receiving end. Sometimes the latter occurs in the absence of any intentional authority. An example is computer programs that possess authority not by assignment but by perception or acceptance. This raises serious ethical questions (Weizenbaum 1976). The widespread use of expert systems also raises concerns: “If [expert systems] are not fully understood by people who buy them and use them and sell them, then you can distort the nature of what is being done” (Winograd in Baumgartner and Payr 1995:296).

According to constructivists any well-functioning system is democratic and collaborative. The designer and user both have areas of their own expertise. Design systems are interactive; they facilitate communication between the participants (agents). The user and the designer are inseparable. They are

co-creators of experience, mutually influencing each other. The design task is defined jointly and solutions are developed in collaboration. Evaluation of solutions is also a joint process.

Goal setting

In the positivist world the designer analyses the problem, sets goals unilaterally, and decides when to stop the design process by declaring the solution as satisfactory. “[T]he industrial designer seemed to ‘know what the public wants’” characterizes Petroski (1994:169) the emerging field of industrial design after WWII. Many expert and knowledge based design systems share this view. Their architecture is tailored to the problem solving method of the designer or the expert and little or no effort is being made to keep the system flexible and open. The professional bias of the designer influences the goal setting and the process that leads to the goals. Other participants of the design process have little or no impact on goal setting.

In the constructivist world the designer and user collaborate to formulate problems in solvable form and together they determine when to stop the design process. Once a problem is formulated to the extent that both the user and the designer agree on most of the details, finding solutions is relatively simple and rather technical. To achieve such collaboration, we need new efficient methods to facilitate agent communication.

Setting goals is a dynamic process. As the designer and the user collaborate, they co-evolve set goals and co-create new ones. Goals start out to be unspecific and become more specific as the design process unfolds. They can change even after completion. Flexibility and adaptability is fundamental in each phase of the development. Adaptable and evolvable hardware described in (Sipper and Ronald 2000) are examples for a recent development of anticipating changes and actually changing the product as it is being developed. Design becomes an adaptation of the evolving device to new goals and requirements as opposed to the tradition of developing first simulations and tests before a device is being built. This adaptive design method uses evolutionary computation to deal with unpredictable changes (Koza et al. 1999).

8.3 THE POSSIBILITY OF GENERALIZATION

According to the positivist view absolute time and context free generalizations such as diagnoses and prognoses are possible. Keeping certain parameters constant guarantees the same outcomes of repeated experiments or experiences. This obviously doesn't hold for design tasks that have almost no time- and/or context-independent parameters. Thus, repeating the exact same task is impossible. “We conclude that action is indeed grounded in the situation where it occurs” (Anderson et al. 1997) therefore context-free generalizations and predictions are also impossible. We can find only tendencies.

cies or general patterns and make statistical predictions about the outcome of a process.

Positivists counter that even if the tasks change with time and context, *methods* to analyze a problem and find solutions are time and context independent. The same holds for intelligent design systems. AI tools and methods are generalized for reuse and domain extension. For example SACON (Bennett and Engelmore 1979), an advisory expert system for structural engineers has been expanded and modified to develop other advisory systems in other domains. However, “[t]here is clearly no guarantee of success in designing new things on the basis of past success alone, and this is why artificial intelligence, expert systems, and other computer-based design aids whose logic follows examples of success can have limited application” (Petroski 1994:31).

8.4 THE POSSIBILITY OF CAUSE-EFFECT LINKAGES

Order and structure

According to the positivist view there are real causes that temporally and sequentially precede their effects. Explanation based systems assume that we operate on a reality in which everything is in a linear cause-effect relation. Cause-effect relations are the foundations of human reasoning. Reasoning is considered to be “purely abstract, transcendental, culture-free, unemotional, universal, decontextualized, disembodied, and hence formal” (Lakoff and Núñez 1997: 22).

However, cause-effect or common sense reasoning breaks down completely in complex systems. They are unpredictable and incalculable so that one never knows what is the predecessor or cause and what is the consequence or effect. Also, a cause can be the effect of another cause and the chain might lead to circularity or feedback loops, a cybernetic concept introduced by Norbert Wiener. “A feedback loop” explains Capra (1996:56) “is a circular arrangement of causally connected elements, in which an initial cause propagates around the links of the loop, so that each element has an effect on the next, until the last ‘feeds back’ the effect into the first element of the cycle.” The feedback is the foundation of self-organization and positive feedback loops—where each element in the loop reinforces the initial effect—the one of emergence.

The circular cause-effect relationship is also at the heart of nonlinearity. Most design systems are complex nonlinear dynamic systems with feedback loops. Classic examples are the thermostat (e.g. in air-conditioning systems or refrigerators) and the centrifugal governor of the steam engine (regulating the steam flow). Feedback loops are usually part of larger systems, or loops, such as the thermostat is part of the climate system of the entire building or

even the surrounding environment. In fact, any design system is part of a larger feedback loop involving science and the environment.

The constructivist view is that causality is a context dependent tool rather than inherent property of the world. “So we should not conclude that causal assertion cannot be used at all, but rather we should be aware of their limitations. Causal assertions are normal and useful for talking about entities that have intentions. However, they are often misleading in talk about non-intentional entities.” (Gougen 2001). Such entities are in a state of mutual simultaneous shaping, so it is impossible to distinguish causes from effects. This means that formal reasoning that is used in production or logic-based systems cannot adequately describe human reasoning. Instead of using causality or if-then rules, dynamic neural pattern matching is a more adequate and useful model of reasoning.

Solutions

According to the positivist view inquiries are directed to and focused on the cause of problems. Design focuses on issues that caused or maintain the problem. Finding solutions means to identify the cause of the problem and change it. The solution process is limited in scope because it concentrates only on issues that are directly related to the problem.

The fallacy of this view is that solutions for complex design problems are often outside of the original scope of the problem. In the case of circular causalities, one cannot solve the problem by finding a solution to some or even to all of the causes. To address all issues related to circular causes, we need to leave the problem domain. For example, it is probably better to seek the solution to the pollution problem of car emission in improving (public) transportation rather than in car design, in spite of the fact that the cause of the pollution is the car’s fuel and engine. This is also related to a phenomenon that design theorist, Horst Rittel, called “Sachzwang” or “compelling facts.” Design decisions create parameters that become compelling circumstances for new design tasks. “Compelling facts” are an easy way out to reduce the complexity of design tasks. It is much harder to question the decisions that lead to these “facts” and work through alternatives by reversing (some of) them.

According to the constructivist view design focuses on what might be getting in the way of solving the problem or attaining a desired goal. To find solutions, we need to expand the scope of analysis. Expansion is driven by holistic views exploring various meta-levels.

Orientation

In the positivist world design goals are hierarchically organized. Setting goals and priorities is based on the past with the attempt to change the future.

This view may be restricting and hampering efforts to find solutions for goals and priorities depend on unpredictable complex socioeconomic factors. Past experience is valuable, but cannot always help to cope with new developments. Rigid organization prevents fast adaptation.

According to the constructivist view design goals need to be organized as a network. Goals and priorities are set in the present and future dynamically as the design process evolves.

Interaction

Positivists view human and nature as separate entities and differentiate between social interactions and human interactions with nature. They place special emphasis on design problems that are created by human interaction with nature. They view design as a tool to change the environment by utilizing natural resources often divorced from social issues.

This dualist view is the cause of many environmental problems. Nature has been long exploited for humans' sake. There are plenty of examples from deforestation for new developments to dam and canal building for irrigation and transportation that lead to the destruction of the environment. Environmental consciousness is still not widely accepted integral part of our culture.

According to the constructivist view, society is an integral part of nature so that social developments cannot be separated from the environment. Changing the social structure impacts the environment: the flora and fauna, landscape, air, water, etc.

According to constructivism, design problems are constructed through interaction between people. People are integral part of nature and thus of the same system. Design is about changing systems, not just environments. Systems can be changed only through social interactions. They also play a central role in developing intelligent agents: "Up to now, human factors specialists only have focused on the cognitive and usability aspects of design. Tapping into the social aspects of communication provides the potential for designing more natural interaction" (Trower 1999). In fact, socially interacting agents is the topic of a new emerging AI research area (Dautenhahn 1998, 1999; Breazeal and Scassellati 1998; Dautenhahn 2001).

8.5 THE ROLE OF VALUES

Values are important arbiters of choice. We access them in an ongoing manner for many sequences of mental operations that lead to a particular goal. For example, our values are evident in how we are motivated: what we move toward and away from; what attracts and repels us. In the process of making decisions, we alternate between options and our highest criteria or values, making our selection based on which option best matches our values. We make plans and set priorities based on what is important to us. Similar val-

ues attract people while different ones repel them. There is also a relationship between values and emotions.

Objectivity

According to the positivist view design values are objective. The designer as the expert of the field has the right values and can evaluate solutions objectively. Professional codes and ethics help to enforce these values.

Most values are, however, subjective, based on culture, environment, and personal experience. The designer and the user don't necessarily share the same values. Even professional values are not always shared. For example, investment in and commitment to certain methods or technologies solidify personal beliefs and values as the controversy about alternate (AC) versus direct current (DC) between Edison and Westinghouse demonstrates. Edison's commitment to DC lead to butchering dozens of farm animals in his lab as well as developing the first electric chair and actively endorsing its use (Flatow, 1993:36-42). Moreover, technology itself is not value free: "[E]ach technology poses a mind-set, a way of thinking about it and the activities to which it is relevant, a mind-set that soon pervades those touched by it, often unwittingly, often unwillingly. The more successful and widespread the technology, the greater its impact upon the thought patterns of those who use it, and consequently, the greater its impact upon all of society" (Norman 1993:243).

The National Society of Professional Engineers (NSPE) Code of Ethics for Engineers (NSPE, 1998) contains only general statements with unspecific nominalizations, such as *safety*, *objectivity*, *truthfulness*, *honor*, *responsibility*, *integrity*, *competence*, etc. the meaning of which is anybody's. The Code also emphasizes *truth* and *facts* discussed earlier. Therefore, it is by no means a collection of "objective" values of the engineering profession.

The constructivist view is that design is always value-based. The values of both, the designer and user, are assumed to affect the outcome of the design process. This can only be achieved if the values are explicit. Neither the designer's nor the user's values have priority.

Norms and Meaning

According to the positivist view, the designer uses normative-based assessments to evaluate user's need. The designer (expert) set the norms that are influenced by the society at large. The designer educates the user about values, assigns meanings for the user and interprets users' communications according to the designer's preferred model.

This view ignores the user's interests and preferences. Communication provides a distorted perspective often unrelated to the user's communication and its meanings. An example is medical, in particular assistive devices. Interviews with professionals in the rehabilitation field reveal alarmingly low

success rate as most of these devices wind up unused in a closet. The reason in each case is invariably lack of communication, miscommunication and as a consequence, profound misunderstanding between the designer and the user.

According to the constructivist view, the designer generally does not do formal assessments. The user serves as his or her own norm or he or she is choosing the norm. These norms are elicited in the design process and serve as guidelines for developing solutions. The designer and user communicate to establish which requirements and values are *useful* for the user in the given context. Social values are treated as reference points. There may be multiple and flexible reference points. The designer helps the user to make his or her values explicit. The designer attends to what is important to the user, helps the user to clarify meanings and assign his or her own interpretations.

9. Conclusion

The purpose of this paper is to draw attention to and discuss the impact of a recent paradigm shift from modernism or positivism to constructivism in design in general and in design automation in particular. The constructivist view is emerging in both design and AI. User centered and participatory methods are being acknowledged and applied in design. The attention in AI is turning to integrated, embodied, socially interactive, emotional agents. However, as paradigms are shifting, it becomes crucial to explicitly and deliberately define each field's philosophical assumptions, so that a congruence of researcher's values, methodologies, and research goals can be achieved. In this paper we have laid the groundwork for an explicit philosophical foundation for design and design related AI.

Acknowledgment

We are grateful to Mary Hale-Haniff for numerous insights and discussions that lead to this paper. The second author's work was supported by the following grants: NSF-CISE-EIA-9812636 with the DSP Center, NSF-MII-EIA-9906600 with the CATE Center, and ONR-N000 14-99-1-0952 at Florida International University.

References

- Agre, PE: 1997, *Computation and Human Experience*, Cambridge Univ. P., Cambridge, UK.
Anderson, J, L Reder, and HA Simon: 1997, *Applications and Missapplications of Cognitive Psychology to Mathematics Education*, <http://act.psy.cmu.edu/personal/ja/misapplied.html>
Bailey, RW: 1996, *Human Performance Engineering* (3rd Ed.), Prentice Hall, Englewood Cliffs, NJ.

- Ball, G, D Ling, D Kurlander, J Miller, D Pugh, T Skelly, A Stankosky, D Thiel, M van Dantzich and T. Wax: Lifelike Computer Characters: 1997, the persona project at microsoft, in JM Bradshaw (ed.), *Research in Software Agents*, AAAI Press/The MIT Press, Cambridge, MA, pp.191-222.
- Ball, G and Breese, J: 1998, Emotion and personality in a conversational character, *Workshop on Embedded Conversational Characters*, Tahoe City, California, October 12-15, pp. 83-84, 119-121.
- Bates, J: 1994, The role of emotions in believable agents, *Communications of the ACM*, 37(7): 122-125.
- Baumgartner, P and Payr, S (eds): 1995, *Speaking Minds, Interviews with Twenty Eminent Cognitive Scientists*, Princeton University Press, NJ.
- Bennett, JS and Engelmore, RS: 1979, SACON: a knowledge-based consultant for structural analysis, *Proceedings IJCAI- 79*, pp.47-49.
- Black, JB and McClintock, RO: 1995, An interpretation construction approach to constructivist design, in BG. Wilson (ed.), *Constructivist Learning Environments: Case Studies in Instructional Design*, Educational Technology Publications, Englewood Cliffs, NJ, pp. 1-7; also: <http://www.ilt.columbia.edu/ilt/papers/ICON.html>
- Breazeal-(Ferrell), C and Scassellati, B: to appear, Infant-like social interactions between a robot and a human caretaker, *Special Issue of Adaptive Behavior on Simulation Models of Social Agents*.
- Brown, DC: 1993, Intelligent computer-aided design systems, in A Kent and J Williams (eds), *Encyclopedia of Computer Science and Technology*, Marcel Decker, New York, vol. 28, Supplement 13; also: <http://www.cs.wpi.edu/~dcb/Papers/EofCSandT.html>
- Brooks, RA: 1991, Intelligence without reason, *AI Memo No.1293*, MIT, Cambridge, MA.
- Brooks, RA, Breazeal (Ferrell), C, Irie, R, Kemp, CC, Marjanovic, M, Scassellati, B and Williamson, MM: 1998, *Alternative Essences of Intelligence*, AAAI.
- Brooks, RA and Stein, LA: 1993, Building brains for bodies, *AI Memo No. 1439*, MIT, Cambridge, MA.
- Buchanan, BG and Shortliffe, EH: 1984, *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project*, Addison-Wesley, Reading, MA.
- Capra, F: 1996, *The Web of Life*, Anchor Books, Doubleday, NY.
- Casti, JL: 1990, *Search for Certainty, What Scientist Can Know About the Future*, William Morrow and Co., New York.
- Cusumano, MA and Yoffie, DB: 1999, What Netscape Learned from Cross-Platform Software Development, *Communication of the ACM* 42(10): 72-78.
- Coyne, R: 1995, *Designing Information Technology in the Postmodern Age: From Method to Metaphor*, The MIT Press, Cambridge, MA.
- Crowe, M, Beeby, R and Gammack, J: 1996, *Constructing Systems and Information*, McGraw-Hill, New York.
- Damasio, AR: 1994, *Descartes' Error*, Grosset/Putnam, NY.
- Dautenhahn, K: 1998, The art of designing socially intelligent agents – science, fiction, and the human in the loop. special issue "socially intelligent agents", *Applied Artificial Intelligence Journal* 12(7-8): 573-617.
- Dautenhahn, K: 1999, Socially situated life-like agents: if it makes you happy then it can't be that bad? *Proceedings VWSIM'99, Virtual Worlds and Simulation Conference*; also: <http://homepages.feis.herts.ac.uk/~comqkd/papers.html>
- Dautenhahn, K: 2000, Socially intelligent agents: the human in the loop, *AI Magazine* 22(3): 115-116.

- Extempo: 1999, <http://www.extempo.com/tours/>
- Firefly: 1999, <http://www.firefly.com/>
- Flatow, I: 1992, *They All Laughed, From Light Bulbs to Lasers: The Fascinating Stories Behind the Great Inventions that have Changed our Lives*, Harper Perennial.
- Foucault, M: 1979, *Discipline and Punish: The Birth of the Prison*, Vintage, NY.
- Gaarder, J: 1996, *Sophie's World*, Berkeley Books.
- Gero, JS (ed.): 1992, *Artificial Intelligence in Design'92*, Kluwer, Dordrecht.
- Gibney JR, Luscombe, F and B: 2000, The redesigning of America, *Time Magazine* **155**(11): 66-70.
- Gill, KS: 1991, Summary of human centered systems research in Europe, parts 1 and 2, *Systemist* **13**(1): 7-27; **13**(2): 49- 75.
- Glaserfeld, E von: 1976, Radical Constructivism and Piaget's concept of knowledge, in FB Murray (ed.) *Cognitive Psychology: The Impact of Piaget*, Plenum, N.Y., pp. 109-122.
- Goel, A: 1997, Design, Analogy and Creativity, *IEEE Expert* **12**(3): 62-70.
- Goguen, J: 2001, <http://www-cse.ucsd.edu/users/goguen/courses/275/>
- Hale-Haniff, M and A Pasztor: 1999, Co-constructing subjective experience: a constructivist approach, *Dialogues in Psychology*, **16.0**, <http://hubcap.clemson.edu/psych/oldpage/Dialogues/16.0.html>
- Hersh, R: 1997, *What is Mathematics, Really?* Oxford University Press.
- Isbister, K and Hayes-Roth, B: 1998, Social implications of using synthetic characters: an examination of a role- specific intelligent agent, Knowledge System Laboratory, Report No. KSL 98-01,Stanford University, CA.
- Jirotka, M and Goguen J, (eds.): 1994, *Requirements Engineering: Social and Technical Issues (Computers and People)*, Academic Press, San Diego, CA.
- Johnson, M: 1987, *The Body in the Mind*, University of Chicago Press, Chicago, IL.
- Kjellman, A: 2000, Correspondence with Burton Voorhees, *Modern Science & The Mind*, January 17-May 8, <http://www.consciousness.arizona.edu/msm.html>.
- Koza, JR, Bennett, FH, Andre, D and MA Keane: 1999, *Genetic Programming III: Darwinian Invention and Problem Solving*, Morgan Kaufmann, San Mateo, CA.
- Kuhn, TS: 1962, *The Structure of Scientific Revolutions*, University of Chicago Press, IL.
- Lakoff, G and Núñez, R: 1997, The metaphorical Structure of Mathematics: Sketching Out Cognitive Foundations for a Mind-Based Mathematics, in LD English (ed.), *Mathematical Reasoning, Analogies, Metaphors, and Images*, Lawrence Erlbaum, London, pp. 21-93.
- Latour, B: 1996, *Aramis or the Love of Technology*, Translated by Catherine Porter, Harvard University Press, Cambridge, MA.
- Lincoln, YS and Guba, EG: 1985, *Naturalistic Inquiry*. Sage, Newbury Park, CA.
- Lyon, D: 1994, *The Electronic Eye: The Rise of Surveillance Society*, University of Minneapolis Press.
- Maes, P: 1994, Agents that reduce work and information overload, *Communications of the ACM* **37**(7): 31-40.
- Maturana, H: 1980, Biology of cognition, published originally in 1970, reprinted in H Maturana and F Varela (eds), *Autopoiesis and Cognition*, D. Reidel, Dordrecht.
- Minsky, ML: 2002, *The Emotion Machine*, unpublished book draft, <http://web.media.mit.edu/~minsky/E1/eb1.html>
- Mitchell, T, Caruana, R, Freitag, D, McDermott J and Zabowski, D: 1994, Experience with a learning personal assistant, *Communications of the ACM* **37**(7): 81-91.

- Norman, DA and Draper, S: 1986, *User Centered Systems Design*, Erlbaum, London.
- Norman, DA: 1988, *The Design of Everyday Things*, Doubleday, NY.
- Norman, DA: 1993, *Things that Make us Smart, Defending Human Attributes in the Age of the Machines*, Perseus Books, Reading, MA.
- Noyes, JM and Baber, C: 1999, *User-Centred Design of Systems (Applied Computing)*, Springer Verlag, Berlin.
- NSPE: 1998, Code of Ethics for Engineers: <http://onlineethics.org/codes/NSPEcode.html>
- Participatory: 1998, <http://www.cpsr.org/conferences/pdc98/bibliography.html>
- Participatory: 1998a, *Post-Conference Information of the Participatory Design Conference*, Seattle, Washington, USA, November 12-14, <http://www.cpsr.org/conferences/pdc98/>
- Petroski, H: 1994, *Design Paradigms, Case Histories of Error and Judgement in Engineering*, Cambridge University Press, Cambridge.
- Picard, R: 1997, *Affective Computing*, MIT Press, Cambridge, MA.
- Pool, R: 1997, *Beyond Engineering, How Society Shapes Engineering*, Oxford University Press, NY.
- Ramachandran, VS and Blakeslee, S: 1998, *Phantoms in the Brain: Probing the Mysteries of the Human Mind*, William Morrow & Co, NY.
- Reilly, SN: 1996, *Believable Social & Emotional Agents*, Ph.D. Thesis, TR CMU-CS-96-138, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA.
- Schön, DA: 1984, *The Reflective Practitioner: How Professionals Think in Action*, Basic Books, New York.
- Schuler, D and Namioka, A (eds): 1993, *Participatory Design*, Lawrence Erlbaum, London.
- Schwartz, P and Ogilvy J: 1979, *The Emergent Paradigm: Changing Patterns of Thought and Belief* (SRI International). Cited in Lincoln, Y.S., & Guba, E. G. (1985), *Naturalistic inquiry*, Sage Publications, Newbury Park, CA.
- Sipper, M and Ronald, EMA: 2000, A new species of hardware, *IEEE Spectrum*, March, 59-64.
- Trigg, R: 1993, *Rationality & Science, Can Science Explain Everything*, Blackwell, Oxford, UK.
- Varela, FJ, Thomson, E and Rosch, E: 1999, *The Embodied Mind, Cognitive Science and Human Experience*, The MIT Press, Cambridge, MA.
- Waczlawick, P: 1984, Components of ideological "realities", in P Waczlawick (ed.), *The Invented Reality*, WW Norton & Co, NY, London, pp. 206-248.
- Williams, B: 1978, Descartes: *The Project Of Pure Inquiry*, Penguin, London.
- Wilson, EO: 1998, *Consilience*, A Knopf Inc, NY.

THE SITUATED FUNCTION - BEHAVIOUR - STRUCTURE FRAMEWORK

JOHN S GERO AND UDO KANNENGIESSER

*University of Sydney
Australia*

Abstract. This paper extends the Function-Behaviour-Structure (FBS) framework, which proposed eight fundamental processes involved in designing. This framework did not explicitly account for the dynamic character of the context in which designing takes place, described by the notion of situatedness. This paper describes this concept as a recursive interrelationship between different environments, which, together with a model of constructive memory, provides the foundation of a situated FBS framework. The eight fundamental processes are then reconstructed within this new framework to represent designing in a dynamic world.

1. Introduction

Recent AI in design research has increasingly focussed on developing agent-based design systems (e.g. Campbell et al. (1998), Grecu and Brown (2000)). Yet, many of these approaches have shown only limited success in supporting conceptual designing since they ignore one of conceptual designing's (as opposed to routine designing's) most distinguishing features. As not all the requirements are known at the outset of a design task, conceptual designing involves finding what is needed and modifying it again during the process.

Many agent-based systems are based on traditional models and theories of designing that assume the world as being fixed, well-defined and unchanged by what you do. This static view of the world is not in accord with the results of empirical design research (Schön and Wiggins 1992; Suwa, Gero and Purcell 1999). In order to develop computational design agents as aids to human designers, we need a model of designing in which the knowledge is not encoded *a priori*.

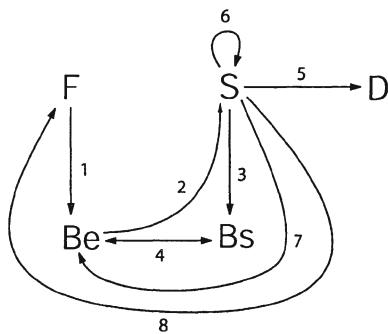
A formal representation of the processes of designing is Gero's (1990) FBS framework. Since its publication, some important insights have been

gained from empirical design research, applying ideas from cognitive science. These can be summarized by the notion of situatedness, which emphasizes that the agent's view of a world changes depending on what the agent does. The FBS framework does not account for this concept and is therefore still unable explicitly to make the move away from encoded knowledge.

The aim of this paper is to develop a situated FBS framework, in which the knowledge of the design agent is grounded in its experience and its interactions with the environment. We present a model of an open, dynamic world consisting of multiple interacting environments.

2. The FBS Framework

The FBS Framework (Gero 1990) represents the developing design in different states. The basic assumption here is the existence of three classes of variables required in a design process: function variables, behaviour variables and structure variables. They are linked together by processes, which transform one class into the other, Figure 1.



Be = expected behaviour Bs = behaviour derived from structure D = design description F = function S = structure	→ = transformation ↔ = comparison
---	--------------------------------------

Figure 1. The FBS framework (after Gero (1990))

The eight processes depicted in the FBS framework are claimed to be fundamental for all designing. They are briefly outlined below:

- *Formulation* (process 1) transforms the design problem, expressed in function (F), into behaviour (Be) that is expected to enable this function.

- *Synthesis* (process 2) transforms the expected behaviour (Be) into a solution structure (S) that is intended to exhibit this desired behaviour.
- *Analysis* (process 3) derives the “actual” behaviour (Bs) from the synthesized structure (S).
- *Evaluation* (process 4) compares the behaviour derived from structure (Bs) with the expected behaviour to prepare the decision if the design solution is to be accepted.
- *Documentation* (process 5) produces the design description (D) for constructing or manufacturing the product.
- *Reformulation type 1* (process 6) addresses changes in the design state space in terms of structure variables or ranges of values for them.
- *Reformulation type 2* (process 7) addresses changes in the design state space in terms of behaviour variables or ranges of values for them.
- *Reformulation type 3* (process 8) addresses changes in the design state space in terms of function variables or ranges of values for them.

The most remarkable kinds of processes (as they do not appear in most traditional models of designing) are those representing the reformulation of the design state space (processes 6, 7 and 8). The most explored process of them is reformulation type 1; common examples are case-based reasoning (Maher, Balachandran and Zhang 1995) and structure analogy (Qian and Gero 1996). Empirical design studies confirm that the reformulation of structure is the predominant type of reformulation (McNeill et al. 1998). The same studies also reveal that the activity of reformulating the problem, in terms of expected behaviour or even function, diminishes during the design process, but never disappears.

The lack of these processes in most other approaches indicates the presence of a fundamentally different view of designing in the FBS framework. All three types of reformulation suggest a non-static world of designing, as they obviously give the on-going design process a new direction that was not anticipated before. However, in its current state, the FBS framework cannot show this open world explicitly.

3. Situatedness and Constructive Memory

Designing is an activity during which the designers perform actions in order to change the environment. By observing and interpreting the results of their actions, they then decide on new actions to be executed on the environment. This means that the designer’s concepts may change according to what they

are “seeing”, which itself is a function of what they have done. We may speak of a recursive process, an “interaction of making and seeing” (Schön and Wiggins, 1992). This interaction between the designer and the environment strongly determines the course of designing. This idea is called *situatedness*, whose foundational concepts go back to the work of Dewey (1896) and Bartlett (1932). In paraphrasing Clancey (1997) we can summarize it as “where you are when you do what you do matters”.

In experimental studies of designers some phenomena related to the use of sketches, which support this idea, have been reported. Schön and Wiggins (1992) found that designers use their sketches not only as an external memory, but also as a means to reinterpret what they have drawn, thus leading the design in a new direction. Suwa, Gero and Purcell (1999) noted, in studying designers, a correlation of unexpected discoveries in sketches with the invention of new issues or requirements during the design process. They concluded that “sketches serve as a physical setting in which design thoughts are constructed on the fly in a situated way”.

An important idea, which fits into the notion of situatedness, has been proposed by Dewey in 1896 (Clancey 1997) and is today called *constructive memory*. The main idea of constructive memory is that memory, instead of being laid down and fixed at the time of the original experience, must be newly constructed every time there needs to be a memory. Certainly, the original experience, which is to be recalled, is used to construct the memory of it. But this process is also governed by the situation pertaining at the time of the demand for this memory. Therefore, everything that has happened since the original experience determines the result of memory construction. Each memory, after being constructed, is added to the experience and thus becomes part of the situation, which affects the kinds of further memories that can be constructed.

Memory, as an overall term, must be seen as a process rather than a fixed state. This idea has been exemplified by a quote from Dewey via Clancey: “Sequences of acts are composed such that subsequent experiences categorize and hence give meaning to what was experienced before”. The significance of the idea of constructive memory in designing has been shown by Gero (1999). Situatedness and constructive memory thus provide the conceptual basis for grounding the knowledge of an agent in the situation being constructed by its interactions with the environment.

4. Creating a Dynamic Context for Designing

We want to develop a setting for representing designing in an open, dynamic context. The ideas of situatedness and constructive memory, founded on the constructive character of human cognition and grounding it in processes of interaction, provide a suitable conceptual basis for this endeavour. However,

these ideas have to be developed further as well as modelled in more formal ways.

4.1 MODELLING SITUATEDNESS

We will approach situatedness by introducing three different kinds of environments that interact with one another, Figure 2.

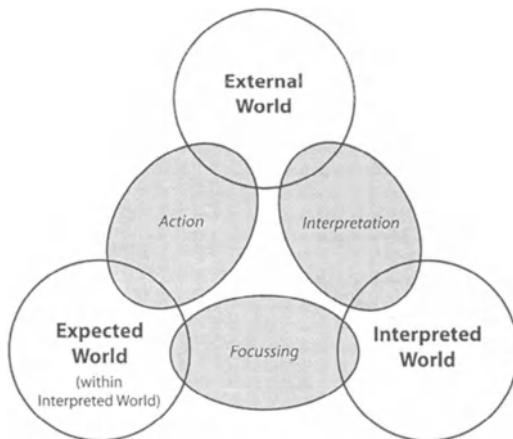


Figure 2. Situatedness as the interaction of three worlds

The *external world* is the world that is composed of representations outside the designer or design agent.

The *interpreted world* is the world that is built up inside the designer or design agent in terms of sensory experiences, percepts and concepts. It is the internal representation of that part of the external world that the designer interacts with.

The *expected world* is the world imagined actions will produce. It is the environment in which the effects of actions are predicted according to current goals and interpretations of the current state of the world.

These three worlds are recursively linked together by three classes of processes. The process of *interpretation* transforms variables which are sensed in the external world into the interpretations of sensory experiences, percepts and concepts that compose the interpreted world. This is done by the interaction of sensation, perception and conception processes (Gero and Fujii 2000). The process of *focussing* focuses on some aspects of the interpreted world, uses them as goals in the expected world and suggests

actions, which, if executed in the external world should produce states that reach the goals. The process of *action* is an effect which brings about a change in the external world according to the goals in the expected world.

We have depicted the expected world separately from the interpreted world to be able to explicitly delineate some important concepts and processes. However, it is important to note that the expected world is located within the interpreted world.

The different environments, thus connected to one another, form the *situation*, which thus consists of both the external and the designer's internal¹ world. The dynamics of the situation stem from the interaction of these three kinds of environments. Potentially, every change in one of the worlds brings about and is brought about by changes in the other worlds.

4.2 MODELLING CONSTRUCTIVE MEMORY

We will now develop in more detail the internal world itself. Here we are dealing with the agent's knowledge that has been constructed and processed from former interactions with the external world. The notion of constructive memory captures this idea, however it still needs more examination to be formally introduced into this framework of designing.

Constructing a memory has been described as being governed both by what was initially there (the original experience) and by what the current situation (made up by previous experiences and memories and the currently focussed concepts) makes with it. At this level of abstraction, constructive memory can be viewed in a similar way as the process of interpretation, which Gero and Fujii (2000) have described as consisting of two parallel processes interacting with each other: A *push process* (or data-driven process), where the production of an internal representation is driven ("pushed") by the sensed data, and a *pull process* (or expectation-driven process), where the interpretation is driven ("pulled") by some of the agent's current concepts, which has the effect that the original data is biased to match the current expectations (about what the interpretation should be).

In this respect, we can use the idea of a push-pull process for generally representing the interaction of an agent with both its external environment (by interpretation) and its internal environment (by constructive memory). Figure 3 depicts how an original experience (E_0) is produced by interpreting something in the external world at a certain point of time. A push-pull process represents this transition from the external world to the interpreted world. The construction of a memory (M_1) of the original experience, at a

¹ We use the term "internal world" to denote the interpreted world including the expected world.

later point of time, is also done by a push-pull process, but within the interpreted world. The pull process here is controlled by the current situation.

Figure 3 shows not only that the processes of interpretation and constructive memory can be represented in the same manner, but also that the result of one can influence the result of the other. Interactions of the interpreted world with the external world can have an impact on the construction of memories, and interactions of the interpreted world with itself can affect the construction (interpretation) of concepts.

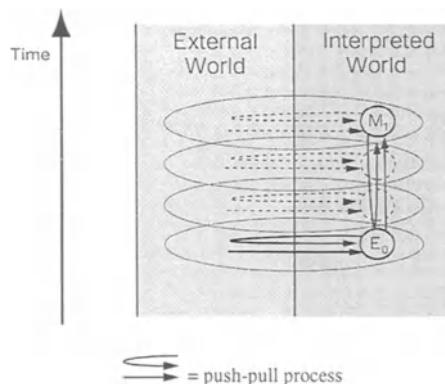


Figure 3. Interpretation and constructive memory

The interpreted world, as depicted in Figure 3, thus “wires itself up” through intertwined horizontal and vertical push-pull processes. Exploring these processes is out of scope of this paper. However, we now have a sufficient description of constructive memory for integrating into a framework of designing. It contributes to our conception of an open multiple-world environment by providing an additional dynamic component inside the agent.

5. The Processes of Designing in a Situated Perspective

Having described situatedness and constructive memory, we are now able to develop a situated framework of designing. Our conception of a dynamic environment as three interacting worlds provides an initial setting, into which we can put the eight fundamental processes in designing. Figure 4 shows these worlds again, similar to Figure 2, however nesting them to imply the design agent (as the internal world) is located within the external world. It also represents more explicitly that the expected world is a subset of the interpreted world.

Figure 4 also shows those general classes of processes that have been developed and formalized in Section 4. A push-pull process represents the interaction of the interpreted world and the external world (via interpretation) as well as the interpreted world interacting with itself (via constructive memory). The process of focussing connects the interpreted world and the expected world and is represented by a double-headed arrow. This accounts for the bi-directional character of this process, which ensures that some concepts may be focussed (and thus transferred into the expected world), whereas other concepts, which have been previously focussed, may be dropped (and thus transferred out of the expected world). The process of action is depicted as a transformation of an expected concept into an external representation.

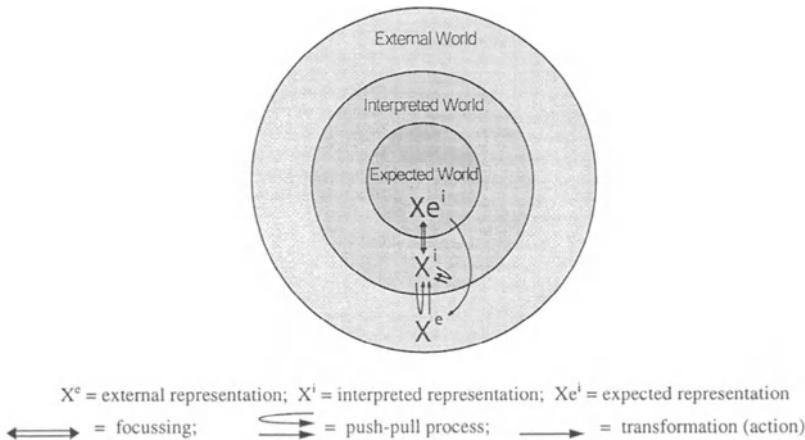


Figure 4. A dynamic setting for the design process

We will now develop a situated FBS framework using the foundations provided in Figure 4. This requires specializing each of the general concepts of external, interpreted and expected representations (X^e , X^i and Xe^i) with respect to the three classes of variables F, B and S.

The following step-by-step reconstruction of each of Gero's (1990) eight fundamental processes in designing from our new situated perspective will illustrate how these processes have been developed as a consequence of distinguishing between representations in different worlds.

5.1 FORMULATION

Formulation is an important process in conceptual designing, as it specifies an initial design state space, within which the design solution is searched.

The original FBS framework does not provide a representation for the notion of a design state space, however the expected world in our new setting can do exactly that. This gives us the possibility to show the set of processes that produces a complete design state space in terms of F, B and S. To better exploit this capability, we explicitly depict the requirements (R) of a design problem.

The original FBS framework, due to the lack of a representation for the design state space (which is a consequence of the lack of the differentiation between different worlds), is restricted to the reasoning process from function (F) to expected behaviour (Be), Figure 5(a). In a situated environment, Figure 5(b), this activity is only one part of the process of formulation. First, the design agent interprets the explicit requirements (R) by producing the interpreted representations F^i and, eventually, B^i and S^i (processes 1, 2 and 3), which are then augmented by representations (implicit requirements) originating from the agent's own experience (processes 4, 5 and 6). An initial design state space is set up by focussing on subsets on these internalised (explicit and implicit) requirements (processes 7, 8 and 9). Process 10 corresponds to the transformation of F into Be in the original FBS framework.

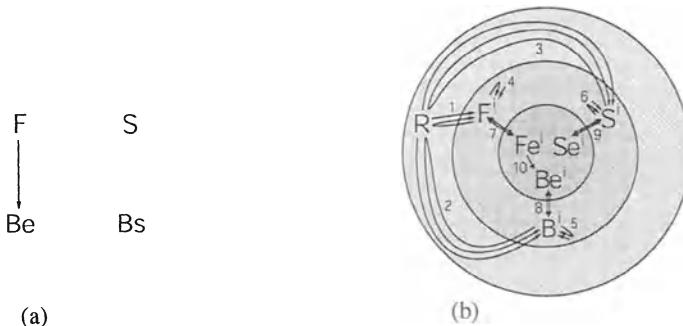


Figure 5. Formulation: (a) original, (b) situated FBS framework

5.2 SYNTHESIS

Synthesis is the process of transforming expected behaviour into external structure. Figure 6(b) depicts this with two processes, one producing an expected structure (process 11) and one externalising it e.g. by sketching or some similar process (process 12). The original FBS framework did not distinguish between expected and external structure and therefore could only show the general process from Be to an unspecified S, Figure 6(a).

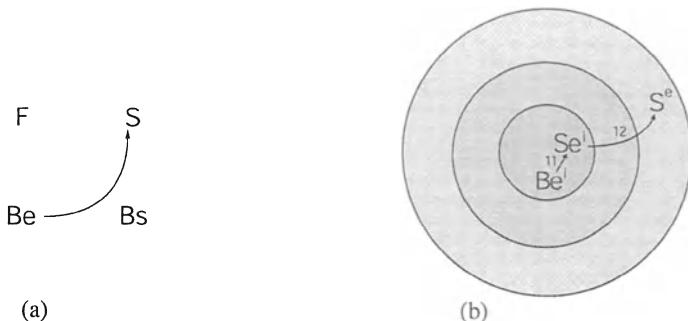


Figure 6. Synthesis: (a) original, (b) situated FBS framework

5.3 ANALYSIS

Analysis, the derivation of behaviour from a synthesized (and thus external) structure, is represented by the two partial processes in Figure 7(b): the construction of an interpreted structure through interpretation (process 13) and its subsequent transformation into interpreted behaviour (process 14). Only the latter is explicitly represented in Figure 7(a), without the interpretation stage of the analysis process.

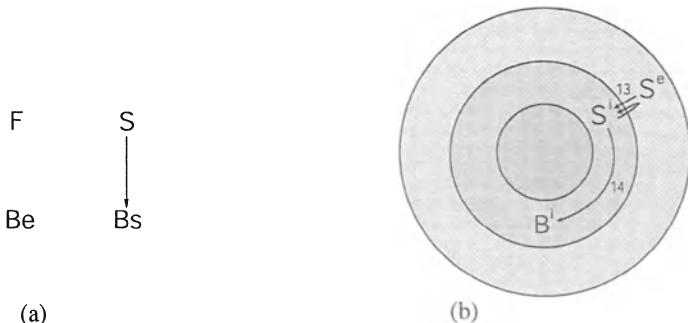


Figure 7. Analysis: (a) original, (b) situated FBS framework

5.4 EVALUATION

The original FBS framework already distinguished between expected behaviour B_e and behaviour derived from structure B_s (which corresponds, in this particular case, to interpreted behaviour B^i). Therefore the evaluation process, which compares these two concepts, remains unchanged in our situated FBS framework, Figures 8(a) and 8(b).

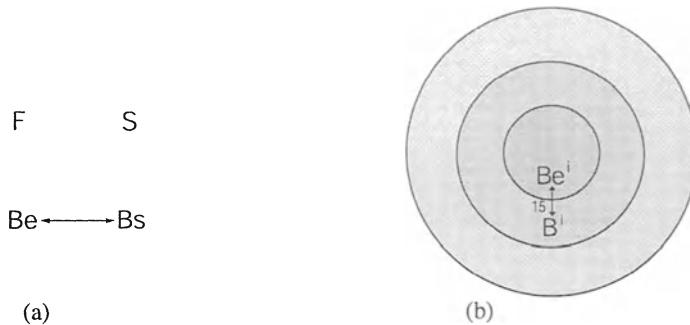


Figure 8. Evaluation: (a) original, (b) situated FBS framework

5.5 DOCUMENTATION

This process denotes the production of the (external) description of a design solution. The original FBS framework described this process as the transformation of S into D, Figure 9(a). In the situated view, Figure 9(b), we generalise the design description as the (standardised) external representation of expected structure (process 12) and, if needed to be represented, expected behaviour (process 17) and expected function (process 18).

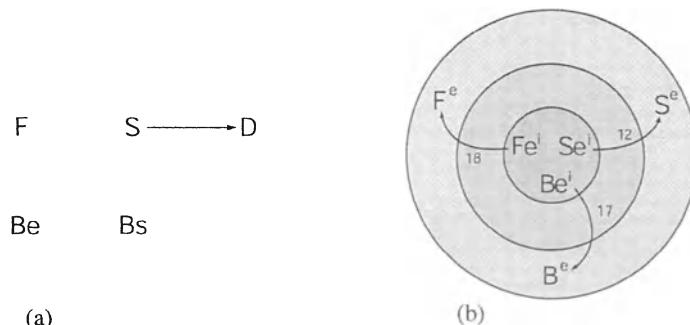


Figure 9. Documentation: (a) original, (b) situated FBS framework

5.6 REFORMULATION TYPE 1

Reformulation of structure addresses changes in the structure state space during designing. In the original FBS framework, the starting point of this process is an existing structure S, Figure 10(a), from which new structure

variables are introduced in the structure state space. Yet it cannot specify if this existing structure is external or internal to the design agent. The situated FBS framework, Figure 10(b), represents the provenance of new variables needed for reformulation in a more detailed manner, as it is capable of distinguishing between these two types of structure. Consequently, it shows two different processes, any one of which can trigger the reformulation of structure (process 9): the interpretation of external structure (process 13) and the construction of a memory related to structure (process 6).

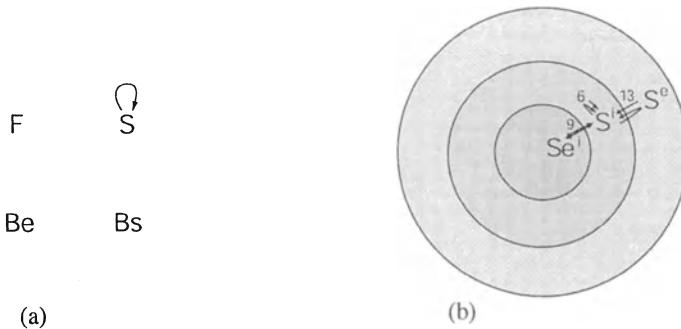


Figure 10. Reformulation type 1: (a) original, (b) situated FBS framework

5.7 REFORMULATION TYPE 2

Reformulation of expected behaviour addresses changes in the behaviour state space during designing. The original FBS framework, Figure 11(a), proposed only an existing structure as the generator for new behaviour variables. The situated FBS framework, Figure 11(b), provides a richer view on the reformulation of expected behaviour (process 8). Besides the derivation of interpreted behaviour from interpreted structure (process 14), it also depicts the interpretation of an external behaviour (process 19) and the internal construction of an interpreted behaviour (process 5) as the possible drivers of this type of reformulation.

5.8 REFORMULATION TYPE 3

Reformulation of function addresses changes in the function state space during designing. The original FBS framework, Figure 12(a), commences only with an existing structure that generates new function variables via new behaviour variables. The situated FBS framework, Figure 12(b), proposes a larger number of processes that can drive the reformulation of function (process 7): the ascription of interpreted function to interpreted behaviour

(process 16), the interpretation of external function (process 20) and the internal construction of an interpreted function (process 4).

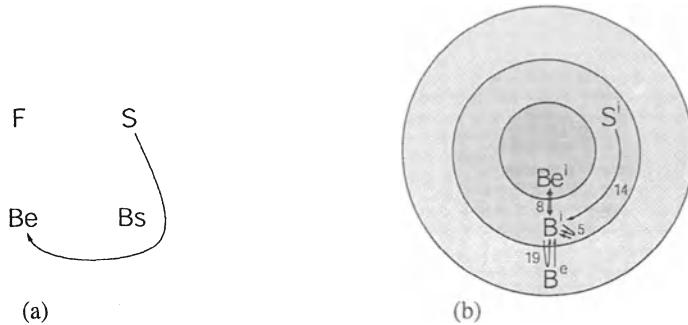


Figure 11. Reformulation type 2: (a) original, (b) situated FBS framework

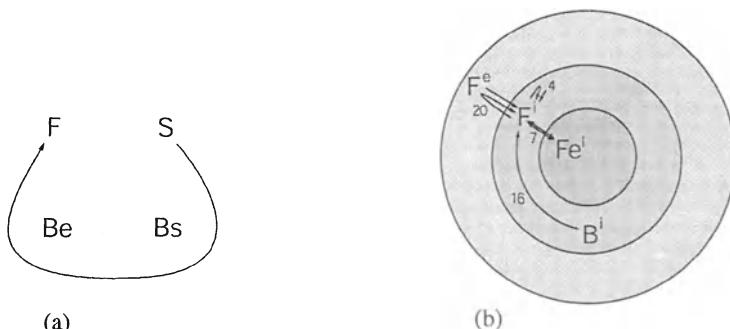


Figure 12. Reformulation type 3: (a) original, (b) situated FBS framework

6. Conclusion: The Situated FBS Framework

The reconstruction of the eight fundamental processes in designing in our situated world has brought about a set of processes, which can now be represented together, thus composing the situated FBS framework, Figure 13.

As can be seen, the number of processes depicted in the new framework, which is now 20, has steeply risen in relation to the previous number. This is a consequence of the capacity of the situated framework to deal with the agent's interaction processes with and within the external world, namely those of interpretation, constructive memory, focussing and action. The original FBS framework was unable to make these activities explicit. This

was due to its restricted perspective, which could only represent processes involving transformations between the three different classes of design variables within one world (in terms of our three world model). An exception here was the explicit division between "expected behaviour" and "behaviour derived from structure". Although the scope of the latter term is too specific to cover all interpreted behaviour (B^i), this distinction fits into the conceptual idea of expected and interpreted worlds.

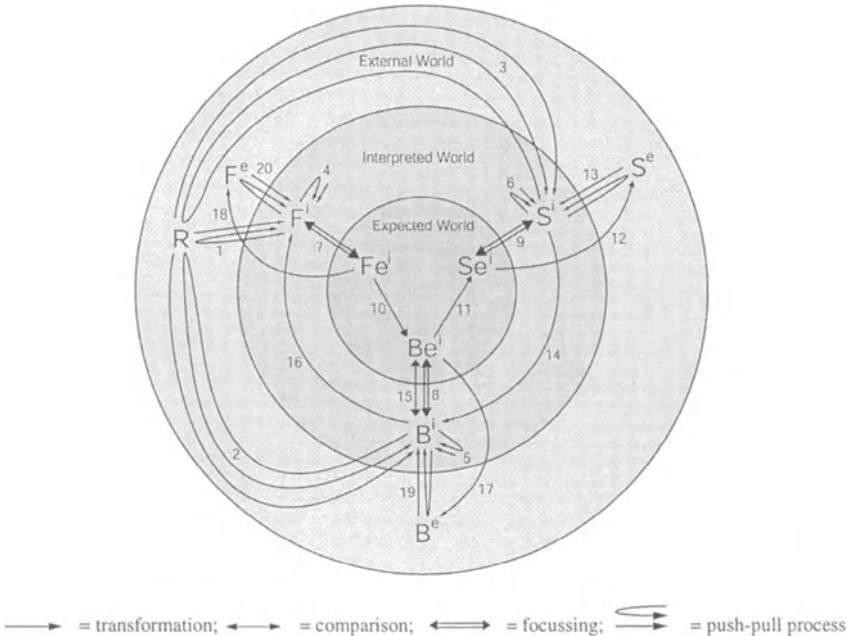


Figure 13. The situated FBS framework

The overall achievement of the situated FBS framework as presented in this paper is its contribution to a better understanding of designing in an open, dynamic world. While still describing a set of distinguishable processes, conjointly mapping on to the eight fundamental processes specified in the original framework, it succeeds in formally integrating the idea of situatedness.

The situated FBS framework presented in this paper opens up a number of possibilities for future research. A new conception of design knowledge representation is conceivable, building on Gero's (1990) design prototypes. These are generalized schemas derived from sets of like design cases, which

unite all the relevant knowledge necessary for designing. Making these design prototypes situated would enhance their applicability in dynamic design contexts.

This framework provides a new foundation for the development of intelligent agent-based design systems. This is mainly because it brings together important concepts of situated agents (like interpretation, focussing and constructive memory) and the three basic variables function, behaviour and structure making up the design world. This ability to deal also with design concepts like behaviour and function, besides structure, can make situated design agents potentially powerful enough to support human designers in the conceptual stages of designing.

Acknowledgements

This research is supported by a Sesqui Research and Development grant from the University of Sydney and by an International Postgraduate Research Scholarship.

References

- Bartlett, FC: 1932 reprinted in 1977, *Remembering: A Study in Experimental and Social Psychology*, Cambridge University Press, Cambridge.
- Campbell, MI, Cagan, J and Kotovsky, K: 1998, A-Design: Theory and implementation of an adaptive, agent-based method of conceptual design, in JS Gero and F Sudweeks (eds), *Artificial Intelligence in Design '98*, Kluwer, Dordrecht, pp. 579-598.
- Clancey, WJ: 1997, *Situated Cognition*, Cambridge University Press, Cambridge.
- Dewey, J: 1896 reprinted in 1981, The reflex arc concept in psychology, *Psychological Review* 3: 357-370.
- Gero, JS: 1990, Design prototypes: A knowledge representation scheme for design, *AI Magazine* 11(4): 26-36.
- Gero, JS: 1999, Constructive memory in design thinking, in G Goldschmidt and W Porter (eds), *Design Thinking Research Symposium: Design Representation*, MIT, Cambridge, MA, pp. 29-35.
- Gero, JS and Fujii, H: 2000, A computational framework for concept formation for a situated design agent, *Knowledge-Based Systems* 13(6): 361-368.
- Grecu, DL and Brown, DC: 2000, Expectation formation in multi-agent design systems, in JS Gero (ed.): *Artificial Intelligence in Design '00*, Kluwer, Dordrecht, pp. 651-671.
- Maher, ML, Balachandran, MB and Zhang, DM: 1995, *Case-Based Reasoning in Design*, Lawrence Erlbaum, Hillsdale, NJ.
- McNeill, T, Gero, JS and Warren, J: 1998, Understanding conceptual electronic design using protocol analysis, *Research in Engineering Design* 10: 129-140.
- Qian, L and Gero, JS: 1996, Function-behaviour-structure paths and their role in analogy-based design, *AIEDAM* 10: 289-312.
- Schön, D and Wiggins, G: 1992, Kinds of seeing and their functions in designing, *Design Studies* 13(2): 135-156.

Suwa, M, Gero, JS and Purcell, T: 1999, Unexpected discoveries and s-inventions of design requirements: A key to creative designs, in JS Gero and ML Maher (eds), *Computational Models of Creative Design IV*, Key Centre of Design Computing and Cognition, University of Sydney, Sydney, Australia, pp. 297-320.

SPATIAL SYNTHESIS AND ANALYSIS

Analysis of architectural space composition using inductive logic programming
Noritoshi Sugiura and Shigeyuki Okazaki

Towards an architectural design system based on generic representations
Sviataslau Pranovich, Henri Achten and Jarke J Van Wijk

Digital sandbox
Ellen Yi-Luen Do

REPRESENTATIONAL FLEXIBILITY FOR DESIGN

RUDI STOUFFS

*Delft University of Technology
The Netherlands*

AND

RAMESH KRISHNAMURTI
*Carnegie Mellon University
USA*

Abstract. We present an abstraction of representational schema to model *sorts* that allows us to explore the mathematical properties of a constructive approach to *sorts*. We apply this approach to representational schema defined as compositions of primitive data types, and explore a comparison of representational structures with respect to scope and coverage. We consider a behavioral specification for *sorts* in order to empower these representational structures to support design activities effectively. We provide an example of the use of *sorts* to represent alternative views to a design problem. We conclude with a comparison with other approaches for flexibility of design representations.

1. Introduction

A variety of design problems requires a multiplicity of viewpoints each distinguished by particular interests and emphases. For instance, the architect is concerned with aesthetic and configurational aspects of a design, a structural engineer is engaged by the structural members and their relationships, and a performance engineer is engaged by the thermal, lighting, or acoustical performance of the eventual design. Each of these views – derived from an understanding of current problem solution techniques in these respective domains - require different representations of the same entity. The work described in this paper is based on the recognition that there will always be a need for different representations of the same entity, albeit a building or building part, a shape or other complex attribute.

This exigency ensues, formally, to define the relations between alternative representations, in order to support translation and identify where exact translation is possible, and to define coverage of different representations.

In order to support representational flexibility for design, a framework must be conceived that provides support for exploring alternative design representations, for comparing design representations with respect to scope and coverage, and for mapping design information between representations, even if their scopes are not identical. Typically, a representation is a complex structure of attributes and constructors, and a representation may be a construction of another (Stouffs, Krishnamurti and Eastman 1996). Comparing different representations, therefore, requires a comparison of the respective attributes, their mutual relationships, and the overall construction. On the other hand, the expressive power of a representational framework is defined by its vocabulary of primitive attributes (or data types) and constructors. A proper definition of this representational framework and its vocabulary can give designers the freedom and flexibility to develop or adopt representations that serve their intentions and needs, while at the same time these representations can be formally compared with respect to scope and coverage in order to support information exchange. Such a comparison will not only yield a possible mapping, but also uncover potential data loss when moving data from less-restrictive to more-restrictive representations.

Requicha (1980) defines a representational schema as a relationship between, on one hand, representations as concrete descriptions and, on the other hand, models as the abstract entities described. Models are considered mathematical abstractions of real-world or designed entities; representations are symbolic descriptions that can be conceived and manipulated using a computer. Each particular representation describes a single model but generally adheres to a global descriptive convention as expressed by the representational schema. We seek to develop a formalism that is based on a general description and, at the same time, applies to the particular representational schema that we are ultimately interested in. Hereto, we consider an abstraction of representational schema to model *sorts* that allows us to explore the mathematical properties of a constructive approach to *sorts*. We then apply this constructive approach to representational schema defined as compositions of primitive data types, and explore a comparison of representational structures with respect to scope and coverage. We consider a behavioral specification for *sorts* in order to empower these representational structures to support design activities effectively. We provide an example of the use of *sorts* to represent alternative views to a design problem. We conclude with a comparison with other approaches for flexibility of design representations.

2. A Conceptual Definition of *Sorts*

A *sort* constitutes the basic entity for our formalism. Conceptually, a *sort* may define a set of similar data elements, e.g., a class of objects or the set of tuples solving a system of equations. For example, points and lines each are a *sort* and so are triangles and squares. *Sorts* are not limited to geometrical objects, colors are a *sort* and other attributes can also define *sorts*. When described by a system of equations, the solutions to this system specify the data elements with respect to some chosen universe, e.g., lines may be expressed as infinite sets of points in a Euclidean space, colors as entities in an RGB (red-green-blue) or HSI (hue-saturation-intensity) space. Within such a system of equations, we can distinguish two types of parameters and equations. Characteristic parameters serve to define a generalized data element within its universe, e.g., a line in Euclidean space, or a color in RGB space. Instance parameters, on the other hand, identify each data element within the *sort*, e.g., a particular line in a *sort* of lines. For example, in the description of a (cartesian) coordinate *sort*, a characteristic parameter x specifies the coordinate as an entity on a cartesian axis, and an instance parameter x_c serves to represent each value x may take within this *sort*, $x = x_c$. Similarly, characteristic equations characterize a generalized data element, while instance equations bound the set of valid data elements. In the description of the coordinate *sort*, $x = x_c$ is the characteristic equation; an optional instance equation may take the form $x_c \geq 0$, limiting the coordinates to positive values.

If for a *sort* a , \mathbf{A}_c denotes the system of characteristic equations, \mathbf{A}_i the system of instance equations, $A^c = \{a_1^c, \dots, a_n^c\}$ the set of characteristic parameters, and $A^i = \{a_1^i, \dots, a_n^i\}$ the set of instance parameters, then, the representation of a can be written symbolically as follows,

$$\begin{array}{|c c|} \hline & \mathbf{A}_c & A^c \\ \hline & \mathbf{A}_i & A^i \\ \hline \end{array} \quad (1)$$

For example, a *sort* of positive coordinates may be specified as

$$\begin{array}{|c c|} \hline & x = x_c & \{x\} \\ \hline & x_c \geq 0 & \{x_c\} \\ \hline \end{array} \quad (2)$$

Similarly, a *sort* of infinite lines may be specified as

$$\begin{array}{|c c c|} \hline & (x_2 - x_1)(y - y_1) = (y_2 - y_1)(x - x_1) & \{x, y\} \\ \hline & x_1 \neq x_2 \vee y_1 \neq y_2 & \{x_1, y_1, x_2, y_2\} \\ \hline \end{array} \quad (3)$$

Additional instance equations may limit which lines belong to the *sort*.

Sorts combine into new *sorts*. A *composite sort* can be defined as the result of an operation on two or more operand *sorts*; sometimes, an expression in terms of a single system of equations, as described above, may exist. Given two *sorts* a and b , the *sort* of all data elements that belong to a or b is the result of the operation of sum, $a + b$. All data elements that belong to both a and b define the result of the operation of product (or intersection), $a \cdot b$. The result is zero, $a \cdot b = 0$, if no data elements belong to both a and b . The result of the operation of difference, $a - b$, is the sort of all data elements that belong to a but not to b . Data elements from a and b can also be combined into 2-tuples under the operation of cartesian product: $a \times b$ contains all 2-tuples of which the first member belongs to a and the second to b .

For each operation, one or more rules can be declared that govern when the expression can be reduced to a single system of equations. For instance, the product $a \cdot b$ is non-zero if the following requirements hold (we refer to (Stouffs and Krishnamurti 1998) for an elaborate account, as well as for a proper definition of the terms *domains*, *identifiable*, and *equivalent* within their respective contexts):

- the characteristic parameters of a and b have identical *domains*, $A^c = B^c$
- the instance parameters of a and b are *identifiable*, $A^i \equiv B^i$
- the characteristic equations of a and b are *equivalent*, $\mathbf{A}_c \equiv \mathbf{B}_c$

The instance equations of $a \cdot b$ are a composition of the instance equations of a and b under the logical connective \wedge (and). In the case that the instance equations of a and b exclude each other, then, $a \cdot b = 0$. The following rules apply:

$$\begin{aligned} & \left\| \begin{array}{cc} \mathbf{A}_c & A^c \\ \mathbf{A}_i & A^i \end{array} \right\| \cdot \left\| \begin{array}{cc} \mathbf{B}_c & B^c \\ \mathbf{B}_i & B^i \end{array} \right\| \rightarrow \left\| \begin{array}{cc} \mathbf{A}_c \equiv \mathbf{B}_c & A^c = B^c \\ \mathbf{A}_i \wedge \mathbf{B}_i & A^i \equiv B^i \end{array} \right\|; \\ & \left\| \begin{array}{cc} \mathbf{A}_c & A^c \\ \mathbf{A}_i & A^i \end{array} \right\| \cdot \left\| \begin{array}{cc} \mathbf{B}_c & B^c \\ \mathbf{B}_i & B^i \end{array} \right\| \rightarrow 0, \text{ otherwise} \end{aligned} \tag{4}$$

The difference of two *sorts* is dependent on the existence of another *sort* that is a common part of both *sorts*. Otherwise, if $a \cdot b = 0$, then, $a - b = a$. If a and b have identical domains and equivalent characteristic equations, then, the instance equations of $a - b$ are a composition of the instance equations of a and the negation (\neg) of the instance equations of b , under the logical connective \wedge . The following rules result:

$$\begin{aligned}
 \left\| \begin{array}{cc} \mathbf{A}_c & A^c \\ \mathbf{A}_i & A^i \end{array} \right\| - \left\| \begin{array}{cc} \mathbf{B}_c & B^c \\ \mathbf{B}_i & B^i \end{array} \right\| &\rightarrow \left\| \begin{array}{cc} \mathbf{A}_c \equiv \mathbf{B}_c & A^c = B^c \\ \mathbf{A}_i \wedge \neg \mathbf{B}_i & A^i \equiv B^i \end{array} \right\|; \\
 \left\| \begin{array}{cc} \mathbf{A}_c & A^c \\ \mathbf{A}_i & A^i \end{array} \right\| - \left\| \begin{array}{cc} \mathbf{B}_c & B^c \\ \mathbf{B}_i & B^i \end{array} \right\| &\rightarrow \left\| \begin{array}{cc} \mathbf{A}_c & A^c \\ \mathbf{A}_i & A^i \end{array} \right\|, \text{ otherwise}
 \end{aligned} \tag{5}$$

Other rules can be specified for the operations of sum, and cartesian product (Stouffs and Krishnamurti 1998). However, these do not always reduce to a single system of equations; given two *sorts* a and b that do not have identical domains, their sum $a + b$ cannot be reduced. Instead, by using distributive rules, any expression of *sorts* over the operations of sum and cartesian product (as well as product and difference) can be reduced to a semi-canonical representational form using sum and cartesian product over at most two levels. The top level specifies a composition over sum of one or more *sorts*, each of which is a composition over cartesian product, at the second level. In this tree structure, the leaf nodes correspond to *sorts* that can be expressed through a single system of equations. Naturally, either or both levels may be absent depending on the particular *sort*. As an example, consider the *sorts* of points, colors and labels: the *sort* of all points with either a color or a label assigned is identical to the *sort* of all colored and all labeled points; $\text{points} \times (\text{colors} + \text{labels}) = \text{points} \times \text{colors} + \text{points} \times \text{labels}$. The following distributive rules apply:

$$\begin{aligned}
 a \times (b + c) &\rightarrow a \times b + a \times c \\
 (a + b) \times c &\rightarrow a \times c + b \times c
 \end{aligned} \tag{6}$$

Furthermore, a subsumption relationship, \leq , may be defined over *sorts*:

$$a \leq b \Leftrightarrow a \cdot b = a \tag{7}$$

From the semi-canonical form described above, we can derive some rules that govern when a *sort* may be a part of another *sort*. Consider the following classification on the universe of sorts: let D_0 be the set of *sorts* that can be expressed by a single system of equations (0 composition levels), let D_1 be the set of *sorts* that can be expressed using only the operation of cartesian product (1 composition level), and let D_2 be the set of *sorts* that include the operation of sum (1 or 2 composition levels). Then, for $a \in D_i$ to be a part of $b \in D_j$, j must either be equal to i or equal to 2 (Stouffs and Krishnamurti 1998). If i and j are 0, rules (4) specify when a is a part of b , or $a \cdot b = a$: provided the characteristic equations are identical, a *sort* subsumes another *sort* if its instance equations form a subsystem of the other *sort*'s instance equations.

The subsumption relationship facilitates the comparison of *sorts* in terms of scope and coverage. For example, the sum of two *sorts* has both operands as a part. Similarly, the product of two *sorts* forms a part of both *sorts*, while the operations of product and difference define a classification of a *sort* with respect to another *sort* into disjoint parts.

3. A Constructive Approach to Representational Structures

For all practical purposes, we consider elementary data types as building blocks for the construction of *sorts* as representational structures. The representational structure of a *primitive sort*, corresponding to an elementary data type, may be the tuple of values that correspond to the instance parameters of the mathematical expression of this same *sort*. The systems of characteristic and instance equations are encoded in the definition of the primitive *sort* in order to allow for an interpretation of the representational structure. Since instance equations constrain the set of valid data elements, the definition of a primitive *sort* may include a number of arguments that offer access to (some of) these constraints when instantiating a primitive *sort*. At instantiation, a primitive *sort* is also assigned a name, in order to semantically distinguish *sorts* that are, otherwise, syntactically identical.

3.1 FORMAL COMPOSITIONAL OPERATORS

Primitive *sorts* combine to *composite sorts* under formal compositional operations over *sorts*. These formal characteristics should derive from the conceptual operations considered above, defining a subsumption relationship that facilitates the comparison of sorts, and offering reduction rules and distribution rules that give access to a semi-canonical form. At the same time, these operations must reflect on semantic relationships that make sense in a representational definition of design data. For example, the operation of *sum* allows for disjunctively co-ordinate compositions of multiple *sorts*; a resulting data collection combines the different types of data elements from its operand *sorts* without imposing any hierarchical relationships. The resulting representational structure distinguishes all operand structures such that each data element belongs explicitly to one of the operand *sorts*. For example, a *sort* of points and lines distinguishes each data element as either a point or a line. Even if both operand *sorts* are syntactically identical, though semantically distinguished, all data elements are still recognized as belonging to one *sort* or the other. For example, an expression of a rule has both a *lhs* (left-hand-side) and *rhs* (right-hand-side)

of the same composite data type; any data element must belong either to the *lhs* or *rhs*. Conceptually, a rule element might instead be considered as a 2-tuple, consisting of a *lhs* and *rhs* element, in accordance to the operation of cartesian product. However, in a general expression of a rule, either of the *lhs* and *rhs* components may be omitted, as allowed by the operation of sum.

The *attribute* operator, instead, specifies a subordinate composition of *sorts*. The resulting representational structure is a combination of both operand structures under an object-attribute relationship. For example, a *sort* of labeled points is specified as a *sort* of points, with one or more labels assigned to each point in the data collection. Similar to the operation of cartesian product, the attribute operation is non-commutative. Other interpretations of the operation of cartesian product can also be considered. Conceptually, a composition of the operation of cartesian product to the n^{th} degree enables the definition of a *sort* of n -tuples of data elements from the respective operand *sorts*. Correspondingly, a *vector* operator can be defined that allows for conjunctively co-ordinate compositions of multiple *sorts*. More interestingly, a *grid* operator can allow for co-ordinate compositions of a single *sort*, with a specification of the grid size according to one, two, or more dimensions. Within the resulting representational structure, operand data elements are distinguished by the grid location these are assigned to. As an example, a grid operation may be useful in tiling design in order to distinguish the individual tiles in a larger composition.

Considering the attribute operation, denoted ‘ \wedge ’, instead of the operation of cartesian product, all associative and distributive rules that apply to *sorts* conceptually (Stouffs and Krishnamurti 1998), still apply to *sorts* as representational structures, e.g.:

$$\begin{aligned} a \wedge (b \wedge c) &\rightarrow a \wedge b \wedge c \\ (a \wedge b) \wedge c &\rightarrow a \wedge b \wedge c \\ a \wedge (b + c) &\rightarrow a \wedge b + a \wedge c \\ (a + b) \wedge c &\rightarrow a \wedge c + b \wedge c \end{aligned} \tag{8}$$

However, the reduction to a semi-canonical form is complicated by the ability to semantically identify a *sort*, that is, to assign a name to a *sort*. Let ‘ $:$ ’ denote the operation of semantic identification. Then, consider the following associative rule over the attribute operation:

$$a \wedge (d : b \wedge c) \rightarrow a \wedge b \wedge c \tag{9}$$

Both sides to the rule are syntactically equivalent, i.e., in both cases a data collection contains data elements from *a*, each of which has an attribute collection consisting of data elements from *b*, again, each of which has an attribute collection consisting of data elements from *c*. However,

semantically, the attribute collections of data elements of a are defined and identified as belonging to d or $b \wedge c$, respectively. When attempting to interpret and deal with a *sort*, such a distinction may be important, as it offers a simple way of limiting the depth at which a *sortal* specification must be minimally considered. Therefore, though rule (9) may be used when comparing *sorts* with respect to scope and coverage, this reduction is not automatically applied in the definition of a *sort*. Similarly, the following distributive rule only serves the comparison of *sorts*:

$$a \wedge (d : b + c) \rightarrow a \wedge b + a \wedge c \quad (10)$$

The situation is even more complex if, under an attribute operation, the first argument is a named composite *sort*, as in the case of $(d : a \wedge b) \wedge c$. Under the attribute operator, data collections are assigned as attribute to individual data elements. In the *sort* $d : a \wedge b$, these are the individual data elements of b within collections that are themselves assigned as attribute to elements of a . A representational structure that links the data collections of c to the individual data elements of b will obliterate the boundaries of the semantic identification as defined for d , resulting in the *sort* $a \wedge b \wedge c$. In order to maintain a reference to this semantic identification, a new, named *sort* $c'd : a \wedge b \wedge c$ may be constructed to replace $(d : a \wedge b) \wedge c$ as the result of this definition. Note that the apostrophe should not be mistaken as an operator, but instead serves as a binding element in the construction of a name out of two component names. Consider the *sorts* of *labeledpoints* : *points* \wedge *labels* and *colors*. Then, the construction *labeledpoints* \wedge *colors* is redefined as *colors'labeledpoints* : *points* \wedge *labels* \wedge *colors*, effectively assigning colors to labels that are assigned to points, while maintaining a reference to the original construction of the *sort* through its name. The following rules can be specified to apply automatically in the definition of a *sort*:

$$\begin{aligned} (d : a \wedge b) \wedge c &\rightarrow c'd : a \wedge b \wedge c \\ (d : a + b) \wedge c &\rightarrow c'd : a \wedge b + a \wedge c \end{aligned} \quad (11)$$

Thus, although a semi-canonical form can be arrived at to support the comparison of *sorts* with respect to scope and coverage, representational structures corresponding to *sorts* may contain many more levels of operations over attribute and sum that reflect on the historical definition of this *sort* as a hierarchical composition of previously defined *sorts*.

Operations of product and difference can also be defined on *sorts* as representational structures. However, their semantic significance to representational structures is doubtful. Consider the following distributive

rules involving either the operation of product or difference with the operation of sum or cartesian product (Stouffs and Krishnamurti 1998):

$$\begin{aligned} (a + b) \cdot c &\rightarrow a \cdot c + b \cdot c \\ a \cdot (b + c) &\rightarrow a \cdot b + a \cdot c \\ (a + b) - c &\rightarrow (a - c) + (b - c) \\ a - (b + c) &\rightarrow (a - b) \cdot (a - c) \end{aligned} \tag{12}$$

$$\begin{aligned} (a \wedge b) \cdot (c \wedge d) &\rightarrow (a \cdot c) \wedge (b \cdot d) \text{ if } a \cdot c \neq 0 \text{ and } b \cdot d \neq 0; \\ (a \wedge b) \cdot c &\rightarrow 0 \text{ and} \\ a \cdot (b \wedge c) &\rightarrow 0, \text{ otherwise} \end{aligned} \tag{13}$$

$$\begin{aligned} (a \wedge b) - (c \wedge d) &\rightarrow (a - c) \wedge b + a \wedge (b - d) \text{ if } a - c \neq 0 \text{ and } b - d \neq 0; \\ (a \wedge b) - c &\rightarrow a \wedge b \text{ and} \\ c - (a \wedge b) &\rightarrow c, \text{ otherwise} \end{aligned} \tag{14}$$

Upon applying these distributive rules to any *sortal* composition involving any of these four operators, the only operations of product or difference that will result will be between primitive *sorts* that differ syntactically only in their arguments shaping the constraints that govern the resulting *sort*. According to rules (4) and (5), as specified for *sorts* conceptually, the resulting *sort* can be interpreted in terms of its overall constraints as a composition of the operand constraints (or instance equations) and defined as such. However, in this case, all references to the original construction of the *sort* will be lost, while the same effect can be achieved simply by altering the respective *sorts*' constraints. If instead, the original construction is maintained, a *sort* results that no longer offers a one-to-one relationship between its definition as an operational expression on primitive *sorts* and its representational structure. After all, the subtraction of a *sort* constrains the data elements that can be represented but the subtraction operation itself cannot be reflected in the representational structure. While we are interested in the definition of a language of expression of representational structures, ultimately, we are concerned with the representational structures themselves and the data that can be, and is, represented in these.

3.2 COMPARING REPRESENTATIONAL STRUCTURES

Sorts can be compared and matched as, roughly, equivalent, similar and convertible. Firstly, two *sorts* are *equivalent* if both are semantically derived from the same *sort*, where a semantic derivation consists of zero, one, or more renaming operations, $b : a$. Under equivalency, one *sort* may be

semantically derived from the other, or both may be semantically derived from a third *sort*. Thus, the equivalency relationship is reflexive, commutative, and transitive. Obviously, two *sorts* are *identical* only if these are also semantically identical. Secondly, two *sorts* are denoted *similar* if these are similarly constructed from the same primitive components, that is, the respective primitive components are syntactically identical and their respective constructions can be reduced to the same semi-canonical form using the associative and distributive rules specified above. Two primitive *sorts* are said to be syntactically identical if these have been identically defined, except for their name. A further distinction may be made between *strongly similar sorts*, which are constructed over equivalent *sorts*, and *weakly similar sorts*, which also contain (derivations from) primitive *sorts* that are only syntactically identical. For example, $a \wedge b$ and $a \wedge c$ are strongly similar if and only if b and c are equivalent. Also, $a + (d : b + c)$ and $(e : a + b) + c$ are strongly similar. The similarity relationship is also reflexive, commutative, and transitive.

Equivalent as well as similar *sorts* guarantee a correct exchange of data without data loss, except semantically. Furthermore, if one *sort* matches a part of another *sort*, under a composition over sum, data exchange without data loss applies from the first *sort* to the second. In the opposite direction, the occurrence of data loss is fully dependent on the actual data that is being exchanged. If two *sorts* are not similar, these may still be *convertible*, possibly without data loss. This is the case if two primitive *sorts* differ only in their arguments or constraints, similar to the instance equations of the mathematical approach. Data loss is then dependent on the relationships between these constraints and, possibly, on the actual data. This is also the case if, given a *sort*, another *sort* is constructed from this *sort* by reversing one or more attribute relationships, for instance, $a \wedge b$ and $b \wedge a$ are said to be convertible. (Whether data-loss occurs in such conversions is dependent on the behavioral categories of the constituent *sorts*; see section 4.2 "Composite Behaviors.") Naturally, *sorts* that are similarly constructed over convertible *sorts* are also convertible.

Given a partial match under the attribute relationship, *sorts* are still partially convertible. Again, data loss will be dependent on the actual match and on the direction of the data exchange. For example, two *sorts* a and $a \wedge b$ are partially convertible. Data loss occurs when exchanging data from $a \wedge b$ to a : all attribute information represented in b will be discarded. When exchanging data from a to $a \wedge b$, a default attribute must be assigned to each data element from a in order to ensure the validity of the resulting information. Finally, two *sorts* are incongruous if no match, even partial, applies.

While two *sorts* may be uniquely classified as equivalent, convertible, or otherwise, various alternative matches may still exist. Consider the matching of two compositions of *sorts* under the operation of sum. Each pairing of component *sorts*, one from each composition, may be a potential match. If one composition contains two equivalent *sorts*, neither of which has an identical counterpart in the other composition, though at least one equivalent match does apply, then, multiple combinations of *sorts* from either composition will be evenly matched. A decision from the user will be necessary to resolve this situation. This decision may be stored for later retrieval, such that subsequent occurrences of the same or similar match no longer require the user's interaction. In another case, a single best match may exist and be automatically selected for a component *sort*, even if the intention of the user may be otherwise. As an example, a primitive *sort* may have an equivalent as well as a convertible counterpart in the other composition, with the user's preference for the convertible match. In this case, a lesser match may be assigned a higher priority by the user such that this is considered first in the matching process.

In general, a quantification of the matching between *sorts* may be considered, with integral values defined by the identical, equivalent, strongly similar, weakly similar, convertible, partially convertible, or incongruent character of the match, increasingly in this order, and decimal values reflecting on the details of the match. For example, a decimal value for a strongly similar match may be derived from, amongst others, the number of equivalent versus identical component *sorts*. Then, the user may choose to override selective matches by assigning higher (or lower) matching values.

The comparison and matching of *sorts* for data exchange allows one to monitor data-integrity during the design process, at all times, for a large variety of data. Specifically, the coverage of *sorts* can be compared; data can always be moved from more-restrictive to less-restrictive *sorts* without data loss; and data loss can be measured when moving data in the opposite direction. Active control over which conversions should and should not be allowed or considered may be presented to the user in the form of a level tuner: three user-defined levels specify matching value intervals of predefined handling behavior Table 1.

4. Behavioral Specifications for *Sorts*

Most CAD applications adopt an object-oriented approach at the conceptual level, providing users with line segments, surfaces, or solids as objects with

attributes that maintain their properties at all times, unless explicitly altered by the user. While conceptually attractive and very understandable to the user, this approach is inimical to creative design. Creative design activities rely on a restructuring of information uncaptured in the current information structure, as when looking at a design provides new insights that lead to a new interpretation of the design elements. It can be proven that continuity of computational change requires an anticipation of the structures that are to be changed (Krishnamurti and Stouffs 1997). Creativity, on the other hand, is devoid of anticipation.

TABLE I. Level tuner

level	handling
$< l_1$	<i>sorts</i> are considered equal, conversion is performed without notification
$< l_2$	user is notified of conversion
$< l_3$	user's approval is requested for conversion
$\geq l_3$	conversion is not allowed, unless upon user's specific initiation

Consider for example the composition of squares in figure 1. Specified as two square objects, each square can almost effortlessly be resized and moved. Instead, a manipulation of the individual line segments would require each square to be re-represented as a collection of four line segments. Visually, the composition in Figure 1 contains not two but three squares. Irrespective of whether the composition has been defined as a collection of two square objects or as a collection of twice four line segments, neither representation allows the third square easily to be distinguished and manipulated, unless it is additionally defined in the composition, possibly by drawing the shape over. Instead, if line segments would constitute dynamic data entities that can split themselves into any number of smaller line segments, the resulting composition of line segments would not only represent each of the three squares, but also an infinite number of other collections of line segments. Furthermore, representationally, each of these configurations has the same significance, and the designer may select any one as an interpretation of the design.

When dispensing with the object-oriented approach at a representational level, operations that may otherwise seem trivial, such as adding or removing elements or figures, become resolutely non-trivial. Consider the

addition of two numbers, in the case these represent cardinal values, e.g., a number of columns that is increased, and ordinal values, e.g., for a given space, determining the minimum distance to a fire exit or the (maximum) amount of ventilation required given a variety of activities. Similarly, consider additive versus subtractive colors, depending on whether these refer to the mixing of surface paints or light colors, respectively. A specification of such operational behavior needs to be included in the definition of a data type or structure. Fortunately, this behavioral specification may be reasonably limited to a few basic operations. Specifically, we consider the common arithmetic operations of addition and subtraction, and of product or intersection. The most common CAD operations of creation and deletion, and selection and deselection, can all be expressed as a combination of addition and subtraction operations from one *sort* or design space to another. More complex operations of grouping and layering can be similarly defined over more complex *sorts* (Stouffs and Krishnamurti 1996).

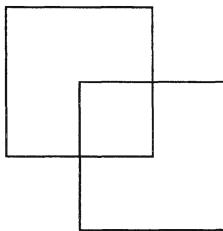


Figure 1. A simple, yet, ambiguous composition of (two or three) squares

A behavioral specification also is a prerequisite for an effective exchange of data between various representations and for a uniform handling of different and a priori unknown data structures. Consider the association of building performance data to design geometries. The behavior of these data as a result of alterations to the geometries can be expressed through a number of operations chosen to match the expected behavior. When an application receives the data together with its behavioral specification, the application can correctly interpret, manipulate, and represent this information without unexpected data loss.

For maximum flexibility, when composing *sorts* from other *sorts*, the operational behavior of the resulting *sort* should be automatically derived from the behavior of the component *sorts*. Hereto, we consider an algebraic framework for the specification of a *sorts'* operational behavior based on a partial order relationship (Stouffs 1994; Stiny 1991). This partial order relationship determines when an element can be considered a *part* of

another element. Fundamentally, any *part* of a data element defines in itself a valid data element of the same data type, and any combination of data elements under addition, subtraction, and product also constitutes a valid data element. Algebraically written, any collection of data elements of the same type is a member of an algebra that is ordered by a part relation and closed under the algebraic operations of addition, subtraction, and product.

4.1 TYPES OF BEHAVIORS

The simplest behavior that fits the requirements is a discrete behavior, corresponding to a mathematical set, where the part relation reduces to the subset relation, and the operations of addition, subtraction, and product correspond to set union, difference, and intersection, respectively. In other words, if a and b denote two data collections of a *sort* with discrete behavior, and A and B denote the corresponding sets of data elements ($a : A$ specifies A as a representation of a), then

$$\begin{aligned} a : A \wedge b : B \Rightarrow a \leq b &\Leftrightarrow A \subseteq B \\ a + b : A \cup B \\ a - b : A / B \\ a \cdot b : A \cap B \end{aligned} \tag{15}$$

Under the discrete behavior, an explicit action is still required from the user in order to alter any data element. Only if two elements are identical do these combine into one. Stiny (1992) explores the application of the algebraic model to geometries with weights as attributes. Weights may be considered to denote thickness for points and lines, or tones for surfaces and volumes. A behavior for weights becomes apparent from drawings: a single line drawn multiple times, every time with different thickness, appears as it was drawn once with the largest thickness, even though it assumes the same line with other thickness. Thus, a collection of weights always combines into a single weight, which has as value the least upper bound of all the individual weight values, i.e., their maximum value. This behavior is termed *ordinal*; using numbers to represent weights, the part relation on weights corresponds to the less-than-or-equal relation on numbers;

$$\begin{aligned} a : \{x\} \wedge b : \{y\} \Rightarrow a \leq b &\Leftrightarrow x \leq y \\ a + b : \{\max(x, y)\} \\ a - b : \{\} \text{ if } x \leq y, \text{ else } \{x\} \\ a \cdot b : \{\min(x, y)\} \end{aligned} \tag{16}$$

Line segments may be considered as intervals on infinite line carriers. An *interval* behavior applies to line segments as well as intervals of time or

other one-dimensional quantities: intervals on the same carrier that are adjacent or intersect combine into a single interval. An interval is a part of another interval if it is embedded in this interval. A specification of the interval behavior can be expressed in terms of the behavior of the interval boundaries. Let $B[a]$ denote the boundary of a collection a of intervals and, given two collections a and b , let I_a denote the collection of boundaries of a that lie within b , O_a denote the collection of boundaries of a that lie outside of b , M the collection of boundaries of both a and b where the respective intervals lie on the same side of the boundary, and N the collection of boundaries of both a and b where the respective intervals lie on opposite sides of the boundary (Stouffs 1994), Figure 2. Then,

$$\begin{aligned} a : B[a] \wedge b : B[b] \Rightarrow a \leq b &\Leftrightarrow I_a = 0 \wedge O_b = 0 \wedge N = 0 \\ a + b : B[a + b] &= O_a + O_b + M \\ a - b : B[a - b] &= O_a + I_b + N \\ a \cdot b : B[a \cdot b] &= I_a + I_b + M \end{aligned} \quad (17)$$

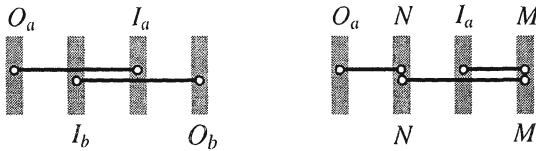


Figure 2. The specification of the boundary collections I_a , O_a , I_b , O_b , M and N , given two collections of intervals a (above) and b (below).

Note that the interval behavior also applies in the case of infinite intervals, provided an appropriate representation of both (infinite) ends of a carrier exists. Similar behaviors can be specified for plane segments and volumes (Stouffs 1994) as well as hypersegments of higher dimension; (17) still applies though the construction of I_a , O_a , I_b , O_b , M , and N is correspondingly more complex, Figure 3.

4.2 COMPOSITE BEHAVIORS

A composite *sort* inherits its behavior from its component *sorts* in a manner that depends on the compositional relationship. Under the operation of sum, the behavior is that of the component *sort* for each component. Data collections from different component *sorts* never interact, the resulting data collection, corresponding the composite *sort*, is the group of collections from all component *sorts*. When an operation applies to two data collections

of the same composite *sort*, the operation instead applies to the respective component collections.

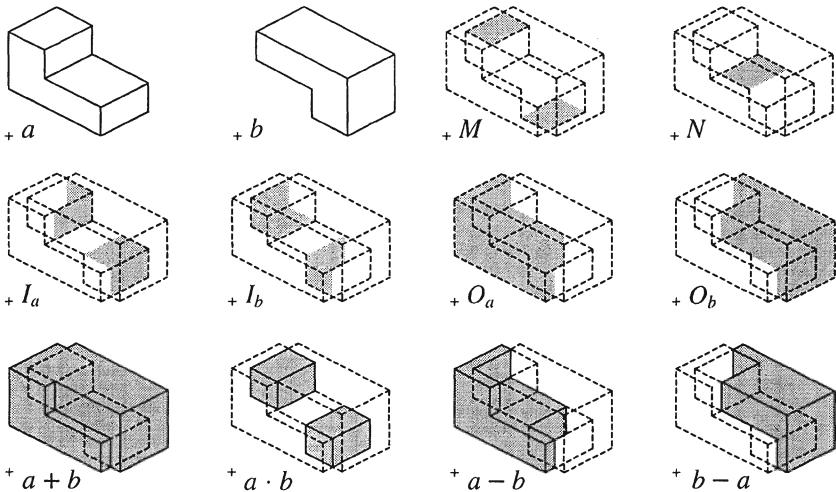


Figure 3. The boundary collections I_a , O_a , I_b , O_b , M and N for two volumes a and b , and the collections of volumes resulting from the operations $a + b$, $a \cdot b$, $a - b$ and $b - a$.

The attribute operation on *sorts* specifies a dependency relation on the *sorts* in a composition, where each component, except the first, defines an attribute *sort* to the previous component. That is, a corresponding data collection consists of data elements of the first component *sort*, each element of which has, as attribute, another data collection corresponding to the *sort* as a composition of all but the first component, in a recursive manner. Thus, the behavior of such a *sort* is defined by the behavior of its first component *sort*. Specifically, when an operation applies to two data collections of the same composite *sort* (under the attribute relationship), identical data elements merge and their attribute collections combine under the same operation. Any elements that have an empty attribute collection are removed. An object-oriented behavior can be achieved for any *sort* by combining this with a *sort* of (unique) identifiers under the attribute relationship. The resulting data form is akin to a database of individuals, where each individual has a unique key assigned.

Behaviors play an important role when assessing data-loss in information exchange between different *sorts*. Reorganizing component *sorts* under the attribute relationship into a different compositional hierarchy may alter the corresponding behavior and trigger data-loss. Consider a *sort* of weighted

points, i.e., a *sort* of points with attribute weights, and a *sort* of points of weights, i.e., a *sort* of weights with attribute points. A collection of weighted points defines a set of non-coincident points, each of which has a single weight assigned. These weights may be different for different points. The collection's behavior is discrete. Instead, a collection of points of weights is defined as a single weight with an attribute collection of points, and has an ordinal behavior. In both cases points are associated with weights. However, in the first case different points may be associated with different weights, whereas, in the second case all points are associated with the same weight. In a conversion from the first to the second *sort*, data-loss is inevitable.

5. Example

Consider design information in the form of design constraints and related information, e.g., for a steel-framed building project (Lottaz, Stouffs and Smith 2000), Figure 4. The information consists, minimally, of a set of authors, a set of constraints for each author, a common set of variables with each variable linked to the constraints defined over this variable, and a constraint solver (in the form of a URL) for each author (or constraint).

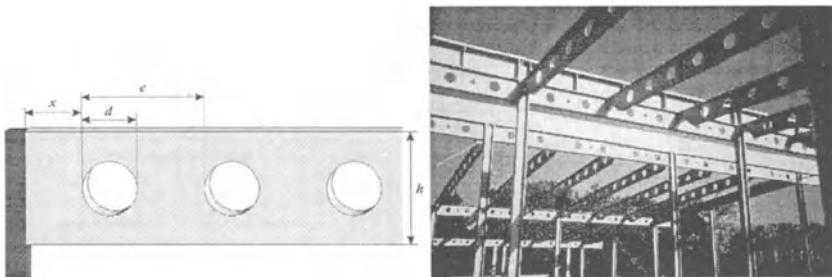


Figure 4. Design problem from a building project: the dimensioning of holes in steel beams

The author names, the constraint expressions and the variable names all define primitive *sorts* with characteristic individual 'Label'. The constraint solvers define a primitive *sort* with characteristic individual 'Url'. Three more primitive *sorts* with characteristic individual 'Property' serve to represent the various links between the constraint expressions and, respectively, the author names, variable names, and constraint solvers. Figure 5 presents a definition of these *sorts*, including a graphical depiction.

```

authornames : [Label]
constraintexpressions : [Label]
variablenames : [Label]
constraintsolvers : [Url]
(isdefinedby, hasconstraints) : [Property] (constraintexpressions, authornames)
(contains, belongsto) : [Property] (constraintexpressions, variablenames)
(issolvedby, solves) : [Property] (constraintexpressions, constraintsolvers)

```

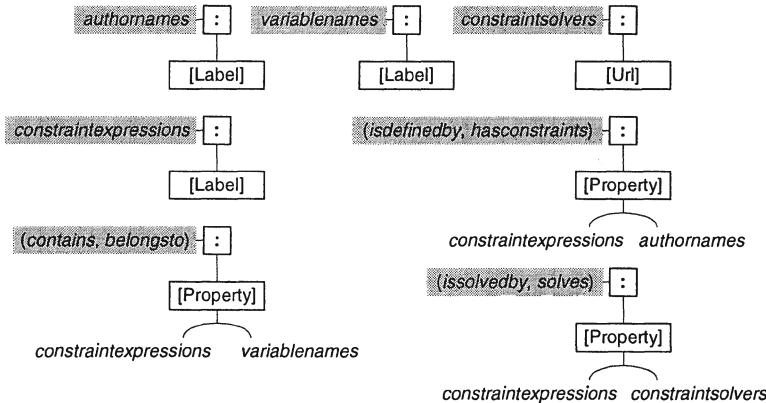


Figure 5. Definition and graphical depiction of primitive *sorts* for the constraints example

An organization of the design information by type, i.e., constraints, variables, authors, and solvers, with entities linked as appropriate, presents a straightforward and efficient way of storing this information into a relational database, Figure 6.

In order to support an actual design session, the author's design itself, i.e., his or her design constraints, should form the focus of the information organization. All other information entities can be made accessible from these, thereby clarifying each constraint's context and role in the design. A corresponding representational schema is presented in Figure 7. The author's constraints each specify the variables affected and provide access to the author's constraint solver. Each variable, in turn, specifies the constraints from other authors that are defined over this variable, and each of these constraints specifies its author. All information links are additionally provided. This representational schema supports the user in evaluating the effect of altering a constraint on the design and whether such a change may interfere with other constraints specified by the collaboration partners.

Figure 8 offers a VRML visualization of design data from the steel-framed building project represented in this schema.

```

database : authors + constraints + variables + solvers
authors : authornames ^ hasconstraints
constraints : constraintexpressions ^ constraintproperties
constraintproperties : isdefinedby + contains + issolvedby + attributes
variables : variablenames ^ variableproperties
variableproperties : belongsto + attributes
solvers : constraintsolvers + solverproperties
solverproperties : solves + attributes

```

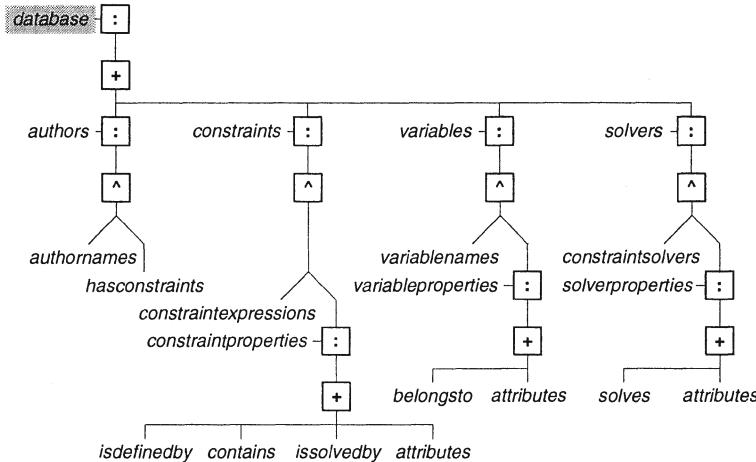


Figure 6. Database view of the design constraints example: definition and graphical depiction of the database sort. Note that a definition of the attributes sort is assumed, but not provided.

6. Discussion

In the definition and development of *sorts* as a concept and framework for representational flexibility, we let ourselves be guided as much by the algebraic strength of the adopted approach as by its practical benefits. This algebraic character offers a handle for the comparison and matching of representational structures and for the exchange of data accordingly. It also provides the ability to alter representational structures on the fly while maintaining their uniform approach of dealing with and manipulating data entities. The practical benefits must show in the description and building of representational structures as compositions of basic building blocks, that serve the design process in all its aspects. Ideally, a designer should be able to build or alter a representational structure to reflect on a particular need, in an almost intuitive way. Hereto, the behavioral specification serves as an attempt to bridge the desired simplicity of expressing representational structures and the necessary power of these structures to support design

activities effectively. In this paper, we have explored and argued the algebraic strength on both a conceptual and representational level and offered a simple example in the context of design constraints. In order to further illustrate the approach's practical benefits, other, more complex, examples of practical design cases are currently being investigated.

```

myconstraints : constraintexpressions ^ myconstraintproperties
myconstraintproperties : myvariables + solvers + issolvedby + attributes
myvariables : variablenames ^ myvariableproperties
myvariableproperties : otherconstraints + belongsto + attributes
otherconstraints : constraintexpressions ^ otherconstraintproperties
otherconstraintproperties : authors + contains + issolvedby + isdefinedby + attributes

```

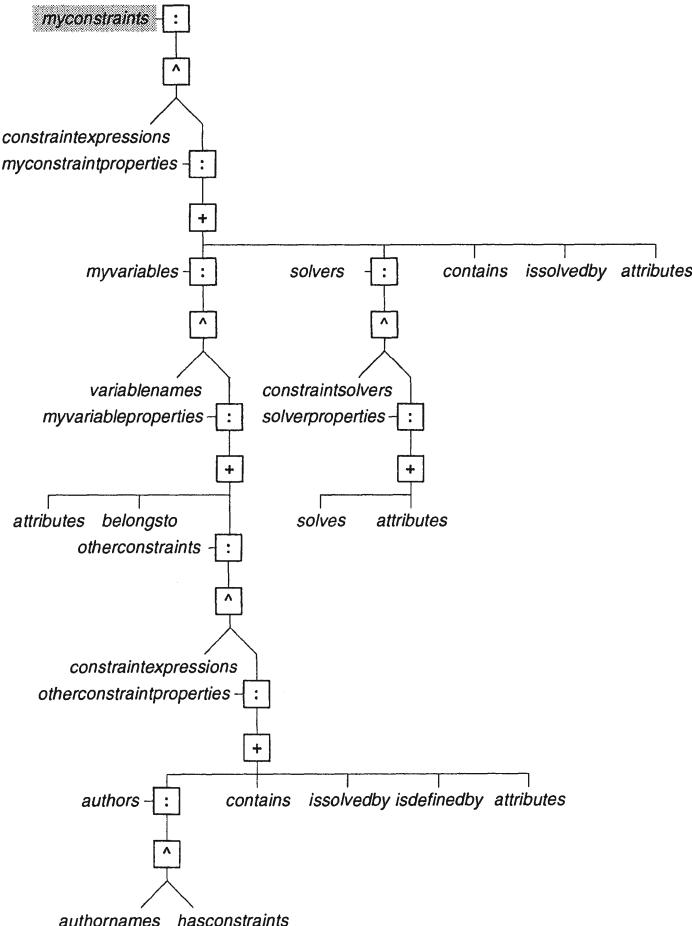


Figure 7. Design view of the design constraints example: definition and graphical depiction of the myconstraints sort

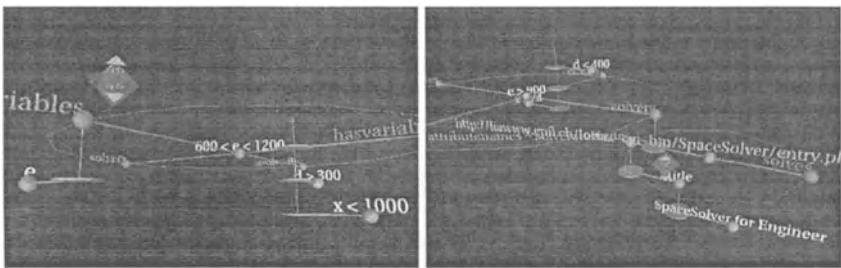


Figure 8. Snapshots of a VRML visualization for the steel-framed building project

While many other approaches exist and are being explored to support flexibility and extensibility of design models and representations, as well as the exchange of information between various representations or applications, few attempt to achieve this through a simple expressive language. Both van Leeuwen (1999) and Snyder (1998) adopt a schema approach. Feature-based modeling (van Leeuwen 1999) allows for the extensibility of conceptual schemas and supports these to be shared between different applications or designers. It also allows the designer to apply schemas to particular design situations by modifying or extending these with relationships and properties. Additionally, feature type recognition (van Leeuwen and de Vries 2000) enables the mapping of user-defined schemas to approved schema libraries, further supporting communication. In order to access and develop these schemas while modeling the design information, a 3D graphical tool is being developed that offers access to both the design model and the schema library (Coomans 2001). The SPROUT modeling language (Snyder and Flemming 1999; Snyder 1998) allows for the specification of schematic descriptions that can be used to generate computer programs that provably map data between different applications.

The LexiCon semantic model defines a formal vocabulary for the storage and exchange of information in the construction industry (Woestenenk 1998). It is defined in a semi-syntactic approach in which concepts are unambiguously defined by their constituent attributes. In this way, these attributes comprise the primitive concepts defining the semantic vocabulary of the model. If this descriptive approach is taken one step further, the attributes themselves can be described syntactically, leading to a purely syntactic description of concepts as compositions of primitive data types. Similar to *sorts*, a formal descriptive framework may then allow these syntactical descriptions to be compared independently of their conceptual meanings, allowing for synonym concepts, and for various degrees of similarity between alternative concepts.

XML defines such a formal framework. XML is a meta-language that serves to define markup languages for specific purposes. By specifying a grammatical structure of markup tags and their composition, a markup language is defined that can be shared with others. When an agreement can be reached on the tags, various markup languages can be developed based on these tags, and information exchanged between these, even if these languages differ in scope or composition. XML has the advantage that it is readable by both humans and the computer; markup languages based on XML can easily be adapted or extended to one's own specific purposes or needs. XML is particularly suited to structure otherwise unstructured information, such as textual data, and to organize information available over the Web. However, an XML based markup language does not provide any information on how to manipulate its data and, as such, is ill suited to represent detailed graphical or geometrical data.

From XML, our approach borrows a foundation consisting of an extensible vocabulary of data components that can be composed hierarchically into a representational language. In order to integrate the behavioral specification in the implementation, an object-oriented approach is applied, defining data elements as objects that encapsulate both the data structure and the operations defined on these structures. Also, the compositional operators are selected such that all representational structures offer the same operational functionality and derive this behavior from their component structures.

Then, a language specification is derived on two levels, similar to the approach used by the International Alliance for Interoperability (IAI) in the specification of its Industry Foundation Classes (IFCs) for a building object model (Bazjanac 1998). On a first syntactic level, the vocabulary of primitive object classes and their respective behaviors is defined. This behavior, in itself, does not provide any meaning to the object class. In fact, a same data structure may define two or more object classes if as many different behaviors can be said to apply, for different purposes. On a second level, a selection of object classes is defined and, individually, named in order to express a semantic concept. These named classes can, subsequently, be composed into a hierarchical structure in order to define an appropriate representational schema. In contrast to the IFC approach, this semantic concept can be specified by the user and the representational structure composed accordingly. Alternative representations can be defined by altering the compositional structure or the selection of component classes. As each representation defines the same common operations, these can be reasonably plugged into an applicative interface for manipulation.

Acknowledgements

This work is partly funded by the Netherlands Organization for Scientific Research (NWO), grant nr. 016.007.007. The second author is partially supported by the National Science Foundation, grant nr. CMS-0121549. The first author would like to thank Michael Cumming for his invaluable comments.

References

- Bazjanac, V: 1998, Industry foundation classes: bringing software interoperability to the building industry, *The Construction Specifier* **6**: 47–54.
- Coomans, MKD: 2001, DDDiver: 3D interactive visualization of entity relationships, in D. Ebert (ed.), *Data Visualization 2001: Proceedings of the Joint Eurographics and IEEE TCVG Symposium on Visualization*, Springer, Vienna, pp. 291–299.
- Krishnamurti, R and Stouffs, R: 1997, Spatial change: continuity, reversibility and emergent shapes, *Environment and Planning B: Planning and Design* **24**: 359–384.
- Lottaz, C, Stouffs, R and Smith, I: 2000, Increasing understanding during collaboration through advanced representations, *Electronic Journal of Information Technology in Construction* **5**: 1–25. [itcon.org/2000/1/]
- Requicha, A.A.G.: 1980, Representations for rigid solids: theory, methods and systems, *Computing Surveys* **12**: 437–464.
- Snyder, J.D.: 1998, *Conceptual Modeling and Application Integration in CAD: The Essential Elements*, Ph.D. dissertation, School of Architecture, Carnegie Mellon University, Pittsburgh, PA.
- Snyder, J and Flemming, U: 1999, Information sharing in building design, in G. Augenbroe and C. Eastman (eds), *Computers in Building*, Kluwer Academic, Boston, pp. 165–183.
- Stiny, G: 1992, Weights, *Environment and Planning B: Planning and Design* **19**: 413–430.
- Stouffs, R: 1994, *The Algebra of Shapes*, Ph.D. dissertation, Department of Architecture, Carnegie Mellon University, Pittsburgh, PA.
- Stouffs, R and Krishnamurti, R: 1998, An algebraic approach to comparing representations, in J. Barallo (ed.), *Mathematics & Design 98*, The University of the Basque Country, San Sebastian, pp. 105–114.
- Stouffs, R and Krishnamurti, R: 1996, The extensibility and applicability of geometric representations, *Architecture Proceedings of 3rd Design and Decision Support Systems in Architecture and Urban Planning Conference*, Eindhoven University of Technology, Eindhoven, pp. 436–452.
- Stouffs, R, Krishnamurti, R and Eastman, CM: 1996, A formal structure for nonequivalent solid representations, in S Finger, M Mäntylä and T Tomiyama (eds), *Proceedings of IFIP WG 5.2 Workshop on Knowledge Intensive CAD II*, International Federation for Information Processing, Working Group 5.2, pp. 269–289.
- van Leeuwen, JP: 1999, *Modeling Architectural Design Information by Features*, Ph.D. Dissertation, Eindhoven University of Technology, Eindhoven.
- van Leeuwen, JP and de Vries, B: 2000, Modelling with features and the formalisation of early design knowledge, in R Gonçalves, A Steiger-Garçao and R Scherer (eds), *Product and Process Modeling in Building and Construction*, A.A. Balkema, Rotterdam, pp. 167–176.

Woestenenk, K: 1998, A common construction vocabulary, in R Amor (ed.), *Product and Process Modelling in the Building Industry*, Building Research Establishment, Watford, pp. 561–568.

ANALYSIS OF ARCHITECTURAL SPACE COMPOSITION USING INDUCTIVE LOGIC PROGRAMMING

NORITOSHI SUGIURA AND SHIGEYUKI OKAZAKI
Kyoto University
Japan

Abstract. In this paper, we summarized an application of the ILP technique to the analysis of the human design process in architectural design. We focused particularly on discovering the patterns in the spatial composition process in the framework of design analysis called Architectural Space Montage Technique. Spatial composition data used in this paper are complexly structured data, which are collections of architectural objects with geometric relationships. The objects and relationships are constrained within a given set of attributes (such as object type, object angle, time stamp, relation type) together with background knowledge on the class hierarchies of object types and relation types. We modeled these spatial composition process data in first-order logic and applied the ILP system Progol to discover design patterns that characterized the given data.

1. Introduction

In this study, the process of architectural design was analyzed by Inductive Logic Programming (ILP) (Muggleton 1994; Lavrac and Dzeroski et al. 1999). ILP is a machine learning technique based on first-order logic that executes Inductive Reasoning, which is the generalizing of results from examples to generate new concepts.

In various contexts, it has been reported in the domain of phenomenology or developmental psychology that humans have unconscious spatial schemata that enables them to recognize space (Merleau-Ponty 1945; Piaget 1963). In support of this theory, it has been proposed that the human design process is affected by the schemata, and these appear as compositional patterns of such architectural elements as walls, furniture, buildings and so on (Schulz 1973; Bollnow 1963). In order to create an architectural space suitable for human recognition, it is important to find the latent patterns of

spatial composition affected by the spatial schemata from a psychological point of view. This is currently a key issue in the architectural field.

From the above context, this study investigates the patterns in the initial process of architectural design, which is the process of visualizing one's own mental images. We focused particularly on discovering patterns that indicate the differences between groups of architecturally trained and untrained individuals.

Several studies on architectural design patterns have been done. For example, the Shape-Grammar for F. L. Wright, who was a famous architect, was defined. It was a set of production rules, which could generate floor plans in Wright's architectural style (Koning et al. 1981). The Shape-Grammar, however, does not reflect the actual design process. In this paper, actual design processes using Architectural Space Montage Technique (ASMT) were analyzed.

The ASMT was developed by one of the authors to elucidate the fundamental patterns of spatial composition that exist in human beings. In an experiment using ASMT, a subject composes architectural space by placing various miniatures such as walls, furniture, and so on, at a scale of 1:50 on a white board. Figure 1 illustrates two examples of architectural models developed using this method. In this study, we regard a spatial composition process using this method as an initial process of architectural design.

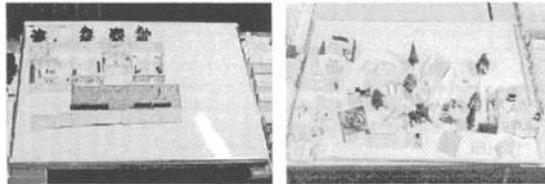


Figure 1. Examples of models constructed using ASMT by an undergraduate student (left) and a kindergarten child (right)

In single ASMT experiment, it is possible to have dozens to hundreds of miniatures placed. Moreover, a miniature newly placed has plural relationships to the previously placed miniatures. It is difficult to discover patterns by only relying on human inspection in complexly structured data such as those in the spatial composition process. Therefore, in this study, we applied Progol, one of the ILP systems, to find latent patterns of the spatial composition process in ASMT.

Recently, machine learning systems (see C4.5 (Quinlan 1993)) that generate decision trees based on propositional logic have been used as data mining engines in practical domains such as the medical field (Tsumoto and

Tanaka 1997) and the distribution industry (Numao and Shimizu 1997). However, in order to learn the chain of relationships between miniatures, ILP, based on expressive first-order logic, is more appropriate for ASMT than systems of learning based on propositional logic.

We modeled the spatial composition process with the Entity-Relationship (ER) model and designed a module to convert the spatial composition process data recreated with a CAD system to Prolog clauses, which constitute input data to Progol. The integration of the CAD system with Progol is applied to discover patterns of the spatial composition process.

Brief overviews of ASMT and Progol are given in Sections 2 and 3, respectively. In Section 4, the spatial composition process is modeled in first-order logic. We show the procedure for analysis in Section 5. In Section 6, we show an example of discovering characteristic patterns of architecturally trained subjects and untrained subjects from their spatial composition process data. The results of ASMT experiments and pattern discovery are shown. Our conclusions and future work are stated in Section 7.

2. ASMT

ASMT was originally developed in the context of psychotherapy. Clinical psychological analysis has been undertaken on the characteristic patterns of the spatial compositions formed by schizophrenic patients, school children, mentally handicapped children, and kindergarten children (Okazaki et al. 1992a, 1992b, 1997, 1999, respectively). In ASMT, architectural space is composed by simply placing three-dimensional miniatures. Therefore, the subject is not limited by his or her drawing ability. Various subjects can readily express a 3-D architectural space. Moreover, we can observe the steps in the design process clearly.

The types of miniatures used in ASMT differ slightly depending on the experimental groups. In this study, we prepared the following 44 kinds of miniatures: six kinds of walls made of styrene board of different lengths (1800 mm, 2700 mm, 3600 mm, and 5400 mm) and in various colors (blue, red, yellow, green, white, gray, pink, ivory, cream, mint, and grain) with various openings, mirror walls and glass walls in lengths of 3600 mm and 5400 mm with the glass walls in various colors (blue, orange, and clear), columns, twelve kinds of furniture (e.g. table, sofa, carpet, shelf), six different sanitary fixtures (e.g. sink, toilet, bathtub), four different figures (i.e. man, woman, boy and girl), two types of animals (i.e. dog and cat), six types of vegetation (e.g. lawn, conifer, broadleaf, hedge), and five different



Figure 2. Miniatures used in ASMT experiment (from left: walls, furniture, sanitary fixtures, figures, vegetations, and architectural elements)

architectural elements (e.g. balcony, stairs). Figure 2 shows examples of these miniatures.

The experimental setting and procedure are stated as follows. A white board (60 cm by 90 cm) was placed horizontally on a desk in the experimental room. Two smaller white boards (45 cm by 30 cm) were placed on both sides of the larger board, with miniature walls arranged on top of the boards. The other miniatures were displayed on a shelf.

Each subject constructed the model of his ideal house on the large white board. The experiment ends whenever the subject tells the experimenter that he is finished with the model. The state of the model in the experiment was constantly recorded by a video camera.

3. Progol

Progol is one of ILP systems by Mugletton (1995). Progol combines Inverse Entailment with general-to-specific search through a refinement graph. It allows arbitrary Prolog programs as the background knowledge and arbitrary definite clauses as examples. Input data to Progol consist of a set of positive example $E+$, a set of negative example $E-$, a set of background knowledge BK , and mode declarations used by Progol to guide the process of constructing a generalization from its example. From these data, hypotheses in the form of Horn clause are constructed.

Hypothesis H is complete if $\forall e \in E+ : BK \cup H \models e$, where " \models " means logical entailment. Hypothesis H is consistent if $\forall e \in E- : BK \cup H \not\models e$.

In this paper, one of the versions of Progol, P-Progol 2.7.5, was used. P-Progol was implemented by Srinivasan and Rui in Prolog based on the Progol algorithm (1999).

4. Modeling of the Spatial Composition Process

In order to describe the design process logically, we defined a unit of the spatial composition process as a miniature placement with relationships to the miniatures previously placed. The spatial composition process is a set of miniature placements.

4.1 DATA MODEL

In this study, we modeled the spatial composition process with the ER model, which is a well-known semantic data model. An ER model is built with three elements: entity, relationship, and attribute. Shimazu and Hurukawa expressed relational data of plane figures with the RER (Refined Entity-Relationship) model, which has two types of attributes called "primitive" and "derived" (2000). On the other hand, we refined the ER model in order to express not only a static relational data of figure elements but also the process of generating a figure, Figure 3. In our data model, the relationship has a direction. A placed miniature corresponds to an entity. After this, we universally called the miniature the "object." A geometric relationship occurs between newly placed objects and the existing objects. Each object has attributes such as an object type and an ordinal number of placement occurrence. Each relationship has attributes such as a relation type. In addition, IS-A hierarchies of the attributes based on the inclusion relation among concepts are known in advance. Details of attributes are described in the following section.

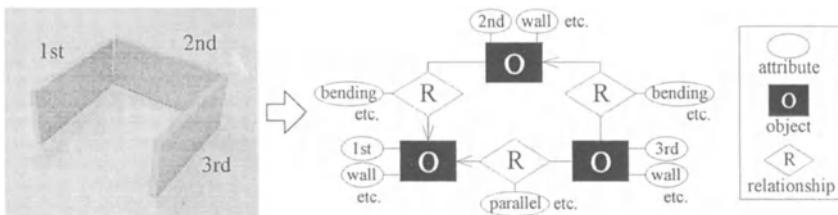


Figure 3. Example of representation of spatial composition process using ER data model

4.2 FIRST-ORDER REPRESENTATION

In this subsection, we show how to describe the above ER model in the form of clauses on first-order logic. The first-order description of a spatial composition process has a finite domain $D = (O \cup R)$, where O is a set of all objects placed and R is a set of all relationships occurred between objects.

4.2.1 Description of the object

Each object o has three attributes: ID number $ObjID$, type $ObjType$, and absolute angle $ObjAngle$, where $o \in O$. The ID number is in four digits, with the first digit representing the experimental case, and the remaining three representing the ordinal number of placement occurrence attached to the object. For example, $ObjID$ attached to the object placed in the third

placement of experimental case 1 is 1003. Two predicates, `type/2` and `angle/2`, were defined for each object o , and these represent the type of the object and the absolute angle of the object, respectively. The number described after "/" means arity.

The type of object is represented as follows:

```
type(ObjID, ObjType),
```

where $\text{ObjType} \in OT$. OT is a set of 44 vocabularies that represent the miniature types prepared for the ASMT experiment. The object types constitute a hierarchy based on the IS-A relationship. The hierarchies are shown in Figure 4.

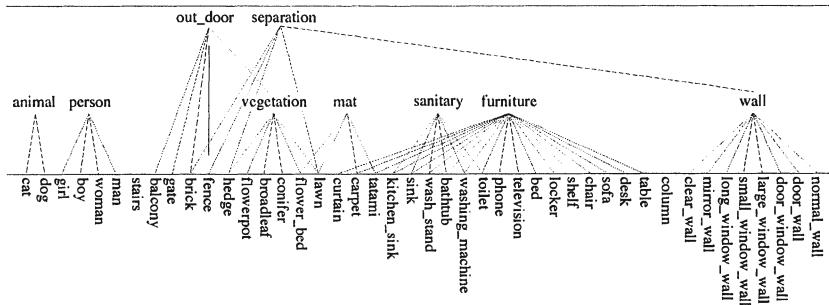


Figure 4. Object type hierarchy

The absolute angle is an angle between the miniature and the long side of the white board. The angles are measured in degrees. The absolute angle of the object is represented as follows:

```
angle(ObjID, ObjAngle),
```

where $\text{ObjAngle} \in \{0, 45, 90, \text{ambiguous}\}$. "Ambiguous" means angles other than the 0, 45, or 90 degrees.

4.2.2 Description of the relationship

Each relationship r between objects has three attributes: type of relation RelType , connecting point C_Point , and difference Diff between the ordinal numbers attached to the objects, where $r \in R$.

A predicate `relation/5` was defined on each relationship r . The relationship between objects is represented as follows:

```
relation(N-ObjID, RelType, C_Point, Diff, E-ObjID),
```

where N-ObjID and E-ObjID are the ID numbers of the newly placed object and the previously placed object related to the newly placed object, respectively. RelType and C_Point are defined as follows.

The relation type has the following seven sub-attributes.

- (1) Contact between objects: This attribute takes a value $Cont \in \{\text{attach}, \text{detach}, \text{isolate}\}$, where each element represents the condition that objects are in contact, objects are out of contact but close to each other, and objects are isolated from each other, respectively.
- (2) Relative angle between objects: This attribute takes a value $Ang \in \{0, 45, 90, \text{ambiguous}\}$.
- (3) Arrangement type of objects: This attribute is available in the case where $Cont$ is "attach" or "detach", and takes a value $Arr \in \{\text{on}, \text{parallel}, \text{T_type}, \text{bending}, \text{L_type}, \text{close}\}$, where "close" represents arrangements other than the formers.
- (4) Alignment of the edges of objects: This attribute is available where Arr is "parallel", and takes a value $Align \in \{\text{reg}, \text{semi_reg}, \text{irreg}\}$, where each element represents the edges of both sides of the objects in alignment, the edges of one side of the objects in alignment, and the edges of both sides of the objects out of alignment, respectively.
- (5) Distance between objects: This attribute is available where Arr is "parallel", and takes a value $Dist \in \{\text{attach}, \text{near}, \text{right}, \text{far}\}$, where "right" means the distance between the objects and the length of the newly placed object are the same.
- (6) Direction of bending: This attribute is available where Arr is "bending", and takes a value $Direc \in \{-r, -l\}$, where "-r" and "-l" represent right-handed and left-handed, respectively.
- (7) Combination of object shapes: The shapes of the objects are classified into three classes: line, point, and mat. The relationship depends on a combination of the classes. Table 1 shows the object types that correspond to these classes. The combination is represented in a row of two letters. Each letter corresponds to the first letter of the object shape. For example, the combination of line and point is described as "lp." This attribute takes a value $Comb \in \{\text{ll}, \text{lp}, \text{lm}, \text{pp}, \text{pm}, \text{mm}\}$.

TABLE 1. Shape classification of object types

Class of shape	Type of object corresponding to class of shape					
line	normal_wall	long_window_wall	desk	television	wash_stand	brick
	door_wall	mirror_wall	sofa	phone	sink	gate
	door_window_wall	clear_wall	rack	toilet	kitchen_sink	balcony
	large_window_wall	washing_machine	locker	curtain	hedge	stairs
	small_window_wall	table	bed	bathtub	fence	
point	column, conifer, broadleaf, flowerpot, man, woman, boy, girl, dog, cat, chair					
mat	tatami, carpet, lawn, flower_bed					

Theoretically, $RelType \subseteq (D_{cont} \times D_{ang} \times D_{arr} \times D_{align} \times D_{dist} \times D_{direc} \times D_{comb})$, where " D_{-} " means the domain of the sub-attribute of $RelType$ given above. In this paper, 65 combinations of the sub-attributes that

occurred frequently in the ASMT experiment were selected. Each combination was assigned a vocabulary which represent the relation type. The relation types constitute a IS-A hierarchy based on the inclusion relation. The hierarchy is shown in Figure 5.

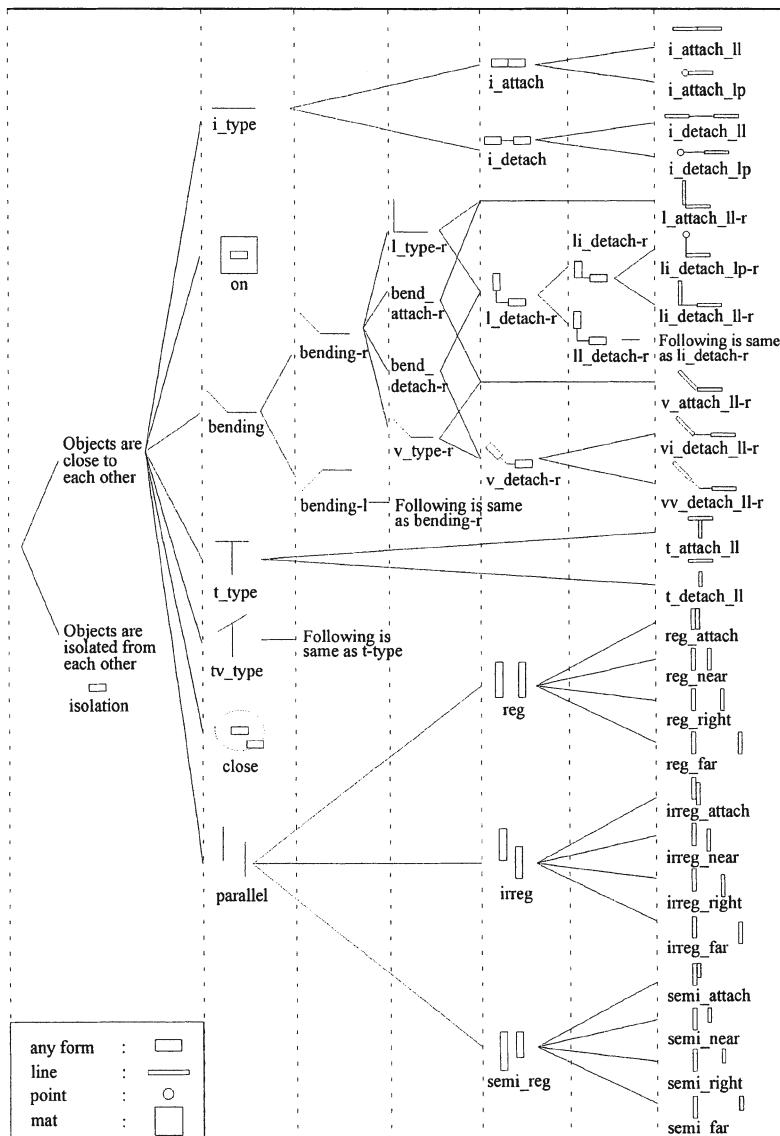


Figure 5. Hierarchy of relation types

The attribute of connecting point indicates where the newly placed object is connected to the previously existing object. This attribute takes a value $C_Point \in \{(bottom, top, left, right) \cup (center, corner, edge, ambiguous)\}$, where "bottom" and "top" denote the left and right ends of the previously placed object, respectively. "Left" and "right" mean left and right side of the existing object, viewing from bottom to top, respectively, Figure 6(a). The former four elements are available where Arr , which is an attribute value of the arrangement type, is "parallel," "T_type," "bending," or "I_type." The latter four elements are available where Arr is "on", Figure 6(b).

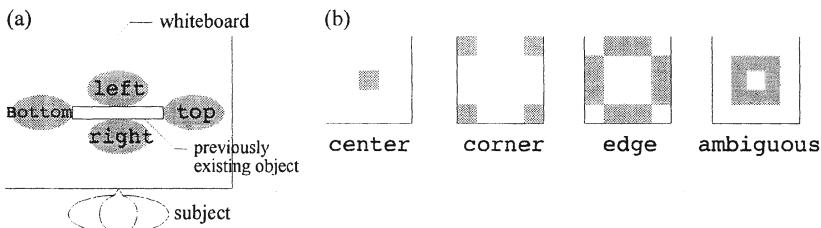


Figure 6. Spheres indicating the connecting points: (a) in the case of "parallel," "T_type," "bending," "I_type," and (b) in the case of "on"

5. Procedure of Analysis

The analysis procedure consists of the following three steps, Figure 7.

- Step 1: The ASMT experiments are conducted.
- Step 2: The actual model is recreated with a CAD system where the order of placements is consciously taken into consideration.
- Step 3: The CAD data are converted automatically into Prolog clauses readable by Progol. These data constitute the input data to Progol.

Steps 2 and 3 are repeated in every experimental case.

- Step 4: The descriptions of the positive examples, the negative examples, and the IS-A relationships of the object type and the relation type are added to descriptions generated in step 3. Every placement is set as the example. These data are inputted to Progol.
 - Step 5: Progol acquires rules from the input data.
 - Step 6: The rules found are interpreted by a person.
- Steps 2, 3, and 4 are detailed in the following subsections.

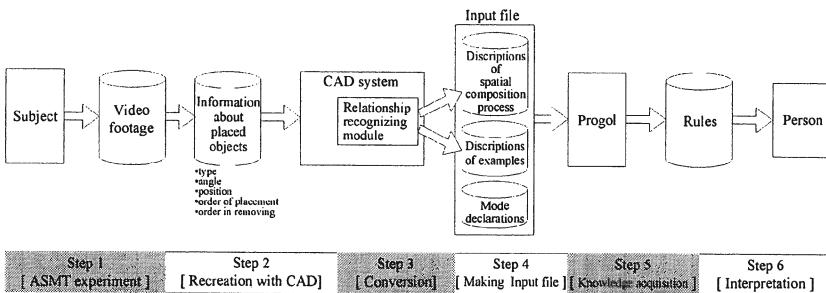


Figure 7. Analysis procedure of the spatial composition process

5.1 RECREATION OF THE MODEL

First, the following information about the placed object was extracted from the video footage of the ASMT experiment:

- type of object
- angle between the object and the long side of the white board
- position on the white board
- order of the object placement
- order in removing the object

Next, the actual model was recreated with one of the CAD systems *MiniCAD7* by placing and removing symbol objects according to the above information. Symbol objects are three-dimensional models of the miniatures prepared for the ASMT experiment. An example of a model recreated with the CAD system is shown in Figure 8.

An ordinal number *PlaceID* indicating the occurrence of the placement is assigned automatically to the object placed. Moving an object is regarded as removing it and placing a new one. To remove an object in the CAD system, the object is simply changed to be invisible by changing its color. An ordinal number *RemoveID* is assigned to it, which indicates the occurrence of the removal.

5.2 CONVERSION TO PROLOG CLAUSES

The Relationship-Recognizing (RR) module was designed to recognize the attributes of the objects and relationships among them, such as *ObjID*, *ObjType*, *ObjAng*, and *RelType*, and so on from the CAD data. These are automatically converted by the module to the first-order representation. The RR module was integrated with the CAD system described in subsection 5.1. The basic information used by the RR module includes (1) vertex

coordinates of the objects, (2) *PlaceID*, (3) *RemoveID*, and (4) ID number indicating the object type.

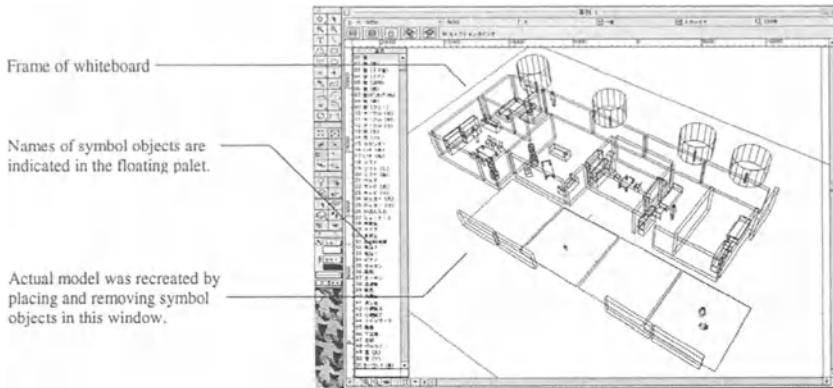


Figure 8. Example of model recreated with CAD system

Descriptions of the object types and the relationships in the form of a ground unit clause are outputted. The object type and the relation type recognized correspond to the leaves of the tree illustrated in Figures 4 and 5, respectively. In the case of object disposition shown in Figure 9, the descriptions generated by this module are illustrated in Figure 10(c).

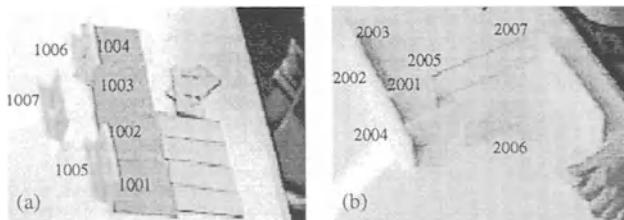


Figure 9. (a) Disposition in experimental case 1, (b) Disposition in experimental case 2 (Numbers in the figure indicate the object ID numbers *ObjID*)

5.3 INPUT FILE TO PROGOL

In this study, the input file to Progol was designed as follows:

- **Positive and Negative Examples.** The target concept in this study is the spatial composition process by a specific group P . All placements by subjects belonging to group P were set as positive examples, and all placements by subjects belonging to another group were set as negative examples.

(a) Positive examples	(c) First-order representation of every placement	(d) IS-A relationships
placement(1001). placement(1002). placement(1003). placement(1004). ⋮	type(1001,lawn). angle(1001,0). relation(1001,isolation,none,no_dif,no_obj). type(1002,lawn). angle(1002,0). relation(1002,close,none,1,1001). ⋮	type(A,wall)-type(A,normal_wall). type(A,wall)-type(A,door_wall). type(A,wall)-type(A,clear_wall). type(A,separation)-type(A,normal_wall). type(A,separation)-type(A,door_wall). type(A,separation)-type(A,clear_wall). ⋮
(b) Negative examples	type(2001,door_window_wall). angle(2001,90). relation(2001,isolation,none,no_dif,no_obj). type(2002,normal_wall). angle(2002,0). relation(2002,I_touch_ll-l,bottm,I,2001). ⋮	relation(A,i_attach,B,C,D):- relation(A,i_attach_ll,B,C,D). relation(A,i_attach,B,C,D):- relation(A,i_attach_lp,B,C,D). relation(A,i_detach,B,C,D):- relation(A,i_attach_ll,B,C,D). ⋮
(e) Mode declarations	:-mode(I,placement(+miniature_id)). :-mode(I,type(+obj_id)). :-mode(I,angle(+obj_id)). :-mode(I,relation(+obj_id,#rel_type,#c_point,#diff,-obj_id)). :-mode(I,relation(+obj_id,#rel_type,#c_point,#diff,-obj_id)). :-mode(I,relation(+obj_id,#rel_type,-c_point,#diff,-obj_id)). :-mode(I,relation(+obj_id,#rel_type,-c_point,#diff,-obj_id)).	:-determination(placement/I,type/2). :-determination(placement/I,angle/2). :-determination(placement/I,relation/5).
(f) Determinations		
(g) Induced rules	placement(A):-relation(A,irreg,B,C,D),relation(D,bend_attach-I,E,F,G),relation(G,irreg,H,I,J). placement(A):-relation(A,bend_attach-I,B,C,D),type(D,wall),relation(D,irreg,E,F,G),type(G,wall). ⋮	

Figure 10. Examples of input and output data in the case that targets are characteristic patterns in case 1: (a) Positive examples, (b) Negative examples, (c) First-order representation of each placement, (d) IS-A relationships of object type and relation type, (e) Mode declarations, (f) Determinations, (g) Induced rules.

- **Background Knowledge.** The background knowledge *BK* consists of two kinds of data. One is the knowledge of attributes of the objects and the relationships, and the other is knowledge of the IS-A relationships between the attribute values. The IS-A relationships between the object types (see Figure 4) and the relation types (see Figure 5) are set as *BK*. IS-A relationships are represented in the form of Horn clause. Examples of descriptions of IS-A relationships are shown in Figure 10(d).
 - **Mode Declarations and Determinations.** In P-Progol, the user indicates the language in which hypotheses are to be constructed by invoking mode declaration and determination. For example, mode declaration can be represented as follows:
- ```
: - mode (recall, p (+a, -b, #c)) ,
```
- where recall is the number of successful calls to the predicate and "+" indicates that the argument is an input. For all calls to this predicate, the argument will be bound to a value. The symbol "-" indicates that the

argument is an output. For all calls to this predicate, the argument will output a term. The symbol "#" indicates that a constant should appear in this argument.

Determination takes the following form:

determination(*TargetName*/*Arity*,*BackgroundName*/*Arity*) .,  
where the first argument is the name and arity of the target predicate, that is, the predicate will appear in the head of hypothesized clauses. The second argument is the name and arity of a predicate that appears in the body of such clauses.

Mode declaration and determination in this study are shown in Figure 10(e) and (f), respectively. According to these instructions, the head of a hypothesis is the predicate placement, and the body of a hypothesis is a combination of attributes represented with the predicates type, angle, and relation. In other words, the body of the hypothesis represents the transitive closure based on the relationships in the ER model. Examples of rules are shown in Figure 10(g).

## 6. Analysis

ASMT experiments were conducted, and the experimental cases were analyzed by a system integrated from the CAD system, the Relationship-Recognizing module, and Progol.

### 6.1 RESULTS OF ASMT EXPERIMENTS AND LEARNING METHOD

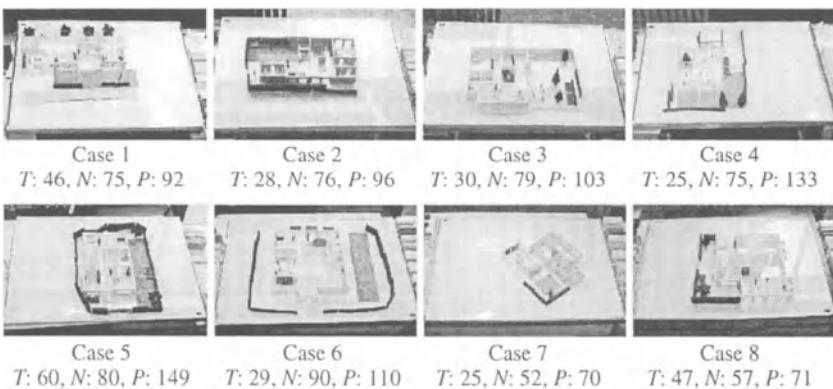
One ASMT trial was conducted on each of the four university students trained in architectural design and the four untrained university students. Results are shown in Figure 11. Progol induced rules from the following two kinds of input data.

- (1) Input data in which the placements by the trained students were set as positive and the placements by the untrained students were set as negative.
- (2) Input data in which the placements by the untrained students were set as positive and the placements by the trained students were set as negative.

Rules induced from input data (1) and (2) indicate the characteristic patterns of the trained students and those of the untrained students, respectively.

### 6.2 RESULTS OF PROGOL LEARNING

As a result, 147 and 149 rules were extracted from data (1) and (2), respectively. In this paper, the rules that cover the placements by more than half of the members of each group were regarded as characteristics common



*Figure 11.* Results of ASMT experiments on architecturally trained students (upper) and untrained students (lower) (working time  $T$  (min.) of the experiment, the number of miniatures on the white board at the end of the experiment  $N$ , the number of placements  $P$ )

to the group. The numbers of the common rules were 28 and 32, respectively. Rules of the trained students and the untrained students are shown in Tables 2(a), 2(b), 3(a), and 3(b) respectively. The coverage shown in the tables is defined as the ratio of positive examples covered by the rules to the total number of positive examples. The rule with the largest coverage is regarded as most general. The variables shown in the "diagram" column in these Tables denote the ID numbers of objects that were generalized. Reverse alphabetical order indicates the order of placement is reversed, where "A" would represent the last placement.

## 7. Discussion

### 7.1 VALIDATION OF RULES

We measured predictive accuracy for unseen cases by using four-fold cross-validations. The eight experimental cases shown in Figure 11 were split into four folds. Each fold contains two cases by the trained student and the untrained student. The examples containing three folds were set as training examples. Rules were induced from the training examples by progl. The examples contained in a remaining fold were classified into negative or positive by using the induced rules. The above procedures were repeated four times while changing the combination of the folds. Predictive accuracy is calculated by  $PA = (P^+ + P^-)/E$ , where  $P^+$  and  $P^-$  are the number of correctly predicted positive examples and negative examples, respectively, and  $E$  is

the total number of the training examples. In the case where the placements by the trained students were set as positive examples, the average of *PA* was 0.511. In the case where the placements by the untrained students were set as positive examples, the average of *PA* was 0.572.

## 7.2 INTERPRETATION

**•Patterns found for the architecturally trained students.** Rules of the trained students were interpreted as follows.

1. Rules *T1*, *T2*, *T3*, *T4*, *T9*, *T10*, *T11*, *T16*, *T17*, *T18*, *T20*, *T23*, *T27*, and *T28* refer to only relationships, indicating that the objects are separated from each other, e.g. "irreg," "close," "li\_detach-r." It can be inferred from these rules that one of the characteristics of the trained students is placement with each object separated from the others.
2. Rules *T1* and *T23* state that three objects were frequently placed with "semi\_irreg" or "irreg" relationships, which indicate that objects are parallel to each other. It can be inferred from these rules that the trained students constructed stratiform compositions.

TABLE 2(a). Rules found for trained students (ID of rule *RID*, Coverage of rule *Cov(%)*, Number of members whose placements are covered by the rule *CM*, Description of rule, and Diagram of rule)

| <i>RID</i> | <i>Cov(%)</i> | <i>CM</i> | Description of common rule                                                                                       | Diagram |
|------------|---------------|-----------|------------------------------------------------------------------------------------------------------------------|---------|
| <i>T1</i>  | 3.78          | 3         | placement(A):-relation(A,irreg,right,B,C), relation(C,semi_reg,right,D,E),relation(E,i_attach,top,F,G).          |         |
| <i>T2</i>  | 3.54          | 4         | placement(A):-angle(A,90),relation(A,close,none,B,C).                                                            |         |
| <i>T3</i>  | 2.12          | 4         | placement(A):-angle(A,90),relation(A,irreg,left,B,C), angle(C,90),relation(A,irreg,left,2,C),type(C,separation). |         |
| <i>T4</i>  | 1.89          | 3         | placement(A):-relation(A,li_detach-r,bottom,B,C), type(C,wall),relation(C,irreg,left,D,E),angle(E,0).            |         |
| <i>T5</i>  | 1.89          | 4         | placement(A):-type(A,person),angle(A,90).                                                                        |         |
| <i>T6</i>  | 1.42          | 4         | placement(A):-type(A,separation),angle(A,90), relation(A,isolation,none,B,C).                                    |         |
| <i>T7</i>  | 1.18          | 3         | placement(A):-type(A,mat),relation(A,isolation,none,B,C).                                                        |         |
| <i>T8</i>  | 1.18          | 3         | placement(A):-relation(A,i_attach,bottom,B,C), relation(C,bend_attach-r,top,I,D),type(D,wall).                   |         |
| <i>T9</i>  | 1.18          | 4         | placement(A):-relation(A,i_detach,bottom,B,C), relation(C,li_detach-r,top,D,E).                                  |         |
| <i>T10</i> | 1.18          | 3         | placement(A):-angle(A,fuzzy), relation(A,li_detach-r,bottom,B,C),type(C,separation).                             |         |

TABLE 2(b). Rules found for trained students (continuation from Table 2(a))

| RID | Cov(%) | CM | Description of common rule                                                                                       | Diagram |
|-----|--------|----|------------------------------------------------------------------------------------------------------------------|---------|
| T11 | 1.18   | 3  | placement(A):-angle(A,0),relation(A,irreg,left,7,B).                                                             |         |
| T12 | 1.18   | 4  | placement(A):-relation(A,on,center,B,C).                                                                         |         |
| T13 | 0.94   | 3  | placement(A):-relation(A,i_detach,top,B,C), relation(C,isolation,none,D,E).                                      |         |
| T14 | 0.94   | 3  | placement(A):-type(A,wall),angle(A,90), relation(A,isolation,none,B,C).                                          |         |
| T15 | 0.94   | 3  | placement(A):-type(A,person),relation(A,isolation,none,B,C).                                                     |         |
| T16 | 0.94   | 3  | placement(A):-relation(A,i_detach,top,8,B).                                                                      |         |
| T17 | 0.94   | 3  | placement(A):-relation(A,reg,left,12,B).                                                                         |         |
| T18 | 0.94   | 3  | placement(A):-type(A,furniture),angle(A,90), relation(A,i_detach,bottom,B,C).                                    |         |
| T19 | 0.94   | 3  | placement(A):-relation(A,t_attach,right,4,B).                                                                    |         |
| T20 | 0.94   | 3  | placement(A):-type(A,person), relation(A,i_detach,bottom,B,C).                                                   |         |
| T21 | 0.94   | 3  | placement(A):-relation(A,i_detach-l,top,B,C), relation(C,t_attach,left,D,E).                                     |         |
| T22 | 0.94   | 3  | placement(A):-type(A,wall),relation(A,i_attach,top,B,C), relation(C,irreg,left,7,D).                             |         |
| T23 | 0.94   | 3  | placement(A):-relation(A,irreg,right,7,C), relation(C,irreg,left,D,E).                                           |         |
| T24 | 0.94   | 3  | placement(A):-angle(A,90),relation(A,t_attach,top,B,C), type(C,wall).                                            |         |
| T25 | 0.71   | 3  | placement(A):-angle(A,90),relation(A,i_attach,bottom,B,C), relation(C,isolation,none,D,E).                       |         |
| T26 | 0.71   | 3  | placement(A):-angle(A,0),relation(A,i_attach,top,B,C), angle(C,0),relation(C,bend_attach-r,top,B,D),angle(D,90). |         |
| T27 | 0.71   | 3  | placement(A):-type(A,furniture),angle(A,90), relation(A,i_detach-l,bottom,B,C),type(C,furniture).                |         |
| T28 | 0.71   | 3  | placement(A):-type(A,person),relation(A,close,none,B,C), relation(C,i_detach-r,bottom,D,E).                      |         |

3. Rules T2, T3, T5, T6, T14, T18, T24, T25, T26, and T27 refer to an object angle of "90." It can be inferred from these rules that the trained students placed objects in the vertical direction.
4. Rules T9, T13, T16, T18, and T20 state that objects with the "i\_detach" relationship were frequently placed. It can be inferred from these rules that one of the characteristics of the trained students is the placement in a straight line with the object's ends detached.

6. Rules  $T6$ ,  $T7$ ,  $T13$ ,  $T14$ ,  $T15$ , and  $T25$  state that objects were frequently placed in "isolation." It can be inferred from these rules that trained students constructed elements in isolation, and integrated them at the end.

• **Patterns found for the untrained students.** Rules of the untrained students were interpreted as follows.

- Rules  $U2$ ,  $U3$ ,  $U4$ ,  $U6$ ,  $U8$ ,  $U13$ ,  $U16$ ,  $U19$ ,  $U22$ ,  $U23$ ,  $U24$ ,  $U26$ ,  $U27$ ,  $U28$ ,  $U29$ ,  $U31$ , and  $U32$  only refer to relationships indicating that the objects are in contact with the others at their ends, e.g. "bend\_attach-l," and "i\_attach." It can be inferred from these rules that one of the characteristics of the untrained students is placement with each object in contact with the others.

TABLE 3(a). Rules found for untrained students (ID of rule  $RID$ , Coverage of rule  $Cov(%)$ , Number of members whose placements are covered by the rule  $CM$ , Description of rule, and Diagram of rule)

| $RID$ | $Cov(%)$ | $CM$ | Description of common rule                                                                                                                          | Diagram |
|-------|----------|------|-----------------------------------------------------------------------------------------------------------------------------------------------------|---------|
| $U1$  | 3.72     | 3    | placement(A):-relation(A,irreg,left,B,C), relation(C,i_attach,top,I,E),type(E,wall),angle(E,0).                                                     |         |
| $U2$  | 2.98     | 3    | placement(A):-relation(A,bend_attach-l,top,B,C), relation(C,i_attach,bottom,D,E).                                                                   |         |
| $U3$  | 2.98     | 4    | placement(A):-type(A,wall),relation(A,bend_attach-r, bottom,B,C),type(C,wall),relation(C,i_attach,top,D,E).                                         |         |
| $U4$  | 2.73     | 3    | placement(A):-relation(A,i_attach,top,B,C),type(C,wall), relation(C,i_attach,bottom,D,E),type(E,wall).                                              |         |
| $U5$  | 2.73     | 4    | placement(A):-relation(A,i_attach,top,B,C), relation(C,li_detach-r,bottom,D,E),type(E,wall).                                                        |         |
| $U6$  | 2.73     | 3    | placement(A):-relation(A,i_attach,bottom,B,C), relation(C,bend_attach-l,bottom,D,E).                                                                |         |
| $U7$  | 2.48     | 3    | placement(A):-relation(A,close,none,B,C),type(C,mat), relation(C,close,none,D,E),type(E,furniture).                                                 |         |
| $U8$  | 2.23     | 4    | placement(A):-relation(A,bend_attach-l,bottom,1,B), relation(B,bend_attach-l,bottom,C,D).                                                           |         |
| $U9$  | 2.23     | 3    | placement(A):-relation(A,bend_attach-l,bottom,B,C), relation(C,irreg,left,D,E),type(E,wall), relation(E,bend_attach-r,bottom,F,G).                  |         |
| $U10$ | 1.99     | 3    | placement(A):-type(A,separation),relation(A,i_attach, top,B,C),type(C,separation),relation(C,irreg,left,D,E), relation(E,bend_attach-r,bottom,F,G). |         |
| $U11$ | 1.74     | 3    | placement(A):-type(A,balcony).                                                                                                                      |         |
| $U12$ | 1.74     | 3    | placement(A):-relation(A,irreg,right,B,C),relation(C, li_detach-r,bottom,D,E),relation(E,bend_attach-r,bottom,F,G).                                 |         |

TABLE 3(b). Rules found for untrained students (continuation from Table 3(a))

| RID | Cov(%) | CM | Description of common rule                                                                                                              | Diagram |
|-----|--------|----|-----------------------------------------------------------------------------------------------------------------------------------------|---------|
| U13 | 1.74   | 3  | placement(A):-relation(A,bend_attach-l,bottom,B,C), type(C,wall),angle(C,0),relation(C,i_attach,top,D,E).                               |         |
| U14 | 1.74   | 3  | placement(A):-type(A,separation),relation(A,bend_attach-l,bottom,B,C),relation(C,irreg_left,D,E),relation(E,bend_attach-l,top,F,G).     |         |
| U15 | 1.74   | 4  | placement(A):-relation(A,li_detach-l,bottom,B,C), relation(C,bend_attach-r,bottom,D,E).                                                 |         |
| U16 | 1.74   | 4  | placement(A):-relation(A,bend_attach-r,top,B,C), type(C,wall),relation(C,bend_attach-l,bottom,D,E), relation(E,i_attach,bottom,F,G).    |         |
| U17 | 1.49   | 3  | placement(A):-type(A,furniture),relation(A,li_detach-r,top,B,C),relation(C,bend_attach-r,bottom,D,E).                                   |         |
| U18 | 1.49   | 4  | placement(A):-type(A,furniture), relation(A,li_detach-r,top,B,C),relation(C,i_attach,top,D,E).                                          |         |
| U19 | 1.24   | 4  | placement(A):-relation(A,bend_attach-r,bottom,4,B).                                                                                     |         |
| U20 | 1.24   | 3  | placement(A):-relation(A,bend_attach-r,bottom,B,C), relation(C,li_detach-l,top,D,E).                                                    |         |
| U21 | 1.24   | 3  | placement(A):-type(A,separation),relation(A,i_attach,bottom,B,C),type(C,out_door),relation(C,semi_regular,right,D,E).                   |         |
| U22 | 1.24   | 3  | placement(A):-relation(A,bend_attach-r,bottom,1,B),type(B,separation),relation(B,i_attach,bottom,C,D),type(D,out_door).                 |         |
| U23 | 1.24   | 4  | placement(A):-relation(A,i_attach,top,1,B),relation(B,i_attach,top,C,D),relation(D,bend_attach-l,top,E,F).                              |         |
| U24 | 1.24   | 3  | placement(A):-relation(A,bend_attach-l,bottom,B,C), angle(C,0),relation(C,bend_attach-r,bottom,D,E).                                    |         |
| U25 | 1.24   | 3  | placement(A):-type(A,furniture),relation(A,close,none,B,C), relation(A,close,none,1,C),relation(C,isolation,none,D,E).                  |         |
| U26 | 0.99   | 3  | placement(A):-relation(A,i_attach,bottom,B,C), relation(C,bend_attach-l,top,2,D).                                                       |         |
| U27 | 0.99   | 3  | placement(A):-relation(A,bend_attach-r,bottom,B,C), relation(C,i_attach,top,1,D).                                                       |         |
| U28 | 0.99   | 3  | placement(A):-type(A,sanitary),relation(A,i_attach,top,B,C).                                                                            |         |
| U29 | 0.99   | 3  | placement(A):-relation(A,bend_attach-r,bottom,B,C), type(C,wall),angle(C,90),relation(C,bend_attach-r,top,D,E).                         |         |
| U30 | 0.99   | 3  | placement(A):-relation(A,li_detach-l,bottom,B,C), type(C,wall),relation(C,bend_attach-r,top,D,E), relation(E,bend_attach-l,bottom,F,G). |         |
| U31 | 0.74   | 3  | placement(A):-relation(A,bend_attach-r,top,B,C), relation(C,bend_attach-r,bottom,4,D).                                                  |         |
| U32 | 0.74   | 3  | placement(A):-relation(A,i_attach,bottom,3,B),type(B,wall).                                                                             |         |

2. Rules *U2*, *U3*, *U13*, and *U27* state that objects with the "i\_attach" relationship were placed first, and another object with a relationship such as "bend\_attach-l" or "bend\_attach-r" was placed after that with its end in contact at the joint of the former objects. It can be inferred from these rules that compositions by the untrained students are more spread out in branches.
3. Rules *U8*, *U12*, *U17*, *U29*, and *U31* state that three objects with "bending" relationships in the same direction were placed. It can be inferred from these rules that the untrained students constructed U-shaped enclosures.
4. Rule *U1* states that object *C* is related to object *E*, of an angle "0," with the "i\_attach" relationship, and object *A* is related to object *C* with the "irreg" relationship. This condition means that all of the objects have "0" angle. It can be inferred from these rules that the untrained students constructed in a horizontally parallel manner.

### 7.3 COMPARISON BETWEEN THE TWO GROUPS

By comparing the rules of the trained and the untrained students, the contrasts between the two groups shown in Table 4 were surmised.

TABLE 4. Contrasts between architecturally trained individuals and untrained individuals

| Feature                    | Group | Trained individuals                                                                                        | Untrained individuals                                                                                   |
|----------------------------|-------|------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------|
| Angle                      |       |  vertical direction      |  horizontal direction |
| Condition of contact       |       |  not in contact         |  in contact          |
| Characteristic composition |       |  stratiform composition |  U-shaped enclosure  |

## 8. Conclusions

ILP was applied to the analysis of spatial composition processes using ASMT. Complexly structured data of spatial composition processes that consist of many objects, relationships between them, and their attributes were modeled in first-order logic. A RR (Relationship-Recognizing) module was designed to convert the spatial composition process into first-order representations automatically, after the actual model was recreated using a CAD system.

One ASMT experiment was conducted on each of the four architecturally trained students and on the four untrained students. These experimental

cases were analyzed by a system integrated from Progol, which is one of the ILP systems, and the RR module. As a result, 28 rules for the trained students and 32 rules for the untrained students were found. These rules stated characteristic patterns of miniature arrangement of trained students (i.e. (1) placing objects with each object separated from the others, (2) stratiform composition, (3) placing objects in the vertical direction, (4) placing objects in a straight line with their ends detached, and (5) constructing elements in isolation) as well as the characteristic patterns of miniature arrangement of untrained students (i.e. (1) placing each object in contact with the others, (2) spread out elements in branches, (3) U-shaped enclosure, and (4) parallel composition in the horizontal direction).

In the future, we will need to analyze a greater number of experimental cases in order to find more universal patterns. We also need to refine the RR module in order to recognize more geometric relationships among objects.

## References

- Bollnow, O: 1963, *Mensch und Raum*, W, Kohlhammer GmbH, Stuttgart.
- Koning H and Eizenberg J: 1981, The language of the prairie: Frank Lloyd Wright's prairie houses, *Environment and Planning B* **8**: 295-323
- Lavrac, N, Dzeroski, S, and Numao, M: 1999, Inductive logic programming for relational knowledge discovery, *New Generation Computing* **17**: 2-23.
- Merleau-Ponty, M: 1945, *Phénoméologie de la Perception*, Gallimard.
- Muggleton, S and De Raedt, L: 1994, Inductive logic programming: theory and methods, *Journal of Logic Programming* **19-20**: 629-678.
- Muggleton, S: 1995, Inverse entailment and PROGOL, *New Generation Computing* **13**: 245-286.
- Numao, M and Shimizu, S: 1997, Data mining for distribution industry, *Journal of Japanese Society for Artificial Intelligence* **12**(4): 528-535 (in Japanese).
- Okazaki, S and Ito, T: 1992a, Architectural space montage technique and schizophrenic patient, *Journal of Architecture, Planning, Environment Engineering AII* **436**: 127-137 (In Japanese).
- Okazaki, S: 1992b, Architectural space montage technique and school children, *Journal of Architecture, Planning, Environment Engineering AII* **438**: 109-118 (In Japanese).
- Okazaki, S, Ooi, F, et al: 1997, Architectural space montage technique and mentally handicapped children, *Journal of Architecture, Planning, Environment Engineering. AII* **496**: 237-245 (In Japanese).
- Okazaki, S, Yanagisawa, K., et al: 1999, Architectural space montage technique and kindergarten children, *Journal of Architecture, Planning, Environment Engineering AII* **518**: 313-320 (In Japanese).
- Piaget, J and Inhelder, B. 1963, Les images mentales, in P Fraisse and J Piaget (eds), *Traité de Psychologie Expérimentale: 7 L'intelligence*, Presses Universitaires de France, Paris, pp. 65-108.
- Quinlan, JR: 1993, *C4.5: Programs for Machine Learning*, Morgan Kaufmann, San Mateo, CA.
- Schulz, CN: 1971, *Existence, Space and Architecture*, Studio Vista Limited, London.

- Shimazu, K and Hurukawa K: 2000, KDD system, DB-Amp -design, implementation and its application to an expert system, *Journal of Japanese Society for Artificial Intelligence* **15**(4): 629-637 (In Japanese).
- Srinivasan, A., and Camacho, R.: 1999, *P-Progol User Manual*  
<http://oldwww.comlab.ox.ac.uk/oucl/groups/machlearn/PProgol/ppman.html>.  
Hosted at the Oxford University, Machine Learning at the Computing Laboratory.
- Sugiura, N., and Okazaki, S.: 2001, Analysis of spatial composition process by inductive logic programming - extracting peculiar rules in process of spatial composition on space montage technique -, *Journal of Architecture, Planning, Environment Engineering AJE* **546**: 141-148 (In Japanese).
- Tsumoto, S and Hiroshi T: 1997, Medical application of data mining, *Journal of Japanese Society for Artificial Intelligence* **12**(4): 536-543 (in Japanese)

## TOWARDS AN ARCHITECTURAL DESIGN SYSTEM BASED ON GENERIC REPRESENTATIONS

SVIATASLAU PRANOVICH, HENRI ACHTEN AND JARKE J  
VAN WIJK

*Technische Universiteit Eindhoven*  
*The Netherlands*

**Abstract.** Computer Aided Architectural Design systems offer a broad scope of drawing and modeling techniques for the designer. Nevertheless, they offer limited support for the early phases of the design process. One reason is that the level of abstraction is too low: the user can define walls and such in great precision, but no support is offered to define global or structural characteristics of the design. In previous research, structuring representations such as zones, grids, and axial systems, as well as object representations such as element vocabularies and contours have been identified as formative elements in architectural design. They have been defined and described as so-called graphic units. Furthermore, sets of graphic units form generic representations, in which the relationships between graphic units are well defined and meaningful to the architect. Graphic units and generic representations, or their equivalents, are not supported in current CAAD systems. In this paper, a system is proposed that is based on graphic units and generic representations. It has tools for making graphic units and also provides a way to define and maintain changes in the relations between graphic units. This is implemented on the basis of a graph representation where the nodes represent graphic units and the edges relations between graphic units. Changes in the graphic units are propagated through the graph. The paper shows how the theoretical work on graphic units and generic representations leads to the current implementation. The system structure is explained and the way changes in graphic units are propagated through the graph representation.

### 1. Introduction

In the early phase of the architectural design process, the designer is searching for basic solutions to the design task. In this process, graphic representations play an important role for concept development and external

representation of design ideas (Verstijnen 1997; McFadzean 1999). The production of graphic representations is fast and numerous (McCall et al. 1997), the level of detail is well adjusted to the amount of knowledge present about the design task and solution, and there are generally used conventions of depiction such as plan, section, and perspective that aid in understanding the inherently three-dimensional nature of building design (Akin 1986). In the early design phase, the plan representation is used predominantly to explore and represent building design layout, circulation, spatial composition, relation with site, and so forth.

Current drawing systems in the domain of architectural design offer primitives such as lines, rectangles, circles, etc., and enable the user to manipulate them, via operations such as transformation, grouping, and aligning. A Computer Aided Architectural Drawing system offers primitives tailored to the domain, such as walls, doors, and windows, and has built-in rules to simplify manipulations. These systems have reached a high level of sophistication, and are perfectly suited for the production of the final technical drawings in the closing phases of the design process. However, they do not offer support for the early phase of the design process when concept formation is important, supported by sketch-like drawing activity.

In order to develop tools that support architects in an architectural fashion, it is necessary to look at the way architects use graphic representations in the early phase of design. In this paper, the concepts of graphic unit and generic representation are introduced for this purpose. These will be outlined in the next section. A design system is built based on these concepts. The implementation approach is outlined in Section 3 and the underlying graph control structure in Section 4.

## 2. Graphic Units, Generic Representations And Design Support

Research in the use of drawings by architects leads to the view that architects use well-defined forms of graphic representations to depict their design intentions (Do 1997, 1998; Verstijnen 1997; McFadzean 1999; Koutamanis 2001). In Achten (1997), these forms have been identified and described as graphic units. A graphic unit is a set of graphic elements that are organized in a specific way and that have a generally agreed upon meaning for the designer. Examples of graphic units are: Grid, Contour, Axial system, and Zone.

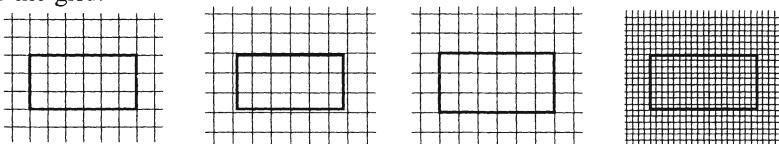
Sets of graphic units form so-called generic representations, such as Contour in Grid, Contour and Axial system in Grid, etc. Any drawing that belongs to the class defined by a generic representation deals with the same kind of design issues, thus enabling designers to reason about and share their design drawings. For this reason, we propose that graphic units and

generic representations form a good basis to implement these higher-level knowledge structures in Computer Aided Architectural Design tools for the early design phase. Furthermore, since graphic units define generally agreed upon design representations, they can be used to analyze design processes (Achten and van Leeuwen 1998; 1999), or to build design information systems (Achten, Oxman and Bax 1998; Achten 2000).

Graphic units can be considered as a medium to express the ideas in an architectural design. In the early phases the architect is trying to resolve basic issues of composition, layout, structure, circulation, and preliminary indication of arrangements. Most graphic units are concerned with structuring the design (e.g. Grid, Zone, Circulation system), rather than describing the design itself (e.g. Contour, Complementary contours, Element vocabulary).

For example, a designer uses an Axial system to show the symmetry between two contours. The Axial system does not only describe the design, but also structures the information in the design. Current drawing systems offer options to mirror graphic objects, but when the operation has been done, the information that two contours are symmetric is lost. Furthermore, whereas an architect often explicitly denotes an axial system, the operation of mirroring often involves only a temporary indication of the mirror action. The presence of an axial system as a separate structuring device and object in the drawing is ignored in such an approach.

A Grid is another good example in this respect. Drawing systems offer grids to position elements more easily, but only as a tool and not as a meaningful element on its own. There is a strong link between a grid and the objects that are coordinated on this grid, Figure 1, and changing one or the other should have consequences for either the grid or the objects that relate to the grid.



*Figure 1 . Relationship between Grid and Contour. a: A contour corresponds to a grid; b: The grid-module is increased; c: The contour changes to correspond with the grid again; d: The grid subdivides to correspond to the contour and the new module.*

Another example concerns the simultaneous use of multiple grids, which structure the design and relate to high-level concepts behind the final design (for example Bax 1985). The architect experiments with different

orientations and scales, coordinates objects and grids relative to each other, and observes how the grids influence and enhance each other.

We propose that a design system has to incorporate not only the pure graphical and geometric aspects of each individual graphic unit, such as position, orientation, color, but also the conceptual and abstract level of information representation based on the designer's understanding of graphic representations. Current drawing and design systems fall short in this respect. The concept of graphic unit and generic representation can be instrumental in this development since they combine graphic information with design content. We aim at the development of a system, where all graphic units (Contour, Grid, Axial system, etc.) have the same status: all graphic units are objects that can be manipulated and changed, and have a suitable visual representation.

A second key point concerns the mutual interaction between graphic units, which almost by definition is crucial. Using the axial system as an example again, when a designer changes something in one contour, then the changes must be reflected in the other contour also. When the designer changes the position of the axial system, the position of the contours also has to change.

Summarizing, we aim at a system where the user can define a design in terms of graphic units and relations between them, such that during subsequent object manipulation all objects are active and influence each other and react to user actions in a meaningful and predictable way. This should enable an architect to define a design space and to explore this space in an effective, efficient, and hopefully creative way.

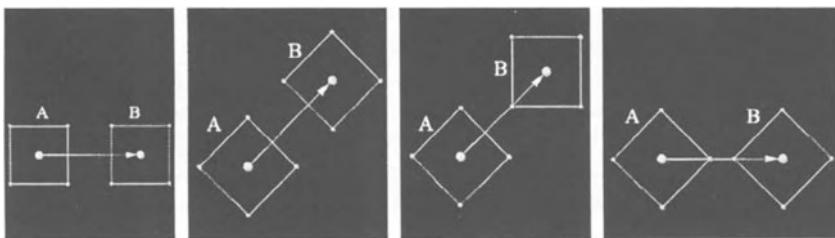
With respect to related work, there have been recently many efforts to further develop graphic support of (architectural) design. The work by Gross et al. (1988) and Gross (1990) has been informative for the research on generic representations, in particular the inherent definition of constraints in graphic representations. More recent examples for graphic design support are "Netdraw" by Qian and Gross (1999), "SketchBoX" by Stellingwerff (1999), "Piranesi" by Richens (1999), a transparent sketch-tool by Trinder (1999), and "EsQUIsE" by Leclercq (2001). Of these, in particular Leclercq (2001) is much related, as the "EsQUIsE" system interprets sets of sketch lines as spaces and functions. Characterizing strokes, or so-called glyphs, is also the basis of the "Cocktail Napkin" application (Gross 1996). Both systems work on real-time interpretation of strokes. The system that is presented in this paper does not utilize stroke or glyph recognition, but offers instead a set of drawing tools that are based on graphic units and that function accordingly. The graphic unit and graph representation allow individual handling of parts of the drawing while maintaining relationships.

### 3. Design System Based on Generic Representations

The design system under development evolves by implementing graphic units, defining the possible relations between graphic units and handling both graphic units and relationships. The current prototype supports the graphic units Contour and Grid, and enables the user to define relations between them (in effect using the generic representation Contour in grid). The images in Figures 2 and 3 are screenshots taken from the prototype system.

Instances of graphic units (multiple contours and grids) and relations can be added and changed by direct manipulation. Furthermore, each graphic unit and relation has several properties that can be set by the user and that influence how manipulations affect other graphic units. This gives the user much flexibility in how he wants to explore design possibilities. In this section we give an example, in the next section the interior machinery will be described.

Consider two contours A and B, where A controls B, Figure 2a. Suppose we rotate A, what should happen to B? The answer is not unique. In this case, nothing could happen (in the case when B has to react only to translation), B can be rotated around the center of A, Figure 2b, the center of B can be rotated around the center of A, Figure 2c, or B can be rotated around its own center, Figure 2d. In our system the user can specify which behavior is desired by setting attributes of the relations and graphic units. In the user interface, the relations themselves are defined by selecting points of different graphic units and pressing a button, upon which the relation is visualized as an arrow.



*Figure 2.* a: Intitial positions of contours; b: Rotation around A; c: Rotation with the rotation for contour B constrained; d: Rotation with the rotation for relation A-B constrained.

One graphic unit can have multiple parents. A parent is a graphic unit with a supervising relation to this graphic unit. Such parents influence the behavior, where the form of influence depends on the type of graphic unit.

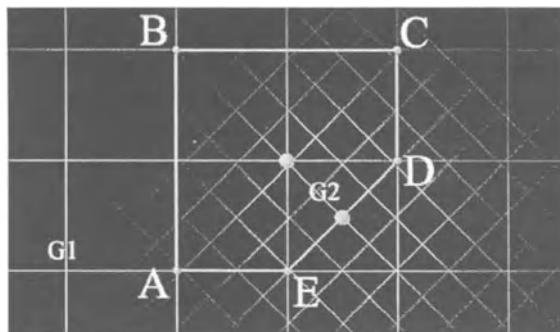
The previous example concerned two graphic units of the same type. The situation become more complex and also more challenging when types are mixed. We take the grid as an example. How does a grid influence other objects? Normally a grid helps to determine dimensions and locations of objects:

- Determine dimensions: the lines in a grid are spaced at a particular distance, called the “module”. This module is a basic unit of measurement. Every element that is placed in the grid, and that adheres to the grid, has dimensions that are a whole multiple of the module. So, if the module is 30 cm, then any dimension of the object on the grid is 0, 30, 60, 90, 120 cm, ... etc.
- Determine locations: the intersections of the lines in the grid determine the begin- and endpoints of elements that are placed in the grid. In this way, constant dimensions of the element and coordination of the objects in the grid is ensured.

Also, other objects can influence a grid. Let us take the generic representation “Contour in grid”, which consists of two graphic units: a Contour and a Grid. There are two types of behavior possible for such a generic representation. Firstly, a contour is bound to a grid. This situation occurs for instance when a designer positions a building in the space defined by a grid. The points and lines of the contour must share the orientation and position of the grid. Secondly, a grid can be bound to a contour. This situation can happen when a designer wants to define a new measuring system to define new objects within a specific contour. The properties of the grid are now derived from the position and orientation defined by the contour.

As an example, consider a grid G1, which influences a contour ABCDE, to which a second grid G2 is attached, Figure 3. The local influence of the grid G2 is visualized by fading the grid off a distance from the parent object. When a user translates the contour ABCDE (one of the points or the whole contour), the system helps the user to position the contour within the grid G1: The positions of points are rounded to the nearest grid point. Also, the position of the grid G2 will be affected. Again, extra flexibility is offered to define the relation between a grid and other graphic units. The following levels of geometric influence are offered:

- Line-Line level: the orientation of the supervised object must coincide with the orientation of the grid.
- Point-Line level: points of the supervised object must be located on one of the grid lines.
- Point-Point level: points of the supervised object must be located at intersections of grid lines.



*Figure 3.* Grid G1 (large module) influences contour ABCDE. The edge DE of contour ABCDE influences grid G2.

In this way, the prototype system can maintain not only the shapes of the contours, and coordinate these with the grid, but it also maintains the internal relationships between the grids and the contour. These relations are meaningful to the architect when he is working with grids and contours.

In the next section, we describe the internal structure of the system.

#### 4. Graph-Based Processing

Internally, a design is described as a graph, with graphic units as nodes and relations as edges. Relations can be either uni-directional (one graphic unit supervises another) or bi-directional (two graphic units influence each other). The current prototype however imposes constraints on the graph: Between any two nodes in the graph at most one path may exist. Hence, cycles are not allowed. The graph does not have to be connected. Multiple independent sub-graphs can be used.

The graph is used by the system to propagate geometrical transformations from one node to another. All graphic units and relations are shown as graphic objects that can be manipulated and changed by direct manipulation. The user is enabled to use translation, scaling, rotation, and skewing.

The use of a hierarchical structure to model geometric or graphic objects is standard in computer graphics (Foley et al. 1999). However, in standard drawing packages this mechanism is used only in a limited way: The user is allowed to hierarchically group elements. Our current system offers more flexibility, in the sense that for each kind of transformation and for each graphic unit and relation a separate decision can be made how transformations are dealt with.

The use of points and relations to model interactions between graphic units is partly based on work we have done earlier in the context of interactive simulation (Wijk and van Liere 1997). Again however, more flexibility is offered here, and also the supervision of graphic units such as a grid is novel.

A relation between two graphic units is presented graphically as an arrow between two points, which belong to related graphic units. Multiple connection points can be defined per graphic unit. Such a point defines the origin of the local coordinate system for subsequent transformations of a graphic unit. A geometric transformation of a relation changes the position of these connection points. The graphic unit itself (in this case the shape of a Contour or the spacing and direction of a Grid) is defined by another set of points. These points are defined in a local coordinate system attached to each graphic unit. A geometric transformation of a graphic unit changes its local coordinate system, and hence the position of the points in global coordinates.

The propagation of transformations starts at a point selected by the user. The transformation is propagated through the graph affecting all the graphic units and the relations on its path.

Let us take a simple example to show how the propagation of the geometrical transformation is dealt with. The contours A, B, C are sequentially connected by relations  $A \rightarrow B$  and  $B \rightarrow C$ . The connection points are the local origin of the graphic units, Figure 4a.

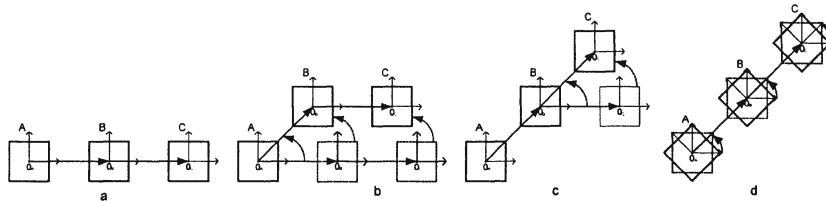


Figure 4. Propagation of rotation transformation.

When the user applies a rotation of 45 degrees to the geometrical center of contour A, this transformation is propagated as follows through the graph:

1. The transformation of the relation  $A \rightarrow B$  affects the sub-graph B-C: the local coordinate systems of contour B and contour C are translated to reflect the 45 degrees rotation of  $O_B$  around  $O_A$ . The contours B and C are translated as a result of this, Figure 4b.

2. The transformation of the relation  $B \rightarrow C$  affects the local coordinate system of contour C: the position is translated according to a 45 degrees rotation of  $O_C$  around  $O_B$ . Contour C is translated as a result, Figure 4c.
3. Finally the rotation around the origin is applied to the local coordinate systems of contour A, B, and C in succession, Figure 4d. The contours change their orientation together with the local coordinate systems.

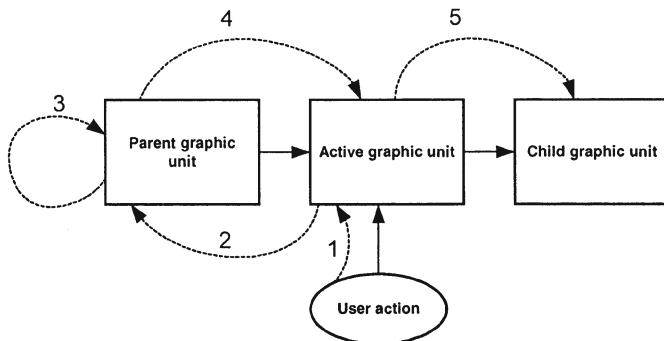
In this scheme all transformations are done independently and the final transformation of the graphic units is equal to the product of these transformations.

The user can set attributes for each graphic unit and relation to define which transformations in the graph have to be performed or not, such that a large variety of geometric relations is available to the user. For example, if the user in the previous case constrains the processing of the geometrical transformations to relations only (step 1 and 2), then the result is that only the contours are translated, Figure 4c. If only transformations of units are enabled, the result is that all contours rotate around a local origin. This latter option is not available in other design systems. And, each combination of these can be used.

The user can set the transformation attributes for each type of transformation individually. A click on the right mouse button on a relation or graphic unit gives a context menu from which all attributes of an object can be changed.

Another important aspect of the processing of geometrical transformation in the graph is the influence of parent objects. If an object that is manipulated by the user has an incoming relation from a grid, the influence of this grid has to be taken into account. The processing of influence takes place before the transformations are propagated through the graph. In the current version of the system, only the parent influence (and not for instance the grandparent influence) is taken into account. The scheme of the user action processing is presented below, Figure 5:

1. The active graphic unit gets an action from the user to make a geometrical transformation.
2. The active graphic unit sends a request to its parent(s) to check the transformation.
3. The parent checks, and if necessary modifies parameters of the transformation.
4. The parent sends the transformation back to the selected activated graphic unit.
5. The active graphic unit processes the transformation command and sends it to its child.



*Figure 5.* User action processing.

The type of parents influence depends on the type of graphic unit. So far, we have implemented the effect of a grid on other graphic units.

## 5. Discussion and Future Work

The system can offer total freedom in the relations between graphic units that are present. From the usage of the system both with researchers and architects in such a setting, it appeared that so much freedom was not productive since the user then had to maintain all meaningful relations. From the theory of generic representations outlined in section two, relationships between graphic units that hold for all generic representations can be implemented. In an experimental setting, it is then possible to check whether these relations in practice actually map with the predicted use, and if not, to which extent they are over- or under defined.

Furthermore, the user interface is critical to an unobtrusive use of the system. It is not that difficult to offer many options and much flexibility, a major challenge is to define interaction techniques and visual metaphors such that the user can define what he wants in an intuitive way. One possible route we pursue is to let the system offer alternatives, based on different settings of the attributes, from which the user can make a selection.

The development of the system is still in full progress. Addition of more graphic units is high on the priority list. The graphic unit of Axial system will be the next one. In particular the implementation of possible relationships between graphic units becomes more complex.

Nevertheless, the current prototype already has attracted attention from architects, which were especially enthusiastic about the option to define multiple, interacting grids. This was for us a first confirmation that the theory of generic representations offers an attractive foundation for an

interactive design system for the early phases of the architectural design process.

### Acknowledgements

The paper reports from ongoing Ph.D. work by S. Pranovich at the Department of Mathematics and Computer Science under supervision of J. van Wijk. The work is informed by discussions that take place in the context of E3DAD: Easy 3D Architectural Design, which is a collaboration between the groups of Visualization of the Department of Mathematics and Computing Science, the IPO Centre for User-System Interaction, and Design Systems of the Department of Architecture, Building and Planning of the Technische Universiteit Eindhoven.

### References

- Achten, HH and Leeuwen, JP van: 1998, A feature-based description technique for design processes: A case study, in H Timmermans (ed.), *Proceedings of the 4th Design & Decision Support Systems in Architecture & Urban Planning Conference*, Maastricht, Eindhoven, pp. 1-25.
- Achten, HH, Oxman, RM and Bax, MFTh.: 1998, Typological knowledge acquisition through a schema of generic representations, in J. Gero and F. Sudweeks (eds), *Artificial Intelligence in Design '98*, Kluwer, Dordrecht, pp. 191-207.
- Achten, HH and Leeuwen, JP van: 1999, Feature-based high level design tools: A classification, in G. Augenbroe C. Eastman (eds), *Computers in Building: Proceedings of the CAADFutures '99 Conference*, Kluwer Academic Publishers, Dordrecht, pp. 275-290.
- Achten, HH: 1997, *Generic Representations – An Approach for Modelling Procedural and Declarative Knowledge of Building Types in Architectural Design*, Ph.D. thesis, Technische Universiteit Eindhoven/Faculteit Bouwkunde, Eindhoven.
- Achten, HH: 2000, Design case retrieval by generic representations, in J.S. Gero (ed.), *Artificial Intelligence in Design '00*, Kluwer, Dordrecht, pp. 373-392.
- Akin, Ö: 1986, *Psychology of Architectural Design*, Pion Limited, London.
- Bax, MFTh.: 1985, The design of a generic grid, in R. Beheshti (ed.), *Design Coalition Team – Proceedings of the International Design Participation Conference*, Technische Universiteit Eindhoven, Eindhoven, pp. 320-333.
- Do, EY-L: 1997, Computability of design diagrams, in R. Junge (ed.), *CAAD Futures 1997*, Kluwer, Dordrecht, pp. 171-176.
- Do, EY-L: 1998, *The Right Tool at the Right Time: Investigation of Freehand Drawing as an Interface to Knowledge Based Design Tools*, Ph.D. Thesis, Georgia Institute of Technology, Georgia.
- Foley, JD, van Dam, A, Feiner, SK and Hughes, JF: 1990, *Computer Graphics - Principles and Practice*, 2nd Ed., Addison-Wesley, Reading MA.
- Gross, MD, Ervin, SM, Anderson, JA and Fleisher, A: 1988, Constraints: knowledge representation in design, *Design Studies* 9(3): 133-143.
- Gross, MD: 1990, Relational modeling: A basis for computer-assisted design, in M McCullough and WJ Mitchell and P Purcell (eds), *The Electronic Design Studio - Architectural Knowledge and Media in the Computer Era*, The MIT Press, Cambridge, MA, pp. 123-136.

- Gross, MD: 1996, The electronic cocktail napkin - computer support for working with diagrams, *Design Studies* 17(1): 53-70.
- Koutamanis, A: 2001, Prolegomena to the recognition of floor plan sketches - A typology of architectural and graphic primitives in freehand representations, in H Achten, B de Vries and J Hennessey (eds), *Design Research in the Netherlands 2000*, Eindhoven University of Technology, Eindhoven, pp. 95-105.
- Leclercq, PP: 2001, Programming and assisted sketching - Graphic and parametric integration in architectural design, in B de Vries, JP van Leeuwen and HH Achten (eds), *Computer Aided Architectural Design Futures 2001*, Kluwer, Dordrecht, pp. 15-31.
- McCall, R, Johnson, E and Smith, M: 1997, Hypersketching: Design as creating a graphical hyperdocument, in R. Junge (ed.), *CAAD Futures 1997*, Kluwer, Dordrecht, pp. 849-854.
- McFadzean, J: 1999, Computational Sketch Analyser (CSA): Extending the boundaries of knowledge in CAAD, in A Brown, M Knight and P Berridge (eds), *Architectural Computing from Turing to 2000 – Proceedings of the 17th Conference on Education in Computer Aided Architectural Design in Europe*, The University of Liverpool, Liverpool, pp. 503-510.
- Qian, D and Gross, MD: 1999, Collaborative design with Netdraw, in G Augenbroe and C Eastman (eds), *CAAD Futures'99*, Kluwer, Dordrecht, pp. 213-226.
- Richens, P: 1999, The Piranesi system for interactive rendering, in G Augenbroe and C Eastman (eds), *CAAD Futures'99*, Kluwer, Dordrecht, pp. 381-398.
- Stellingwerff, M: 1999, SketchBoX, in A Brown, M Knight and P Berridge (eds), *Architectural Computing from Turing to 2000 – Proceedings of the 17<sup>th</sup> Conference on Education in Computer Aided Architectural Design in Europe*, The University of Liverpool, Liverpool, pp. 491-497.
- Trinder, M.: 1999, The computer's role in sketch design: A transparent sketching medium, in G Augenbroe and C Eastman (eds), *CAAD Futures'99*, Kluwer, Dordrecht, pp. 227-244.
- van Wijk, JJ and van Liere, R: 1997, An environment for computational steering, in GM Nielson and H Mueller, H Hagen (eds), *Scientific Visualization: Overviews, Methodologies, and Techniques*, CS-Press, Los Alamitos, pp. 103-124.
- Verstijnen, IM: 1997, *Sketches of Creative Discovery – A Psychological Inquiry Into the Role of Imagery and Sketching in Creative Discovery*, Ph.D. Thesis, Delft University of Technology, Delft.

## DIGITAL SANDBOX

*Integrating landform making and analysis for landscape design*

ELLEN YI-LUEN DO  
*University of Washington*  
USA

**Abstract.** Digital Sandbox is a computational tool for landscape architecture design that provides design and analysis capabilities in the same environment. It uses image processing of hand gestures to infer the landform sculpting actions and simulates storm water accumulation over the digital terrain.

### 1. Introduction

#### 1.1 A SCENARIO

Imagine you are a landscape architect. You are commissioned to design a landscape on a hill site for green space and some residences. Imagine you could start sculpting the landform in a digital sandbox. In the sandbox you can form the hills, dig out valleys, and put vegetation and buildings on different locations with your hands. You could rotate and move the sandbox around to examine the quality of space, color and texture of your design. You could move the buildings and trees to different places. To examine how the summer lighting condition would impact your design you turn on the sunlight simulation. Realizing that storm water and flooding could be a major concern on this site, you activate the water flow simulation analysis. The terrain model would display different gradients of color. This visualization shows that certain areas in your landform design would have a high volume of water accumulation on rainy days. You quickly move the buildings, and plant more trees and the problems go away.

Imagine all these imaginings are true. What would it take to build a system like this that would provide both functions of design and analysis in a single environment? In this paper we describe a prototype system, called the Digital Sandbox, to support landscape design tasks. The Digital Sandbox

system integrates a quantitative model of an ecological process with a spatial model of a digital landform. It provides designers the ability to design a landform and simultaneously analyze the storm water flow over that landform. The system employs a gesture-based modeling interface for a designer to manipulate the spatial model using hand gestures in three-dimensional space.

### 1.2. BACKGROUND: WHY BUILD A DIGITAL SANDBOX?

Human hands have dramatically changed the earth's landscape over the past several hundred years. Landscape design is a complex task with multiple goals and constraints. It involves both reasoning of form and function. The landform design in three-dimensional space requires an understanding of the structure and the function of the land. Landscape architects must understand the ecological impacts of their designs as well as the formal aspects. To achieve these multiple goals, designers often employ a variety of tools from hand drawn sketches to simulation models. All these tools are indispensable to designers. However, in the design process, designers would like to move back and forth seamlessly from qualitative landform making to quantitative analysis without interruption. Designers would like to have the right tool at the right time in the same work environment. No such ideal tool exists—yet. Currently there is no landform design tool that provides design and analysis in the same environment. Designers often use sketches and cardboard models to explore the spatial qualities of a design, then use digital tools such as Geographic Information Systems (GIS) to perform ecological analysis.

We built the Digital Sandbox system to provide designers a single environment where they can sculpt a landform while obtaining feedback of their design. We call the system Digital Sandbox because it is digital, to compute ecological simulation, and it is also like a traditional sandbox, that is easy to interact with by hand. This paper is organized as follows. Section 2 reviews related work in design computing work in digital terrain modeling, gesture-based modeling methods, and research on integrating design and analysis tasks in computational environments. Section 3 provides an overview of the Digital Sandbox implementation. Section 4 describes a design use scenario of the Digital Sandbox. The paper concludes in Section 5 with a brief discussion and identifies future research directions of this new landscape design tool.

## 2. Related Work

The building of a digital sandbox concerns three areas in design computing: digital landform making, gesture-based modeling, and the integration of

design and analysis tasks in computational environments. To better position the Digital Sandbox, and to identify conceptual issues and methods for better earth-forming tools, this section reviews related work in these three categories.

## 2.1 DIGITAL LANDFORM MAKING

Landform making is fundamental to landscape architecture. Developing new terrain design tools is difficult because the representation and manipulation of topology is inherently complex (Sawyer 1998). However, there have been few innovations in the digital tools for such design task. Among these efforts, the Topographic Surface Sculptor (TSS) (Westort 1998) and Leveller™ (DaylonGraphics 2001) both utilize procedural metaphors to provide better support for creative landscape forming. Existing CAD-based tools require the designer to digitize contours to create mesh landform models. This process is tedious and time consuming. To ease this process, TSS uses a bulldozer as metaphor for landform operations (Westort 1996). It consists of three parts: a blade tool to sculpt an excavation path, a library of geometric shapes, and landform operations. A designer can select a bulldozer blade, and adjusts its shape to create a digital landform. The transformation of the digital surface is executed through the designer-defined path, operated with a given blade and shape.

A commercial height field modeler developed by Daylon Graphics called Leveller™ takes a similar but different approach. It is a raster-based paint program. The interface is a grid surface of pixels. Different colors represent different elevation heights. A designer can modify the terrain model by editing the height field using a paintbrush function, similar to using a conventional paint program. The system provides a variety of brush sizes and 'dig-and-raise' tools to excavate and modify the terrain. The plan view windows provide the ability to edit the value of the field heights. A perspective window displays the perspective of the resulting terrain.

These systems use metaphors such as 'bulldozer' and 'paint' to make it easier for a designer to understand the operations on a digital terrain map. By providing more construction-like methods for terrain manipulations, these systems improve over current CAD tools.

## 2.2 GESTURE-BASED MODELING

Current trends see the emergence of a number of gesture-based interfaces for design applications. These systems use image processing and pattern matching of gestures to pointing or modeling operations in 3D modeling. For example, GestureVR is a vision-based input interface for three-

dimensional object creation and manipulation (Segen and Kumar 1998). It is a single-handed gesture-recognition system with input from two video cameras. The system does not require the user to wear a data glove for gesture detection. Instead, it employs two cameras placed about three feet above the gesture space to capture the plan and profile views of user's hand gestures. By computing and processing images from the image input the system determines the spatial location and configuration of the user's hand. The information of the 3D position (x, y, and z location) of the fingertip and the azimuth and elevation angles of the finger's axis is used to draw curves and input digital objects to the three-dimensional scene by selecting from a tool palette on the screen. Each gesture is mapped to a command. For example, the user can draw a line with an index finger, and make a fist gesture to indicate a selection in Segen and Kumar's '3D Scene Composer' environment.

The Two-handed Gesture Environment Shell (Nishino, et al. 1997), on the other hand, uses an instrumented data glove to interact with the virtual environment. Unlike Gesture VR, it is not vision-based. Instead, input from the data gloves is parsed through a driver for initial processing. The commands triggered by the hand gestures are identified through the shape, position, and orientation information from the data stream. The system employs pattern matching by a neural network to process dynamic hand gestures. A similar approach by Wexelblat places sensors at various points on the user's hands and upper body (Wexelblat 1995). He argues that a model of the user's body is necessary to understand human's natural gesture in a furniture layout design task. The system has two major components. First, the analyzer receives and analyzes input from the body model and sends the data to the interpreter. The interpreter then uses gesture input to add or move furniture in the virtual room. Gesture modeling is also used in artistic sculpturing to create 3D shapes. Surface Drawing (Schkolne, et al. 2001) uses a semi-immersive virtual environment called Responsive Workbench (Kruger and Frohlich 1994). The system combines the use of stereoscopic glasses and a data glove to create three-dimensional organic shapes. Users of the system can use hand movements and strokes in a 3D space to create digital objects. In addition to gestures, the system also provides a variety of tangible tools to operate upon objects. For example, a user can use tongs to move and rotate the drawing, or an eraser to remove portions of the drawing. All viewing and interactions occur in user's physical space.

These gesture-based modeling tools have great potential to be used in landform design. The ideal tool would integrate the creation of 3D geometry with analysis abilities.

### 2.3 INTEGRATING DESIGN AND ANALYSIS TASKS

To be useful for designers, computational environments should provide both design and analysis functionality. Many research projects have attempted this integration by providing a more visual interface to geographic information data, or to bring CAD and GIS applications into the same environment.

One such example of a hybrid GIS-CAD application was developed by Mayall et al. (Mayall, et al. 1994). To assess the visual impact of a proposed landscape design, a designer can input attributes such as building heights, colors, vegetation type and size to the GIS database and the system would invoke its display in a three-dimensional modeling environment of a CAD program. The system uses a simple scripting language to map and translate the GIS data into three-dimensional landscape objects in CAD representations. A project by Ervin (Ervin 1992) also creates 3D models from GIS data. It associates unique land use codes in the GIS database with a predefined library of object symbols – buildings, houses, vegetation, and paved surfaces. These systems are improvements over the existing tools because they link visualization and analysis together. However, designer can only make design changes on the GIS database, not in the CAD model. Often there is a significant lag time between analysis and visualization.

There are also projects focusing on visualization systems and providing interactive feedback. For example, the SmartForest project (Uusitalo, et al. 1997) provides real time update of the visual display of design changes. The system provides dialog boxes for user to change the growth, crowding, and thinning parameters of tree type, size and health in different growth models. These changes update the underlying database of forest composition and produce a new visualization display of the forest landscape environment. Similarly, the Toronto University Urban Modeling System (Danahy and Wright 1988) also provides user real-time feedback about design impact. Users can import a parcel file, and choose from pull-down menus to assign spatial zoning constraints and add buildings to the parcel. Design factors such as floor-space index and construction cost would then be displayed in a spreadsheet window. It provides numeric information display of the design impact after a designer's manipulation of the design.

These projects illustrate the efforts toward, and the continuing need for, integrating design and analysis tools to support better decision making in landscape design.

### 3. Implementation of the Digital Sandbox

In Spring 2001 we built a working prototype of the Digital Sandbox system. The Digital Sandbox is a gesture-based interface that integrates design and analysis tasks to support landscape design. A designer can use the Digital Sandbox to sculpt a digital terrain, add trees and buildings to the landform, and run a storm water accumulation model to predict environmental impacts. In the design process, designer can engage in landform sculpting, addition and manipulation of the landscape objects, as well as running the water flow simulation as often as needed, in a single work environment. These design tasks are executed by mapping hand gestures to specific predefined commands. The prototype system was built to explore the viability of such an application in digital land forming.

#### 3.1 SYSTEM OVERVIEW

There are two major parts of the Digital Sandbox system. First, it uses a machine-vision approach called Gesture Modeling (Gross and Kemp 2001), developed at our research lab, the Design Machine Group (DMG 2001). The second part of the system is the SandboxToy module, also developed at our research lab. This module consists of a set of object classes to accept changes of attributes and methods applied to them to compute and display the water flow simulation. The system building was reported as part of a Master of Landscape Architecture thesis by Robert M. Harris (Harris 2001).

The physical setup of the system has two primary components: a gesture modeling space and a 3D modeling display. The 3D modeling space is displayed on a translucent screen with rear projection of horizontally inverted image. The gesture modeling space is defined under a simple, desktop video camera positioned approximately three feet above a white tabletop in front of the modeling screen Figure 1 shows a designer interacting with the Digital Sandbox.

To sculpt a landform mesh, the designer wears a black glove and gestures under the camera. With the contrast between the black glove and the white background, the system can easily identify the user's gestures. There is no sensor or instrumentation embedded in the glove.

Raw images captured by the camera are transmitted continuously to the computer and passed on to Vision SDK 1.2, a low-level library of methods to acquire live images. These gesture images data are compared to known image templates to find appropriate matches and to issue commands to graphic rendering implemented in Microsoft Direct3D.

Figure 2 shows the two program windows on display. The large screen displays a simple three-dimensional scene for the modeling actions, and the

smaller gesture recognition window displays the image captured by the camera.

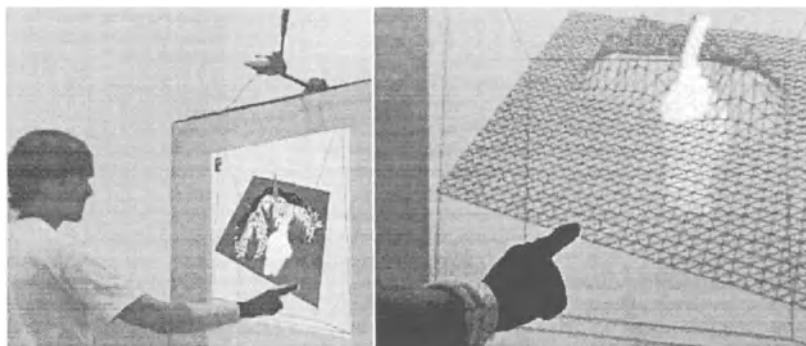


Figure 1. Designer interacting with the Digital Sandbox

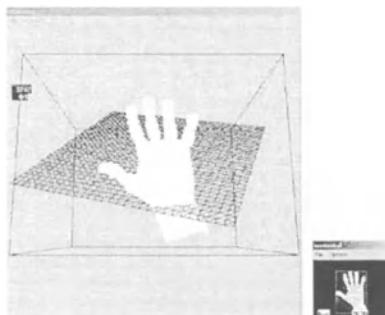


Figure 2. 3D modeling space (left) and gesture-recognition window (right)

The Digital Sandbox is written in C++. The system consists of several classes; each has a group of objects. These objects contain attributes and methods to operate upon these attributes. For example, the size, color, and shape are the attributes for an object. Object behavior is implemented through defined methods of operation such as changing the size, color, or shape under certain conditions. Each object in the system encapsulates attributes and behaviors defined by its classes. Each class thus functions independently of others. As is customary in object oriented systems, class inheritance enables each object to inherit properties from its parent classes and thus only class specific features need to be modified and defined. Thanks to this class hierarchy and individual object definitions, a module of

the program can be tested and expanded easily. At present the Digital Sandbox only has a water simulation function, but using this infrastructure, we could add other dynamic ecological models (e.g., light, wind, growth, and fire) in the future.

Currently there are two facets of the Digital Sandbox. Gesture Modeling deals with how mesh geometry is modeled with gesture commands. The SandboxToy supplies the attributes and behaviors for water accumulation and flow model simulation. Two general class categories were developed for each of these functions. The following sub-sections discuss these implementations.

### 3.2 IMPLEMENTATION OF GESTURE MODELING

#### *3.2.1 Initial Image Processing*

Input from the camera is a 160 by 120 bitmap image. An intensity level is set as threshold to isolate the glove from the background. Each pixel in the image is compared to the threshold value. When the intensity value is less than the threshold, it is replaced by a white pixel. Intensities above the threshold value are replaced with black pixels. The result of threshold operations is a black silhouette of the glove (hand) against a white background. The system then draws a bounding box to contain all areas of black pixels to distinguish the hand from the rest of the raw image captured by the camera.

#### *3.2.2. Mapping of Gestures to Commands*

Currently the system recognizes seven different gestures. These seven examples shown in Figure 3 capture the essence of generic gestures for 3D manipulations. They are point, thumbs up, two (v symbol), five (open palm), fist, gun, and pinch. These gestures suffice for most landscaping operations. Certainly more gestures can be added to the system. However, adding more items might increase the user's cognitive load (Hix and Hartson 1993) and thus reduce usability.

The system uses gesture commands to add, edit, move, and analyze three-dimensional geometry. The association of gestures with commands is the basis of the system. Each input gesture in the system is associated with a command to execute an action in the digital environment. Each gesture is mapped to an operational command as shown in Table 1. Designers can use these gestures to perform specific landscape forming tasks. For example, a "point" gesture can deform the mesh to sculpt a digital landform. When the "point" gesture is recognized, the system finds the coordinates of the index finger from the image, and identifies the intersection point. When the

designer's index finger moves up, this point acts as the control to deform the mesh and creates a peak or a hill.



Figure 3. Gestures recognized by the Digital Sandbox.

TABLE 1. The mapping of gestures to commands for the Digital Sandbox

| <b>Gesture</b> | <b>Command</b>                                               |
|----------------|--------------------------------------------------------------|
| Point          | Deforms the digital mesh terrain.                            |
| Thumbs up      | Activates the storm water accumulation simulation.           |
| Two            | Adds a building (a cube).                                    |
| Five           | Selects a building (when intersected with the hand gesture). |
| Fist           | Moves a building.                                            |
| Gun            | Adds a tree (a cone above a cylinder).                       |
| Pinch          | Selects and moves a tree.                                    |

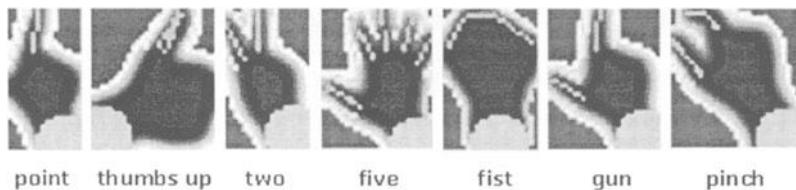
### 3.2.3 Pattern matching with template images

The initial image processing determines the location of the hand in the three-dimensional space. The position of the bounding box determines the x and y coordinates of the hand. The size of the bounding box determines the depth or z coordinate. The closer the hand to the camera, the larger its image becomes. After initial processing of raw images, the second part of the gesture recognition process performs pattern matching with known template images.

The system has a set of pre-stored template images for pattern recognition. Once the shape of the hand gesture is distinguished from the

background and contained with a bounding box, it is compared to a set of template images to find the closest match. Figure 4 shows some of the templates images stored by the system.

Each template is a spatial map of image features such as segments, edges and lines. The templates of the hand gestures are constructed (currently through manual editing) using images captured by the camera. To facilitate easy comparison, important spatial features are identified with different colors. The background white pixels (empty space) are replaced by red pixels. The black pixels indicating palm area are replaced by blue ones. The finger axes are drawn as green lines. These axes are further outlined with yellow lines to indicate finger boundaries. The wrist and arm areas are colored cyan. The region of gray pixels remains unmodified as buffer are for pattern-matching.



*Figure 4. A sample of template images for pattern recognition*

The pattern-matching process involves two classes: KnownImage and KnownImageSet. The template images are objects that belong to the KnownImage class. The class functions as a data structure and uses a data type from the VisionSDK library to store dimensions and spatial structure for each image. The KnownImageSet class compares the template images to input images from the camera in real time. The system performs image processing on a per-frame basis. To determine which template is the closest match, a similarity score is generated as each input image is compared to the templates. Matching on each image frame produces three best matching templates. The system then selects the gesture that matches the largest number of the last several frames' top-ranked templates.

Template matching includes three major processes. Each of these processes is a test to determine the next step of comparison. These test conditions are a set of if/then rules. First the system compares aspect ratio of the images. A template is considered an inappropriate match if the aspect ratio of the images differs by more than a constant amount. The template is discarded and comparison stops. The next template is then considered, starting the comparison process over. The second step compares spatial features. The red and blue pixels in the template are compared to

corresponding white and black pixels in the input image. Gray pixels are compared as well. If the percentages of each color composition are lower than a certain value, the template is not a match. Otherwise, the matching score is incremented by an amount proportional to the percentage of matching pixels. Finally, a similar process compares green and yellow pixels and checks against the template's eight nearest neighbors. The percentage of green pixels with black neighbors and yellow pixels with white neighbors are calculated. The score is then further incremented by an amount proportional to these percentages. The gray area serves as the buffer zone to provide possible match when there are variations in the two images.

The system proceeds sequentially through each stored template to form a list of potential matches in a ranked order. The system then selects the top-ranked gesture template of the last five frames and replaces the last one in the list with this new gesture. The most frequent gesture in the list is then considered to be the gesture for the current frame. This five-frame "buffer" approach is taken to provide stability to the system. By disregarding short-term (fewer than 5 frames) candidates, the system avoids a 'flicker' of wrong match fluctuations.

### 3.3 IMPLEMENTATION OF THE SANDBOX TOYS

A set of classes was developed to represent a traditional, physical sandbox. A mesh class represents the 'sand' in the sandbox. Landscape objects are toys made of by simple geometric primitives (e.g., cones, cylinders, and cubes).

There is also a separate class for the sandbox application. Each object has its own methods and interacts with each other to compute the storm water accumulation model. These toy objects are also embedded with attributes of their real-world counterparts (e.g., trees intercept water). Below we discuss these classes, their attributes and methods.

#### 3.3.1 *TheSandbox*

The top-level control structure of the system is *TheSandbox* class. *TheSandbox* class directs the user interactions and creation of new toys. The main components of *TheSandbox* are the mesh and the active toy objects in the sandbox. The attributes for this class includes (1) *Toy* - an array listing currently active *SandboxToys*, (2) *CurrentlyPlaying* – a variable indicating, if any of the toys in the sandbox are being manipulated, (3) *ActiveToy* – A pointer to the current active toy.

There are four methods for this class: (1) *CreateNewToy* – produces a *SandboxToy* object (i.e., a tree or a building). (2) *AddToyToSandbox* – adds the newly created toy to the *Toy* array. (3) *InteractWithToys* – directs

interaction, executes commands based on the recognized gestures to create, select and reposition toys in the sandbox, deform the terrain mesh, and activate the storm water accumulation simulation. (4) *IntroduceTheNeighbors* – finds each toy's position and its location on the mesh to decide and to modify the interception or infiltration of the mesh beneath the toy.

### 3.3.2 SandboxToy

The SandboxToy class is the parent class for Mesh, Tree and Building classes. Figure 5 shows that the SandboxToy class is the base (parent) for Mesh, Tree and Building. The Mesh contains MeshTriangles. The Tree contains TrunkType and CanopyType classes.

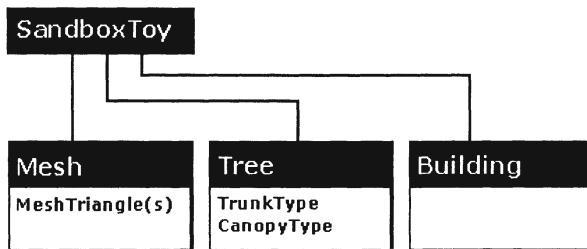


Figure 5. Inheritance relationship of the sandbox toys

The SandboxToy class has two attributes: (1) *ID*, to identify the toy type – a Mesh, Tree, or Building. (2) *PlayMode*, to signify the current status of the toy (active or not). The method for the SandboxToy is *Interact* – to direct toy interactions based on recognized gestures. Each derived class (Mesh, Tree, Building) has its own implementation of the method.

### 3.3.3 Mesh

The Mesh class makes up the digital sand in the sandbox. The Mesh constitutes the terrain surface. This mesh surface is the base for landform sculpting. It is therefore a permanent object that can not be created or destroyed by the user. The mesh is formed by a series of MeshTriangles. These MeshTriangles are stored and organized into an array of rows and columns. The Mesh class is also embedded with the storm water accumulation model so it can display itself according to the simulation results.

The Mesh deforms itself in response to gesture input, allowing the user to sculpt the surface. It also displays itself according to storm water

accumulation model when the user initiates the simulation command. Deformation of the Mesh is achieved by modifying the z-coordinates of its component triangles. This calculation is expressed through the Gaussian equation,

$$f(x) = (1 / (2\pi\sigma)^{1/2}) e^{-(x - \mu)^2 / 2\sigma^2} \quad (1)$$

where  $\mu$  is the control point where the finger touches the mesh, the midpoint of the progression, and  $\sigma$  is the variance. The  $\sigma$  value controls the shape of the deformation. Figure 6 shows the Gaussian equation distribution, and the  $\sigma$  value decides the appropriate number of points to be moved around  $\mu$ . Currently, the system has a hard-coded value of 0.2. The goal is to have the deforming curve slope gently. The value was derived from the trials that produce not excessively pointed or flat curves. It could also be a user set value.

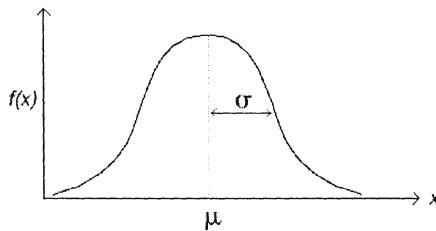
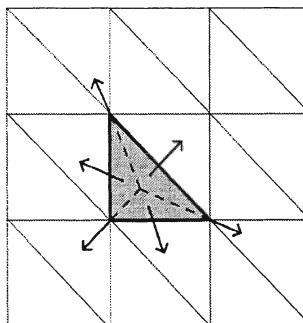


Figure 6. Mesh deformation is calculated according to Gaussian equation

The storm water accumulation model was designed to predict flow paths across the digital mesh surface of the sandbox. There are complex approaches for modeling storm water accumulation (e.g., Amer and Blais 2000) that route flow through structural features of the terrain (e.g., ridges, channels, and valleys). A simpler approach was adopted for the Sandbox. Each discrete mesh unit is treated as an independent component. Each mesh unit receives water from all sources, route flow across itself and passes water on to the neighboring surfaces. This approach is similar to the Triangulated Irregular Network approach (Silfer, et al. 1987). The model uses local rules and the topological relationships between adjacent triangles to route flow across the network of triangles. Two adjacent mesh triangles are joined at their edges. Boundary conditions for these triangles are thus relatively easy to specify. Therefore, to derive local rules in a triangulated model to specify the flow between triangles is easier than using a point-based grid Digital Elevation Model (Wilson and Gallant 2000).

Using the point as the component unit has one major problem for calculating the storm water accumulation model because the adjacent surfaces created from point matrices may not always be coplanar. These surfaces are also potentially more complex. In contrast, a surface derived from triangular planes guarantees adjacent triangulated units of the mesh will always be coplanar. Therefore, all possible boundary conditions for every triangle in the model are easier to handle. To route flow across the mesh the model only needs to define flow across an individual triangle.

Computing the storm water accumulation model involves several steps. First, all mesh triangles are sorted according to their maximum heights. The runoff calculation starts from the top – the triangle with the highest elevation. Next, the runoff is routed from the triangle to the nearest downhill neighbor. This is determined by calculating the orientation of the line of steepest descent. Flow is routed based on the orientation of this line, to one of six possible directions. Figure 7 shows these six directions intersect either the triangle's three adjacent faces or one of its three corners. The next procedure is to check whether this neighbor has lower elevation to the triangle. If it is lower than the current triangle, then the flow is routed to it. If not, the system finds the next lowest triangle along the line of steepest descent from a wider neighborhood. When a lower neighbor is found, flow is routed to it. If no lower neighbor is found, the triangle is regarded as a sink that absorbs the flow (instead of routing it). When neighboring triangles are facing each other, the slope line for flow routing is computed by averaging their aspects. The model proceeds step-wise to the next triangle. The procedure is repeated until all triangles in the sorted list are exhausted. These simple, local rules direct the routing of the flow across the entire mesh surface from the highest to the lowest.



*Figure 7.* Flow can be routed to any of the six possible directions in a triangle

### 3.3.4 MeshTriangle

The Mesh is subdivided into, and consists of MeshTriangle class objects. Each mesh triangle can perform several tasks. It can modify its vertices when the mesh is deformed. It calculates and updates its gradient, aspect, and elevation. It also computes its storm water runoff; displays its color and fill mode based on the runoff value. The equation for storm water runoff calculation is based on the following model (Burrough and McDonnell 1998),

$$\text{Runoff} = \text{Input} - \text{Interception} - \text{Infiltration} - \text{Evaporation} + \text{Runon} \quad (2)$$

where Runon is the runoff quantity routed to the triangle by its uphill neighbor. The values used by the model are currently hard-coded into the system. The system could easily provide a dialog box for the user to explore and adjust different numeric values for the model.

The fill mode and color for each mesh triangle is determined by its computed runoff value. If the runoff exceeds a threshold value the triangle is painted with a color. If the value is below the threshold, then the triangle remains in wire-frame mode and displays no color. Currently the threshold value is hard-coded into the system. The user could also specify this value when an input interface is provided. The color for the triangle is determined according to a gradient color scheme. Lower runoff values are represented with less saturated colors, while higher-saturation colors correspond to higher runoff values.

There are several important components for this MeshTriangle class. The attributes for this class includes (1) *Elevation*, the height of the triangle; (2) *Aspect*, orientation of the steepest descent line for the triangle; (3) *Gradient*, rate of elevation change across a triangle;. (4) *InputPrecipitation*, precipitation quantity; (5) *Evaporation*, water evaporation, not set for the current runoff model but would be useful for future module of atmospheric or ecological simulation model; (6) *Interception*, precipitation intercepted by objects located above the triangle; (7) *Infiltration*, amount of precipitation infiltration for the triangle; (8) *RunoffFromNeighbors*, runoff volume received from neighbors, and (9) *RunoffToNeighbors*, amount of runoff to be routed to a neighbor.

### 3.3.5 Tree

The Tree class contains a TrunkType and a CanopyType. It is a class derived from the SandboxToy class. The Tree class determines attributes of its trunk and canopy. It uses these attributes to intercept rainfall water. Currently there is only one tree type available for the Digital Sandbox. However, the current framework is extensible to accommodate multiple

types of trees. In the future implementation, each tree type could potentially interact differently with the terrain mesh by intercepting different quantities of rainfall water (just like the real trees they simulate). The attributes of this class include (1) *Trunk*, and (2) *Canopy*. The methods for this class include (1) *Interact*, move position according to recognized gesture, and (2) *SnaptToGrid*, that ensure the tree is always positioned on the surface of a mesh triangle instead of floating in space. The TrunkType class and CanopyType class both maintain their dimensions and can be moved by the user. Future extension could contain different methods and attributes to specify different water interception behaviors of the trunk and canopy types.

### 3.3.6 Building

The Building class is also a descendant of the SandboxToy. Buildings are typically impermeable to rainfall. Therefore, when a Building object is positioned on a mesh triangle, it decreases the infiltration values of mesh triangles located downhill. Like Tree, Building also uses methods to maintain its own dimensions and location in the sandbox. The current *Interact* method for Building is more complex than the Tree's method. Two gestures are involved to manipulate the building. The gesture "five" is used to select and deselect a building, and "fist" grabs the building and moves the building around. Therefore, the *Interact* method responds to a sequence of recent gestures (instead of one) to determine the current play mode status. For example, the user may be selecting the building ("five" followed by "fist"), releasing the building ("fist" followed by "five"), or moving the building ("fist").

## 4. Interacting with the Digital Sandbox

The goal of the Digital Sandbox project is to create a new type of tool to facilitate a more straightforward and direct process for landscape design. The Digital Sandbox project takes advantage of image processing for gesture recognition, and object oriented programming for embedding object behaviors and dynamic simulations. The idea is to enable designers to use simple freehand gestures to create and edit a digital landform. It is also intended to bridge the separation between design and analysis tasks. The Digital Sandbox system provides the ability for a designer to sculpt a landform, activate a storm water accumulation model simulation, and display the result on the landform, all within the same work environment.

Let's come back to consider the design scenario described earlier. The following paragraphs describe the sequence of interactions in a design process within the Digital Sandbox.

A landscape architect might begin a design session wearing the black glove and gesturing in the input space under the desktop camera. The system provides a flat mesh at ground zero for sculpting interaction. The model space on the screen displays the gestures captured by the camera. Figure 8 shows the designer deforming the mesh terrain by moving a hand with index finger extended. Unlike normal CAD applications that require the input of contour lines before any terrain can be formed, simple gestures are enough to sculpt the mesh in the Digital Sandbox.

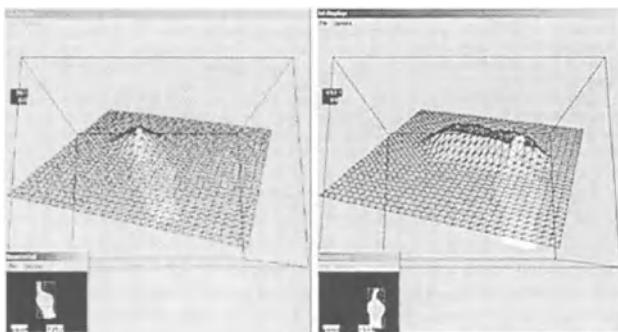
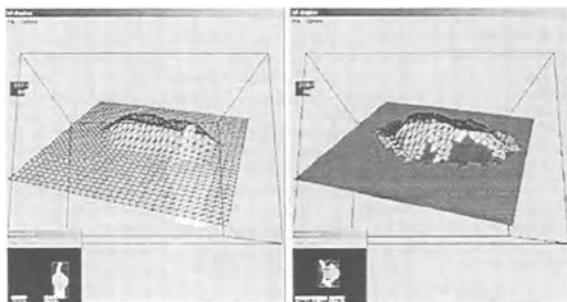


Figure 8. The “point” gesture sculpts and deforms the digital mesh. Left: initial sculpting. Right: a hill is generated after continuous “point” command.

A designer can create peaks and hills by pointing and deforming the digital mesh model. This process is very intuitive. Informal user feedback shows that the majority of users (designers, high school students, etc) can walk up to and use the point gesture to sculpt landform without formal training. The designer can analyze the landform design at any point in the design process. The simulation of storm water flow on the site can be activated by simply makes a “thumbs up” gesture. Figure 9 shows the “point” gesture being used to sculpt the landform, and the “thumbs up” gesture initiates the storm water accumulation model. The system then computes the elevations and orientations of each mesh triangle and routes flow across the terrain surface. The accumulation of water is displayed by a gradient of colors on the mesh triangles. More highly saturated color indicates higher runoff volume. The less saturated colors represent lower runoff values. The white mesh triangles indicate the regions that do not exceed the runoff value threshold.

The system painted all mesh triangles whose runoff volume exceeds a certain threshold value (indicative of erosion danger). The colored mesh triangles indicate the locations for flow paths. This means the simulation

result is displayed directly on the landform, in the design environment. Each triangle is painted according to the water accumulation value it derives. Having analysis displayed on the terrain mesh enables the designer to visualize the flow and relative concentration areas. With this information the designer can make better assessment about the design impact on the ecological system of the landscape.



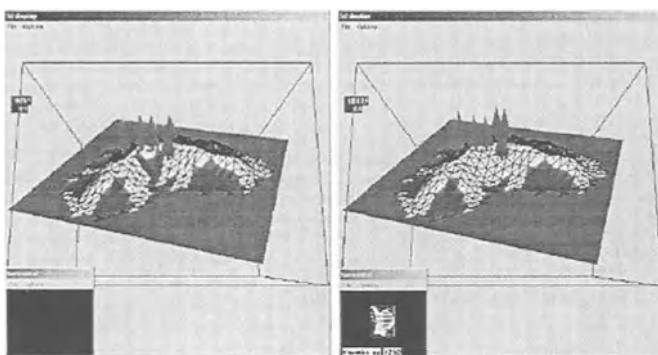
*Figure 9.* The “thumbs up” gesture initiates the storm water accumulation model. Left: original mesh model being sculpted. Right: mesh with painted color showing storm water values resulting from the water accumulation analysis.

To redirect the storm water flow, the designer can continue to sculpt the mesh. The iterative cycle of adjusting the landform and analyzing flow paths can be carried out repeatedly. The designer can also place some trees to intercept the rainfall without changing the landform. To change the concentration of water on various parts of the site, the designer can add trees to the landform uphill from the areas of concern. Figure 10 shows the “gun” gesture being used to add a single tree to the landform, and the “pinch” gesture being used to move and reposition the tree to a new location.



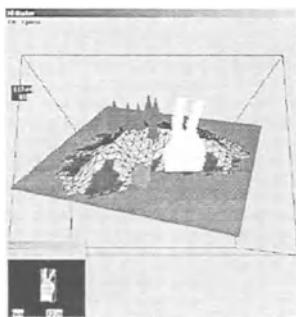
*Figure 10.* Creating, selecting and moving a tree. Left: the designer uses the “gun” gesture to create a tree. Middle: the “pinch” gesture grabs and moves the tree. Right: the tree is released to new location when no gesture is recognized.

To create additional trees, the designer can repeat this process. The designer can run the storm water simulation model again after placing the trees on the site. Each tree intercepts rainfall, and thus decreases the storm water accumulation on the areas immediately below it. Figure 11 shows that a cluster of trees in a particular area would significantly decrease surface flow in the area below.



*Figure 11.* More trees decrease storm water accumulation on the area immediately below. Left: the model with trees. Right: tree clusters cause the disappearance of colored triangles immediately below.

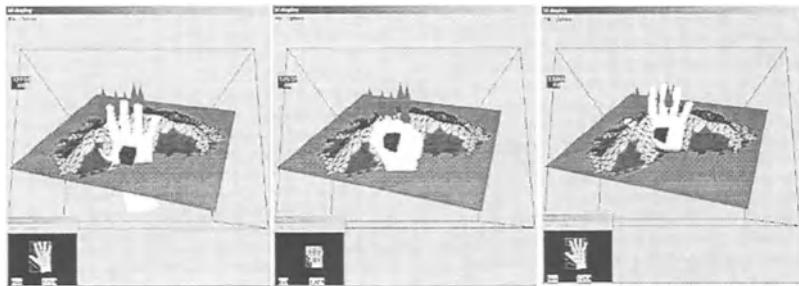
To add buildings to the sandbox terrain, the designer uses a similar method. Figure 12 shows the gesture “two” being used to create a simple cube shaped building.



*Figure 12.* The “two” gesture creates a building

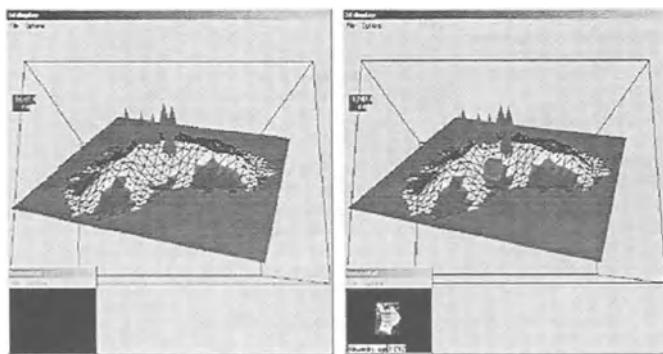
Figure 13 shows the process of moving a building. The designer uses a combination of “five” and “fist” gestures to move the building. First, the

“five” gesture is used to select the building. When the designer’s hand intersects with the building it changes color to signify its position and selection status. A quickly formed “fist” gesture grabs the building. Once the building is grabbed, it follows the hand, and can be moved to a new location on the landform. To release the building, the designer makes the gesture “five” to deselect the building.



*Figure 13.* Left: the “five” gesture selects the building. Middle: the “fist” gesture grabs and moves the building. Right: a “five” gesture after “fist” released the building to new position.

Figure 14 shows the storm water simulation showing (before and after adding a building) how the building affects the runoff values of the area immediately downhill. Buildings are impermeable to rain fall. Therefore, the existence of a building would increase storm water accumulation in the area immediately below it.



*Figure 14.* A building increases storm water accumulation on the area immediately below. Left: water runoff on the terrain. Right: more colored mesh appears immediately below the building indicating more water accumulation value.

The above scenario illustrates how the Digital Sandbox supports integration of design and analysis tasks in an environment for digital landscape forming. By providing the geometric sculpting and simulation analysis in a single tool, the iterative processes of design and analysis are better supported. The system provides designers the ability to use hand gestures to sculpt terrain in space, create objects like trees and buildings and add them to the landform, and activate a storm water accumulation model to see the design impact.

## 5. Discussion and Future Work

In sum, the Digital Sandbox is motivated by the need to integrate form making and ecological functions in the same environment. Current landscape design practice employs different tools for different tasks. These tools are useful but they do not support a seamless transition between design and analysis tasks. Digital Sandbox enables the designer to employ hand gestures to design a landform and to analyze the storm water flow anytime during the design process.

The inspiration for the design of the Digital Sandbox comes from the physical sandbox. The traditional sandbox provides an intuitive environment in which to play and explore landform design. The hands-on experience with sand helps people, children and adults alike, understand the nature and the structure of landscape forming. A computational environment ought to have similar characteristics in order to provide better human computer interactions.

The Digital Sandbox system uses an object-oriented framework for implementation, and to accommodate possible future extension. A set of object classes has been developed to represent real-life objects such as trees and buildings. Each class is embedded with behaviors and attributes to keep track of its interactions with others, current location, computation and display of the storm water accumulation model. Designers can manipulate these objects and issue commands using hand gestures in three-dimensional space. Each gesture is mapped to one particular command. This gesture-command association enables interaction with landform without having to stop to type or find a pull down menu. The designer can sculpt a digital landform, add trees and buildings, and run the storm water accumulation model to display the environmental impact to the landform. The results of the storm water flow analysis are painted directly on the landform mesh triangles. This instant visual feedback on the same environment could empower the designer to make more informed decisions without having to

translate data to different platforms or to change the focus to a different task.

Researchers in artificial intelligence have argued over two different research methodologies (Boden 1995; Schank and Childers 1984). Research projects are classified as focusing on advancement of either intelligent applications, or theories of human mind. The Digital Sandbox project addresses the task-oriented approach of AI research. It is a task-directed technological approach. We are concerned with the technology of getting computers to do things that people already do. It responds to demands of a particular task. Designers regularly perform complex tasks such as design and analysis interchangeably in a design session.

There are several research directions for future work. One obvious next step is to add more simulation models and the ability to import existing data sources, such as Digital Elevation Models into the Digital Sandbox system. The goal would be to use the tool on an actual site with existing topography. We would like to add the capabilities of adding multiple trees and buildings as well as different types of landscape objects that would respond differently to various ecological models, for example, trees with different water infiltration rates or roadwork that provide different permeability and drainage functions. It would also be important to make the interactions more realistic and practical. For example, the landform model can be embedded with physical constraints such as slope limits or environmentally sensitive area boundaries. Connection to a spatial data engine such as a conventional Geographic Information System would allow more parametric manipulations of the behaviors and attributes of the sandbox toys. We could investigate making the modeling space displayed in three-dimensional holographic space instead of the 2D screen. Adding more cameras as inputs could potentially provide more operational opportunities and help calibrate image recognition. For example, with the ability to capture the hand gestures in both plan and profile view could enable the calculation of roll, pitch, and yaw for greater degrees of freedom in manipulating the model. However, we still wish to keep the interface simple and inexpensive (not using an instrumented glove or head mounted display). Another possible direction is to embed physical media with the intelligence of a Digital Sandbox. For example, tactile feedback could be conveyed through physical sand that is tangible while the simulation model is displayed onto the physical terrain model to provide analysis feedback.

We are exploring building a gesture-command mapping module for users to input new gesture templates and to associate them with intended operational commands. For example, a designer could give examples to train the system to recognize new gesture templates. Different designers might

create their personal preference for gestures and commands. Wearing the black glove is a minor inconvenience, though it is more economical and less cumbersome than the data glove and head mounted display set-up that other research projects require. It would also be useful to develop more complex image-processing algorithms capable of detecting the user's hand without the user needing to wear a glove to ensure contrast to the background.

## Acknowledgements

A more complete description of the Digital Sandbox project can be found in Robert M. Harris's master of landscape architecture thesis. This research is based upon work supported in part by the National Science Foundation under Grant No. IIS-00-96138. The views contained in this material are those of the author and do not necessarily reflect the views of the National Science Foundation.

## References

- Amer, A and Blais, J: 2000, Object-oriented digital terrain modeling for geomatics applications, *Geomatica* **54**(1): 43-52.
- Boden, MA: 1995, AI's half-century, *AI Magazine* **16**(4): 96-99.
- Burrough, P and McDonnell, R: 1998, *Principles of Geographical Information Systems*, Oxford University Press, Oxford.
- Danahy, J and Wright, R: 1988, Exploring design through 3-dimensional simulations, *Landscape Architecture* **78**(4): 64-71.
- DaylonGraphics: 2001, <http://www.daylongraphics.com/products/leveler>.
- DMG, 2001, *Design Machine Group*, <http://dmg.caup.washington.edu>, Design Machine Group, University of Washington
- Ervin, S., 1992, *Integrating Visual and Environmental Analyses in Site Planning and Design*, GIS World, 26-30.
- Gross, MD and Kemp, A: 2001, Gesture modeling: using video to capture freehand modeling commands, in B deVries, JP vanLeeuwen and HH Achten (eds), *CAAD Futures 2001*. Kluwer, Eindhoven, pp 33-46.
- Harris, RM: 2001, *The Digital Sandbox: Integrating Design and Analysis in a New Digital Earth-forming Tool*, Master Thesis, Landscape Architecture, University of Washington, 71.
- Hix, D and Hartson, HR: 1993, *Developing User Interfaces - Ensuring Usability Through Product & Process*, John Wiley & Sons, New York.
- Kruger, W and Frohlich, B: 1994, The responsive workbench, *IEEE Computer Graphics and Applications May*, 12-15.
- Mayall, K, Hall, G and Seeböhm, T: 1994, Integrate GIS and CAD to visualize landscape change, *GIS World* **7**(9): 46-49.
- Nishino, H, Utsumiya, K, Yoshioka, K and Korida, K: 1997, Interactive two-handed gesture interface in 3D virtual environments, *ACM Symposium on Virtual Reality Software and Technology*, pp. 1-8.
- Sawyer, C., 1998, Representing landscapes digitally, *Landscape Australia* **20**(2): 128-132.

- Schank, RC and Childers, PG: 1984, *The Cognitive Computer - On Language, Learning, and Artificial Intelligence*, Addison-Wesley, Sydney.
- Schkolne, S, Pruitt M and Schroder, P: 2001, Surface drawing: creating organic 3d shapes with the hand and tangible tools, in J Jacko, A Sears, M eaudouin-Lafon and R Jacob (eds), *Conference on Human Factors in Computing Systems*, ACM, New York: 261-268.
- Segen, J and Kumar, S: 1998, GestureVR: Vision-based 3D hand interface for spatial interaction, in W Effelsberg and B Smith (eds), *Sixth ACM International Conference on Multimedia*, ACM, New York, pp. 455-464.
- Silfer, A, Kinn G and Hassett, J: 1987, A geographic information system utilizing the triangulated irregular network as a basis for hydrologic modeling, in N Chrismas (ed.), *Auto-Carto 8: Proceedings of the Eight International Symposium on Computer-Assisted Cartography*, American Society for Photogrammetry and Remote Sensing, Falls Church, VA, pp. 129-136.
- Uusitalo, J, Orland B and Liu, K: 1997, A forest visualization interface for harvest planning, *ACSM/ASPRS Annual Convention and Exposition*, American Society for Photogrammetry and Remote Sensing, Bethesda, MD **4**: 216-223.
- Westort, C: 1996, Sculpting DTMs: an example tool, *Anthos* **35**(1): 42-45.
- Westort, C: 1998, *Methods for Sculpting Digital Topographic Surfaces*, Ph.D. Thesis, University of Zurich.
- Wexelblat, A: 1995, An approach to natural gesture in virtual environments, *ACM Transactions on Human-Computer Interaction* **2**(3): 179-200.
- Wilson, J and Gallant, J: 2000, Digital terrain analysis, in J. P. Wilson and J. C. Gallant (eds), *Terrain Analysis: Principles and Applications*, John Wiley, New York, pp. 1-28.

## **CONCEPTUAL KNOWLEDGE IN DESIGN**

---

*MMFORTED: A cognitive tool fostering the acquisition of conceptual knowledge about design products*

Elio Toppano

*From concept to embodiment: challenge and strategy*

Zhi Gang Xu, Ming Xi Tang and John Hamilton Frazer

*Artificial intelligence for the design and grading of precious stones*

Tony Holden and Matee Serearuno

## MMFORTED: A COGNITIVE TOOL FOSTERING THE ACQUISITION OF CONCEPTUAL KNOWLEDGE ABOUT DESIGN PRODUCTS

ELIO TOPPANO  
*Università di Udine*  
*Italy*

**Abstract**. The work described in this paper deals with conceptual understanding of design products. A working hypothesis is that the comprehension of something by someone is strongly related to the capability of the subject to build and use multiple representations of the thing under consideration and to explicitly consider their characteristics and their reciprocal dependencies in explanation and problem solving. A cognitive tool called MMforTED has been developed to foster the acquisition of several conceptualisations (ontologies) that can be used to represent and inquiry about artefacts from different perspectives. The system exploits an instructional environment implemented by an electronic hypertext, that enables users to browse through a network of models of several devices. Browsing through the network of models is accompanied by a simultaneous change of perspective. Because of the particular organisation of domain knowledge we have adopted, this amounts to exercising a well defined knowledge transmutation (e.g. conceptual abstraction, generalisation, reduction, approximation and aggregation or their inverses). The ability to explore a situation from different conceptual perspectives is considered fundamental for problem setting and design development.

### 1. Introduction

This paper is concerned with conceptual understanding in design. We assume that the core of understanding is an hermeneutic activity of constructing interpretations (Gadamer 1976). Circles or spirals of understanding arise in design problem setting as well as in solution development. As reported in (Protzen et al. 2000) illustrating the work of Rittel and Schön:

"When we set the problem we select what we will treat as the 'things' of the situation, we set the boundaries of our attention to it, and we impose upon it a

coherence which allows us to say what is wrong and in which directions the situation needs to be changed"

Rather than saying that the designer is faced with an implicit problem formulation that must only be discovered, both Rittel and Schön, recognise the designer's freedom to choose what problem is being solved. The designer's possibilities for choosing the problem are essentially limited only by her conceptual system and imagination. Having a rich repertoire of conceptualisations enables a designer to select what to treat as the "things" and "relations" in a situation. This may be done in a multiplicity of alternative ways thus enhancing her ability to make sense of the state of affairs.

Conceptual systems are fundamental also during design development. Fujii and Gero (2000) introducing the situated agent approach write:

"A situation is not a direct projection of an environment, but is perceived on the basis of a part of internal state of an agent, which is formed through the history of situations and transactions, and an environment. The authors call the part of the internal state concepts "

From the point of view of hermeneutics, understanding a design situation means projecting the new situation within the context of existing knowledge structures (the bootstrapping component), building a higher order component by establishing connections between the new situation and existing knowledge and using that component to reconstruct or replace background knowledge.

The need for computational tools supporting understanding and interpretation has been stressed by various authors. Protzen et al. (2000) write:

"Instead of computational problem solving attention should be given to the power of computational tools as communicative devices: devices that aid in the development of new and different understanding of problematic situations. It is this potential of computers to enhance our own understanding that needs to be explored"

In our opinion there are, at least, two main approaches for developing such kind of systems. The first approach is to build systems that substitute the human in making the interpretation. This approach has been followed, for example, by (Haymaker et al. 2000). The paper illustrates the use of filters that provide the designer with a set of alternative views of a design solution. The interpretations are constructed from a database of geometrical and topological data. Interpretation is seen as the process of deriving attributes of the design not explicitly stated in the description (Coyne et al. 1990). Another approach is to build cognitive tools that help the user acquiring the basic conceptualisations and relative knowledge transmutations that can be used by herself to make the interpretation. To this end, some authors, such as Laurillard (1993) stresses the importance of

systems that support tutorial dialogues; others, see for example, Ohlsson (1993) suggests the analysis of epistemic activities (arguing, describing, explaining, predicting, etc.) which are considered more relevant for conceptual understanding than the study of goal oriented action or procedural skills.

The work described in this paper follows this second approach. The instructional system we are going to describe is aimed at fostering the acquisition of several conceptualisations that can be used to describe, explain and understand design products i.e. artefacts. The system, called MMforTED - an acronym for MultiModelling for Technical Education -, has been conceived for introductory courses on technical education and design theory in the secondary school (age range 14-18). It exploits a learning environment, implemented as an hypermedia, constituted by a collection of models of simple devices. Besides showing models of artefacts the system provides a scaffold of questions and ontologies upon which students can construct their interpretations of a design product.

In recent years several research efforts have been spent to address the issue of learning in design. However, most of these approaches are focused on machine learning in design not design instruction and education (Sim and Duffy 1998). Notable exceptions are the research work described in (Ball et al. 1998) aimed at supporting the earlier generation of concepts in conceptual design and the work illustrated in (Heylinghen and Verstijnen 2000) which explores the effects of using cases in architectural education.

Our system does not directly support any design activity, but is focused on the acquisition of the conceptual knowledge that is propedeutic to other design activities. Our position is that students need more than just the domain to learn about it. Apart from access to domain information (e.g. cases) they need assistance in selecting and interpreting this information in order to build their knowledge bases.

To this end we introduced the notion of perspective. Many authors use the terms "perspective" and "multiple perspectives" but with several meanings. For some authors, such as (Mackellar and Peckham 1998), for example, a perspective is "a specialised representation which is defined as a partition of the global schema that supports some functionally oriented tasks"; for other authors (Rosenman and Gero 1996) a perspective is a kind of physical or domain view such as mechanics, electronics, etc. For (Perrussel et al. 1997) a perspective is "the representation made in a certain language of the domain observed by a viewpoint". In our approach a perspective represents a point in a multidimensional space of possible conceptualisations spanned by a set of specific modelling dimensions (i.e. conceptual abstractness, detail, resolution, coverage, generality). This notion

is based on a representational framework for modelling artefacts that leverages on the existence of a strictly layered organisation of conceptual knowledge.

The organisation of conceptual knowledge guides both the design of the educational material (e.g. artefacts models, questions, ontologies, and exercises) and the learning that subsequently takes place. Questions, for example, are categorised according to the conceptual content that could satisfy them as a response into structural, behavioral, functional and teleological questions. Such conceptual structuring of the information space aides the users in creating a mental model of the learning domain and acquiring the appropriate conceptualisations upon which they can construct their own interpretations of a design product.

As far as ontology is concerned, there are several ontological proposal in design: starting from the purpose-function-behaviour-structure ontology by (Rosenman and Gero 1996) to more recent approaches such as the activity/space ontology (Simoff and Maher 1998), the CPDM ontology (Ball et al. 1998), the structure/behaviour/performance (Varejao et al. 2000), among the others. Our choice builds on the Multimodelling approach (Chittaro et al. 1993). The rational is that this is a general and theoretical founded framework for organising and using many diverse models of an artefact. Moreover, it provides a clear definition of the concept of function and teleology and of their relations to structure and behaviour (Chittaro et al. 1994). The explicit representation of the conceptualisations used to build models makes the MMforTED system "ontology aware" that is capable to communicate with users in terms of the foundamental concepts and domain theories of which its knowledge is composed.

The paper is organised as follows. Section 2 is devoted to describe the theoretical framework we have followed to represent design products from multiple perspectives. In Section 3 we shall use such a framework to illustrate the instructional environment that has been developed to support conceptual understanding. Section 4 discusses the experimental activity that has been done with the system. Finally, Section 5 draws conclusions.

## 2. Theoretical Framework

In this paper, designing is viewed as a situated modelling activity. During this activity, the designer performs design operations that create, elaborate, modify and reformulate one or more artefact models until a satisfying solution is obtained. The following sections illustrate how artefacts are represented by the artefact model. The artefact model consists of several related models each one describing the artefact from a specific conceptual perspective. Given the fundamental role that the concepts of model and

model based communication have in the proposed approach we start by first reviewing these basic notions.

## 2.1 THE CONCEPTS OF MODEL AND MODEL BASED COMMUNICATION

We assume here that a model is basically a device that is built to answer specific questions about some portion of reality. A symbolic model of an artefact is a description with the following properties:

- it is constituted by a set of *assertions* referring to the considered artefact;
- assertions describe the artefact in terms of: *entities*, *properties* of entities, and *relations* among entities;
- assertions are expressed in some *language* that has a well defined syntax and semantics.

Notice, that for an object under investigation there is not "the model" to represent it but a set of models representing it from different points of view (i.e. using different types of entities, properties, and relations) or using different languages according to the observer's background knowledge and the kind of question or problem to be tackled.

Models can be used for problem solving or communication. This paper is mainly concerned with computer mediated communication of artefact models. We shall adopt a point of view that is strongly inspired to the FRISCO framework (Falkenberg et al. 1998). This point of view can be summarised as follows. In communication, a subject (the emitter) generates a message (a model) that represents some knowledge about an artefact expressed in a language. The message is transmitted via a channel (a medium such as a computer system) to another subject (the receiver) who interprets the message and constructs a personal conception of its content. Information is the personal knowledge increment of the receiver in interpreting the message. For FRISCO, information and communication are not absolute but relative concepts. They are seen as linking the individual person ("information" as increase of personal knowledge) and the larger community of which that person is a member (shared knowledge resulting from communication).

Two main processes are involved in communication namely, interpretation and modelling. During *interpretation* the receiver perceives the message with her senses and forms a specific pattern of visual, auditory, or other sensations in her mind. These percepts are then elaborated by various cognitive processes such as categorisation, inference, imagination, etc. in order to form a mental conception. During *modelling* the emitter selects the content of a model that is, the aspects of a mental conception that

are deemed relevant to answer the question or solve the problem of interest (articulation), and represents the content in a language (externalisation).

Interpretation and modelling are driven by the subject's conceptual system. The term conceptual system is intended here to refer to the collection of relatively stable conceptions (e.g. conceptual categories, cognitive models) formed in a person mind during her experience and interaction with the physical and socio-cultural environment in which she lives. In order for the process of communication to be effective it is necessary that the two partners in the communication process share a body of linguistic and conceptual knowledge about the domain of discourse that is they have to commit to an ontology.

## 2.2 ONTOLOGIES FOR REASONING ABOUT ARTEFACTS

In Artificial Intelligence, ontology is defined as "an explicit representation of a conceptualisation" (Gruber 1993). It is worth stressing the fact that an ontology is not only a specification of a conceptualisation but embodies an agreement about that conceptualisation. It helps people belonging to a community of practice or interest to identify what they agree on and what they don't about the domain of interest. Such agreement facilitates accurate and effective communication of meaning which, in turn, leads to other benefits such as inter-operativity, reuse and sharing (Uschold 1998).

Ontology is typically composed of two parts, that is, the conceptual-level ontology and the lexical level ontology. Lexical level ontology provides a human-friendly vocabulary of terms used by the authors of the ontology to describe the domain of interest. Conceptual level ontology specifies the detailed meaning of each concept, the relationships existing between concepts (e.g. taxonomic relations) and a set of semantic constraints (i.e. axioms).

The main goal of our work was to identify a core set of terms used to describe design products and solutions i.e. artefacts and to, as accurately as possible, give definitions to these terms which reflect the most common usage as found in teaching practices and in literature. Considerable care has been devoted to the following issues:

- *global consistency*: we tried to develop one coherent framework where every concept is related to every other one in a specific well-established way, such that a consistent whole exists;
- *generality*: we did not attempt to consider specialised subfields of technical education such as electronics, mechanics, hydraulics, and so forth. Our framework was meant to be as general as possible but specialisable whenever necessary to cater the various subfields;

- *minimality*: the resulting framework had to be as simple as possible. We searched for a kernel collection of fundamental or essential concepts;
- *foundation*: we tried to use as much as possible and whenever appropriate well established concepts from relevant disciplines. For technical education we have anchored concepts to studies and research in the field of model based reasoning - mainly, Qualitative Reasoning and Functional Reasoning - and Design Theory.

In building the ontology we adopted an incremental design process keeping teachers of technical education in the loop. Teacher participation in ontology design has greatly enhanced practicality of the framework and its relevance to instruction. Ontological proposals vary along the formality dimension from highly informal to rigorously formal ontologies. The degree of formality required depends upon the intended purpose of the ontology. Since our purpose is communication an unambiguous but structured informal (i.e. expressed in a restricted and structured form of natural language) has been considered sufficient. For the core terms of the ontology we use a definition template proposed by (Uschold 1998). The template specifies:

- Term: the word or sequence of words which stands for the concept;
- Short definition: a text which briefly defines the concept referred to by the term;
- Elaboration: an elaboration of the short definition containing pertinent background, motivations, technical details helpful in gaining full understanding of the concept;
- Examples: one or more examples illustrating the concept and/or how it is used;
- Variations: synonyms;
- Related terms: a list of terms denoting closely related concepts to the term being defined together with the nature of the relationship between the different terms.

As an instance, Figure 1 illustrates the definition of the concept "TERMINAL".

### 2.3 THE INTERNAL STRUCTURE OF THE ONTOLOGY: EPISTEMOLOGICAL TYPES AND CONCEPTUAL HIERARCHIES

The proposed ontology is based on previous research on modelling physical systems (Chittaro et al. 1993). According to this work, conceptual knowledge about artefacts have been partitioned into four categories called epistemological types. For each category a kernel collection of concepts and related terms together with a set of relevant *prototypical questions* and

*problems* have been introduced. A brief description of epistemological types follows.

1. *Structural knowledge.* This type of knowledge describes which components constitute the artefact and how they are connected to each other (i.e. their adjacency). The basic ontology includes the concepts of "component", "terminal" and "connection". Structural knowledge can be used to represent the connectivity of a system;

**Term:** TERMINAL

**Short definition:** a TERMINAL is a passive channel supporting possible interactions with the outside environment

**Elaboration:** a terminal supports just one kind of physical interaction which identifies its type

**Example(s):** thermal terminal, electrical terminal, mechanical terminal

**Variations (synonyms):** port, interface

**Related terms/concepts:** (type\_of) structural concept, (part\_of) component, (part\_of) connection

*Figure 1.* An example of concept definition

2. *Behavioral knowledge.* This type of knowledge describes how components can work and interact in terms of the physical quantities that characterise their state and the physical laws that rule their operation. The basic ontology includes the concepts of "mode of operation", "physical quantity", "physical law", "behavioral state" and "trajectory" of states. Behavioral knowledge can be used for behavioral prediction, causal dependency analysis and sensitivity analysis.
3. *Teleological knowledge.* This type of knowledge describes the goals assigned to the artefact by its designer and the operational conditions which allow their achievement through correct operation. The basic ontology includes the concepts of "goal", "operational conditions", "use", "expected behavior". Teleological knowledge can be used for interpretation of actual use and definition of proper use.
4. *Functional knowledge* This type of knowledge describes the contribution of individual component behaviors to the realisation of the ultimate goals of the artefact. The concept of function is thus understood as a bridge between behavioral and teleological knowledge. For the class of artefacts whose behavior can be interpreted in terms of flow structures of generalised substances (e.g. material, energy, power, information) the bridge can be represented at two different levels of abstraction:

*Level 1:* at this level the basic ontology includes the concepts of "generalised substance", "generalised current", "functional role" and

"functional role network". Generalised substances represent the abstract entities that flow through a system while a current is the amount of generalised substance that flows through a unit surface in a time unit. The functional role of a component is an interpretation of its behavior - more precisely of the physical equations governing its behavior, aimed at characterising how the component contributes to the realisation of the flow structure in which it takes part. Examples of functional roles are: the conduit, the barrier, the reservoir, and the generator. A component may play different roles in different domains: in a flat-iron, for example, a resistor is a conduit of current in the electrical domain and a generator of heat in the thermal domain. It should be stressed that the association of a functional role to a component is done in a principled way by exploiting formal analogies between laws belonging to different physical domains (Chittaro et al. 1994). Two types of relations between functional roles have been identified, namely, mutual dependency and influence. These are used to represent functional role networks.

*Level 2:* at this level the basic ontology includes the concepts of "cofunction", "process", "phenomenon", and "functional organisation". Specific configurations of roles (called cofunctions) enable the occurrence of elementary processes such as transporting, reservoir charging, reservoir discharging and blocking. Elementary processes can be related together by specific relations such as direct causation, regulation or support to generate phenomena. The network of processes (and phenomena) specifies the functional organisation of the artefact.

Functional knowledge can be used for functional prediction, functional dependency analysis and process detection.

A critical issue in ontology design is the internal concept structure (Noy and Hafner 1997). Rather than having a single tree-like concept hierarchy the conceptual content of each epistemological type has been organised into a number of small local taxonomies that are linked together by relations or axioms. Taxonomies include *type\_of* hierarchies, *part\_of* hierarchies and *precedence* hierarchies namely, *rank* hierarchies and *measure* hierarchies (Hieb and Michalski 1993). As an instance, Figure 2 shows three examples of hierarchies belonging to behavioral knowledge: TH1 is a typological hierarchy of variables; TH2 is a typological hierarchy of equations and MH1 is a measure hierarchy of quantity values.

Conceptual hierarchies let us identify a set of five independent dimensions for conceptual variation:

- *abstractness*: i.e. position of a conceptualisation in a rank hierarchy of epistemological types. The epistemological abstractness of a concept is a measure of the distance of this concept from the immediate experience within some theoretical framework (Marjomaa 1997). We consider the

following ordering of abstractness: structural concepts are more concrete than behavioural concepts which, in turn, are more concrete than functional and teleological ones;

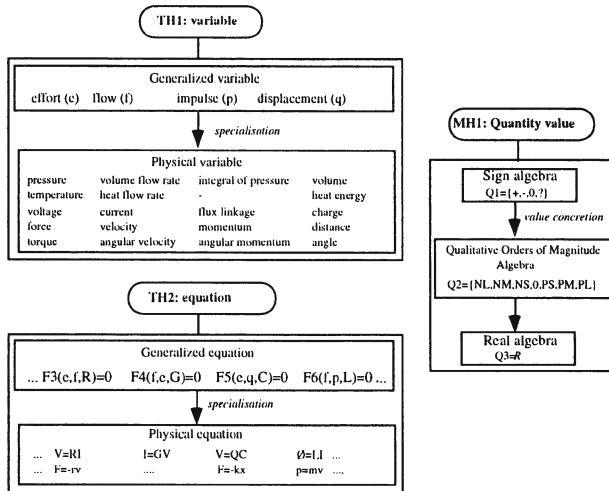


Figure 2. Examples of conceptual hierarchies

- *generality*: i.e. level in a typological hierarchy of the conceptual entities and relations used to describe reality. For example, the type of entity "component" used within a structural model is more general than "electrical component" which, in turn, is more general than "resistor";
- *detail*: i.e. degree of granularity of the knowledge represented by a conceptualisation. For example, the structural model of an electronic device may include conceptual entities at the level of major subsystems (e.g. the concepts of "filter", "amplifier") or can be further refined at the level of elementary entities (e.g. the concepts of "resistor", "capacitor", "diode");
- *phenomenic coverage*: i.e. the range of phenomena taken into account by a conceptualisation and the kind of simplifying assumptions that it presupposes. For example, in order to represent the behaviour of an electrical circuit we can use the concepts of "voltage", "current", "resistance", the Ohm's law and the Kirhoff principles. Using these concepts it is possible to represent electrical conduction. However, the above conceptualisation prevents us to describe the effect of magnetic induction of the current flowing through a wire. Moreover, a physical phenomenon can be represented by means or more or less idealised laws. As an example, in representing electrical conduction in wires we

can use the Ohm's law (i.e.  $V=RI$ ) or we can abstract away the resistance  $R$  and use the idealised law (i.e.  $V=0$ );

- *resolution*: i.e. number of distinctions allowed by the domains of values associated to the attributes of entities and relations of a conceptualisation. For example, the resolution of a quantitative behavioural model can be lowered either by relaxing real valued variables and using qualitative domains of values such as the set  $D=\{\text{negative, 0, positive}\}$  or by representing precise functional relationships by qualitative direct or indirect proportionalities (MQ&D Group 1995).

Dimensions are used for multilevel representation of a design product as shown in the following sections.

## 2.4 THE CONCEPT OF PERSPECTIVE

The ontology provides the basic conceptualisations that can be used to reason about a technical system. A conceptualisation will be represented by a conceptual schema. Formally, a conceptual schema CS is a tuple  $\langle E, R, A/D \rangle$  where

- $E = \{E_i\}$  is a set of entity types;
- $R = \{R_j^k(E_{j1}, \dots, E_{jk})\}$  is a set of relation types. Each relation type has a degree ( $k$ ) that is the number of participating entity types;
- $A/D = \{A_j/D_j\}$  is a set of attributes representing general properties of entities or relations types. Each attribute ( $A_i$ ) has an associated domain ( $D_i$ ) specifying the range of values the attribute may take.

Conceptual schemes are used, during articulation, to provide a semantic content to the specific entities and relations represented in a model. As a consequence the model can be seen as an evolving set of design entities that are instances of the concepts types specified by its associated schema. As an example, consider the behavioral model of a simple electrical circuit constituted by a battery and a resistor connected in series as represented by the following three equations:

$$E_1 = V_1 \quad (1)$$

$$V_1 = R_1 * I_1 \quad (2)$$

$$R_1 = c_1 \quad (3)$$

where  $E_1$  is the electromotive force of the battery,  $V_1$  is the voltage across its terminals,  $R_1$  is the constant resistance of the resistor and  $I_1$  is the current flowing through the system. The associate conceptual schema is

$$CS = \langle \{E, V, R, I, c\}; \{V=R*I, E=V, R=c\}, \{\text{quantity\_value}/\text{Real}\} \rangle \quad (4)$$

Notice that the schema specifies the *types* of variables and relationships used in the model. The same schema can be associated to several models of electrical circuits each one representing a different number of instances of these types.

In selecting the constituents of a conceptual schema we enforce the restriction that no concepts of the same schema can be taxonomically related. In other words we do not mix concepts having different abstractness, generality, detail, coverage and resolution in the same conceptualisation. This choice results in a rigid multilevel system of conceptual schemes each one representing a single perspective.

By the term perspective we mean a specific choice of values (i.e. levels) along each of the above dimensions. Obviously, models inherit the perspective embodied in their schemes. It must be stressed that what can be seen from a given perspective is not "part" of the artefact but the whole artefact as can be perceived and conceptualised through its associated schema. Changing perspective means moving along one or more modelling dimensions thus changing the point of view (that is the conceptualisation) through which we interpret or represent reality.

Because of the particular organisation of knowledge we have adopted this amounts to performing a specific type of *knowledge transmutation* as discussed in (Toppano, 1999). Table 1 summarises modelling dimensions, conceptual hierarchies and associated knowledge transmutations.

TABLE 1. Basic knowledge transmutations and related modelling dimensions

| Modelling dimension                                                              | Conceptual hierarchy                    | Knowledge transmutation (and inverse)      |
|----------------------------------------------------------------------------------|-----------------------------------------|--------------------------------------------|
| Conceptual abstractness                                                          | Rank hierarchy of epistemological types | Conceptual abstraction (Concretion)        |
| Resolution                                                                       | Measure hierarchy                       | Relation or value abstraction (Concretion) |
| Generality                                                                       | Type hierarchy                          | Generalisation (Specialisation)            |
| Phenomenic coverage (range of phenomena that are explicitly represented)         | Rank hierarchy of idealised laws        | Reduction (Expansion)                      |
| Phenomenic coverage (accuracy of relations used to represent relevant phenomena) | Rank hierarchy of idealised laws        | Approximation (Elaboration)                |
| Detail                                                                           | Part hierarchy                          | Aggregation (Refinement)                   |

With reference to the above example, the schema (4) can be generalised by moving up the typological hierarchies TH1 and TH2 shown in Figure 1 thus obtaining

$$CS^* = \langle \{E, e, R, f, c\}; \{e=R*f, E=e, R=c\}, \{\text{quantity\_value}/\text{Real}\} \rangle \quad (5)$$

where 'e' is effort and 'f' is flow. By further specialising the schema in a different domain such as the hydraulic domain we obtain

$$CS' = \langle \{E, P, r, Q, c\}; \{P=r*Q, E=P, r=c\}, \{\text{quantity\_value}/\text{Real}\} \rangle \quad (6)$$

where 'p' is pressure, 'r' is the hydraulic resistance and 'q' is the volume flow rate. The schema can be associated to a simple hydraulic circuit constituted by a tank connected to a pipe. The sequence of a generalisation transmutation followed by a specialisation is called similarisation and it is at the base of analogical reasoning.

## 2.5. REPRESENTING ARTEFACTS USING MULTIPLE PERSPECTIVES

Figure 3 shows an example of representation using multiple perspectives. The considered artefact is a simple lighting system composed by a battery that supplies power to a light bulb when a switch is closed. The representation includes seven models of the device: a structural model (M7); two behavioural models (a quantitative model, M5, and a qualitative - causal - one, M6); a functional role model (M4), a functional process model (M3) and two teleological models (M1 and M2) at different levels of detail. We use a diagrammatic language to represent the models. Links between pairs of models specify the codesignation relations existing among elements belonging to two or more descriptions of the artefact. For example, the link between the structural model M7 and the behavioural model M5 specifies which physical quantities and laws in the behavioural model correspond to which terminals and components in the structural one; the link between the behavioural model M5 and the functional model M4 describes which physical laws in M5 are associated to which functional roles in M4, while the link between the functional role model (M4) and the process model (M3) specifies the correspondences existing between cofunctions (i.e. specific configurations of roles) and processes. Finally, the link between function and teleology is realised by associating goals in the teleological description with the phenomena (or processes) represented in the functional representation which are used to achieve them. The relation between processes and goals is, in general, many-to-many since a process may

participate to the realisation of several goals and, conversely, a goal can be fulfilled by utilising several processes. Codesignation relations can be used to switch from a description to another in order to focalise reasoning or disseminate information.

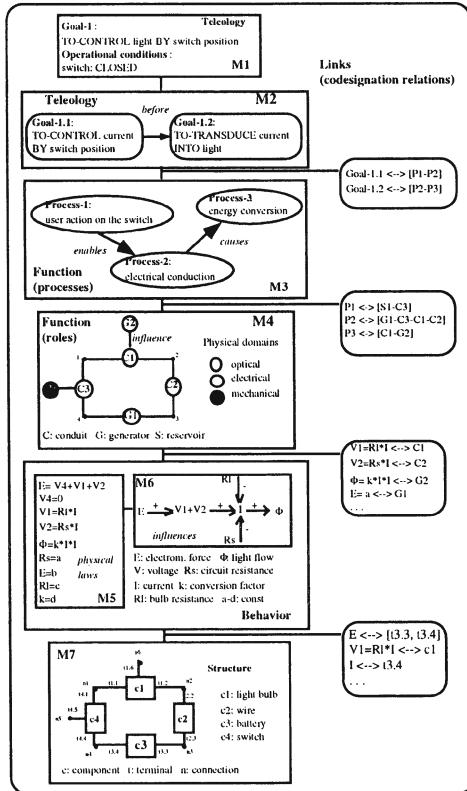


Figure 3. Example of representation using multiple models

### 3. Implementation: the MMforTED Prototype System

MMforTED is a cognitive tool. By this term we intend an instrument that can support, represent or perform an identifiable cognitive process that is part of the complete learning experience by a learner (van Joolingen 1999). The target cognitive processes are multiperspective analysis and multilevel reasoning. Multiperspective analysis means to be able to analyse and conceptualise an artefact from different perspectives and to be able to map corresponding elements belonging to different points of view. Multilevel

reasoning means to be able to integrate several points of view and representations to solve problems and to be able to change representation by selecting the aspects of the object under consideration that are more relevant and the level of accuracy, resolution, and detail of the descriptions that are deemed more appropriate for the problem to be solved.

### 3.1 ARCHITECTURE: INFORMATION SPACE AND NAVIGATION

The learning environment we developed is constituted by a collection of examples (cases) of simple artefacts. As a domain of application we have chosen electrical and fluid-mechanical devices whose schematic description can be expressed as a network of lumped-parameter idealised elements. According to the framework described in Section 2, a case is represented by a graph of models each one representing a specific conceptual perspective about the device under consideration. Models within a case are related together by codesignation links. Associative links are, instead, used to relate models of different artefacts featuring the same perspective.

The collection of all available models and their interrelationships constitutes a kind of "conceptual landscape" that can be criss-crossed in many directions according to the type of problem or task to be executed (e.g. description, explanation). Navigation through models includes:

- *browsing within a case*: the student can gain cognitive flexibility by being exposed to multiple interpretations (perspectives) of the same device (Spiro 1997). Changing perspective amounts to exercising a well defined knowledge transmutation;
- *browsing through cases*: the student can gain knowledge transferability by seeing multiple manifestations of the same interpretation.

Besides showing models of artefacts the system provides relevant questions, exercises and knowledge "about" the environment (i.e. meta knowledge) including the conceptual schemes used to build the models, the set of operational and representational assumptions lying behind each model and data about the kind of transmutation used to transform a model into a related one.

The actual system is a static electronic hypertext with modest interactivity and adaptation capabilities. Our interest is focused on reading, searching and browsing activities that take place while learning with hypermedia. Thus, the initial implementation of the system is aimed at providing a powerful interface for inquiry-based or explorative learning in a richly interconnected and structured information space. We adopted a client-server architecture for the system on the web. All instructional material is represented as HTML files on the server side. We selected the CGI technology to carry on the interactive functionalities (i.e. exercises

evaluation and search engine). In the CGI approach the user interacts with HTML entry forms in the web browser. Each time the server receives information from a browser that is intended for the CGI application the server forwards these data to the program which then replies with HTML pages. In all cases, there is no direct communication between the client and the CGI application: all information is routed through the web server. A limitation of this approach is that it is slow.

### 3.2 PRESENTATION

Figure 4 shows how the learner may access the information space. A typical page is divided into areas that are distinguished by different colour backgrounds.

#### *3.2.1 Reasoning within a model*

This is the main presentation area (A). Each device model is displayed in this area both diagrammatically and by a short textual description to facilitate interpretation and remembering (i.e. a dual code approach). The area includes major buttons for analysing the model:

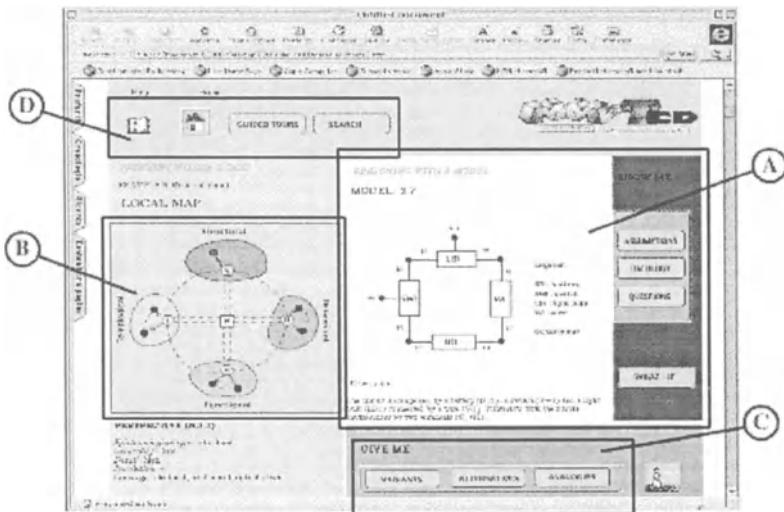


Figure 4. The page representing the structural model of a lighting system

- *Assumptions* visualises the main modelling and operational assumptions taken by the modeller to build the description. Operational assumptions

include operating modes (e.g. the electrical circuit is assumed to operate with the switch in the state "CLOSED"), as well as abnormality assumptions (e.g. the model may assume some form of component fault or malfunction).

- *Ontology* describes the conceptualisation used to build the description. Active terms can be used to navigate the web of conceptual knowledge.
- *Questions* specifies relevant questions that can be used to ask about the artefact. There are three kinds of questions:
  - General questions: these are the basic questions that are used to introduce the main epistemological types (e.g. What the system looks like? How does it work? How does it function? Why it is needed? How can be used?).
  - Templates: represent prototypical questions that are an elaboration of the general questions. As an instance, the questions: Which physical variables describe the potential behaviour of component Ci in operating mode OMj? What affects physical variable Xk? How does the variable Xi change if the variable Xj decreases (or increases)? etc. are a possible elaboration of the general behavioural question How does it work?
  - Specific questions: represent examples of the templates instantiated for the case under consideration.

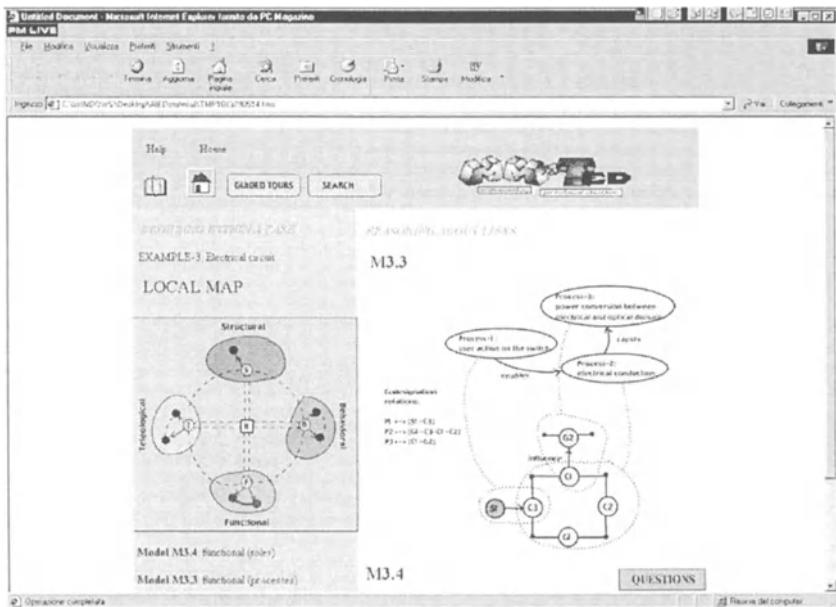
Templates are linked to ontologies and to a set of exercises that are simple quizzes aimed at testing the ability of the learner to discriminate among epistemological types and relevant questions. It must be stressed that exercises are not used to test the student skill at understanding they are used to help build it.

- *What-if*: proposes possible changes of the assumptions lying behind a model in order to analyse their effects both within a perspective and across multiple perspectives. This provides the student with an opportunity to learn about the case before it is modified and about the modification itself. For instance, by clicking on the "What if" button the student may select a different operating mode for the artefact or hypothesise an abnormal state for a component and observe the effects of the modification across epistemological types. Actually, the selection is made within a set of predefined modifications.

### 3.2.2 Browsing within a case

This is the primary navigation area (B). It contains a clickable map to browse within a case. The map shows the available models of the artefact under consideration which are clustered according to their epistemological type. The currently selected model is specified by its perspective. By clicking on the arc connecting a pair of models it is possible to visualise

relational knowledge e.g. codesignation relations, Figure 5. This is valuable to connect model fragments (snippets) across perspectives and understand the behaviour and functioning of specific subsystems or component assemblies. The system provides a set of templates – accessible through the “Questions” button - that can be used to inquiry about the relationships existing between concepts belonging to different conceptualisations. (e.g. Which functional roles support process X?, Which variables are associated to the terminal T of the component C? Which processes realise goal G?).



*Figure 5. A page visualising codesignation relations between a functional role model and a process model of the lighting system*

Following the links bottom-up (i.e. from structural descriptions to teleological descriptions through behavioural and functional ones) it is possible to provide a teleological explanation of the behaviour of a component or to explain the reasons behind a particular arrangement of components. In design problem solving this is valuable when we want to evaluate the effect of a modification of an artefact such as the elimination of a component or the substitution of a component with another one. Analogously, it is possible to follow the links top-down (i.e. from teleological descriptions to structural ones) to assign blame to components i.e. to identify which components are responsible for the achievement of a

given set of goals. This task is accomplished, for example, in design, when it is needed to establish which part of a structural description of an artefact can be reused to fulfil a given set of new goals.

### *3.2.3. Browsing through cases*

This is the secondary navigation area (C). It contains several buttons for browsing through cases:

- Alternatives: proposes a list of models, across cases, which share the same ontology (hence the same perspective) of the currently displayed model. Browsing through alternatives can be valuable to inventory elements of the vocabulary used to build the models under that perspective and note the grammar of their possible combinations.
- Variants: proposes a list of cases which have the same purpose (i.e. teleological goal) of the currently displayed model but different functional organisations (e.g. different process networks).
- Analogies: proposes a list of examples which share the same *type* of functional organisation but the organisation has been realised by exploiting different physical phenomena. As an example, an hydraulic circuit constituted by a tank of water, a pipe, a valve and a turbine has the same functional organisation of the lighting system.
- Where am I: shows actual position (case) in the global environment of available cases.

### *3.2.4 General services*

Finally, the area at the top of the page (D) includes general services, namely:

- Help: this is a link to a general presentation of the approach;
- Home: a link to the site home page;
- Guided tours: include a list of predefined pathways (tours) in the model environment. Each tour is described by i) an introduction specifying the main purpose or the tour and ii) a sequence of stop places (referring to specific pages of the hypertext) highlighting the characteristics of the model displayed in the page with respect to the stated purpose. Some tours are aimed at showing how to perform a means/ends analysis within a case. Others show simple examples of conceptual design exploiting design knowledge represented by the available cases. For instance, a tour shows how to generate the conceptual solution of a pressure gauge by analogical reasoning. The example uses a case representing a voltage gauge in the electrical domain as the source for the analogy. Another tour shows how to design a pressure to voltage transducer by composition of

model snippets of two available cases. See Toppano (1997) for further details about these examples.

- Search: this button allows the learner to exploit the use of an automatic model selector called TAMS - an acronym for Task Adaptive Model Selector - to choose a model on the base of desired perspective or question (Toppano 1996).

#### 4. Experimental Activity

The experimental activity was done within the ICARO project a national initiative whose principal aim is to provide the discipline of technical education with an epistemological foundation and to produce a set of guidelines to help teachers develop educational curricula and web-based educational material. The project involves a selected set of schools and teachers, several regional educational research institutions and experts from different fields such as pedagogy and engineering sciences. Teachers have been fully involved in the design and evaluation of the system as well as in planning the activity to be done with the students. This activity has three distinct phases:

- *Pre-assessment*: students are asked to write an essay where they describe a given artefact and explain how the artefact achieves its purpose. The goal of this activity is to pre-assess the student's capability to understand the artefact functioning and to use appropriate concepts and terminology in describing her understanding.
- *Reflective learning*: The teacher introduces the multiperspective approach and models how to deal with the MMforTED environment. Then she fades her involvement while coaching and supporting the students in their own navigation. The students develop their own comprehension by exploring the information space of cases and observing the many ways artefacts may be described and the many ways the general principles behind their functioning become manifest. This is in accordance with modern learning theories such as constructivistic theory that points out the positive effects of letting the learner create her understanding and knowledge structures. Moreover, the student is requested to reflect on the structuring of knowledge presented through the environment and on the ontologies and questions that each model calls forth in order to make hypotheses about the behaviour and functioning of the artefacts. The underlying rationale is that making hypotheses involves the ability to formulate questions about the object under consideration. In particular if we consider an hypothesis as a guessed relationship among a set of entities (i.e.  $Rk(ei, \dots ek)$ ) than the search space of hypotheses that can be formulated by the student critically depends on the student's knowledge about the possible

relationships that can be assessed and the types of entities that can be involved in a relationship. Hence, our claim is that, if the student has been exposed to a space of possible interpretations then she can enrich her vocabulary of entities and relationships and, thus, enlarge her search space for hypothesis construction.

- *Post-assessment:* finally, the student is invited to write an essay where she describes the behaviour and functioning of a given artefact (which is different from that used in pre-assessment). The result is then compared with the essay generated before the experimentation.

The goal of student assessment is to draw conclusions about the effectiveness of the system by measuring learners outcomes (Gilbert et al. 2001). Assessment is difficult for various reasons. First, the directions given to the learners for carrying out the task (writing an essay) are vague enough for the task to be considered open-ended i.e. the task has a well defined outcome in abstract terms but this outcome may be instantiated in many ways. Second, the system is "open" in the sense that it doesn't stop a learner going wherever she wishes within the environment. As a consequence, there is no fixed series of steps to reach the outcome, there is no a preferred reading of the instructional material. This is very different from most traditional ITS systems focusing on the learning of procedural knowledge and skills. In these systems the student is requested to solve a number of problems and to input intermediate problem solving steps and products. The tutoring system - or the teacher - compares the problem solving path thus generated to an ideal path based on expert performance. If the student's answer differs from that of the expert model, then a remediation move is needed which may take the form of an explanation of what step the expert proposes and why it is considered as the right step to take. This approach simplifies the student assessment since progress can be measured in terms of correctly solved problems and there exists a well founded theoretical notion on skill acquisition. None of these advantages are available in the case of coming to understand something. As reported by (Andriessen and Sandberg 1999) understanding is basically expressed in the medium of language and is not intrinsically tied to particular performance. Furthermore learning is inherently personal: the student creates meanings not only by following the links explicitly represented in the hypertext by the designer but also on the base of various kinds of intertextual relationships that are implicit in the her mind.

In the post assessment phase, we ask the students to write an essay in which they describe an artefact and explain its behavior and functioning. The idea is to engage the students in a constructive activity (Iacucci and Pain 2001). When students are required to explain the functioning of an

artefact they abandon their status of “hypertext audience” and are engaged in being teachers of the meanings they have created by exploring the learning environment and reflecting on the extent and quality of their knowledge. Evaluating their learning experience involves evaluating them as designers of models. These models are then compared in terms of consistency, terminological appropriateness and completeness with respect to epistemological types and perspectives. We can only provide a qualitative assessment. While many of the pre-assessment essays are characterised by limited generation of concepts, poor terminology and a general inability to describe the object under consideration from several conceptual perspectives, we find that all students involved in the experimentation, each one at his or her own pace, made improvements in their skill to interpret and argue about the given artefacts. We are actually exploring the role of collaboration in improving such results. The idea is that if the students have to collaborate to write the final essay then they are forced to explicitly externalise their conceptions and points of view, understand each other and negotiate meanings. The knowledge negotiation approach in Education holds that the goal of education is not knowledge acquisition per se, but to acquire the flexibility to participate in the discourses of several communities of practice, that is specific groups of professionals acting and communicating in specific ways. Participating in professional groups implies the ability to understand the important debates and problems and use the right language and conceptualisations to examine and influence ongoing debate.

## 5. Conclusions

The instructional system we have illustrated integrates ideas from three main areas:

- model based reasoning
- knowledge modelling and ontological engineering
- hypermedia and web based instruction and education

One of the primary contributions of this work, one that is the focus of the paper, is the notion of perspective and multiperspective analysis. The proposed notion is strongly related to the particular categorisation and organisation of conceptual knowledge we have adopted.

The MMforTED system has been designed to reflect the multilayered organisation of conceptual knowledge. Students are invited to browse the learning environment and to discover the coherent structure and linkages that exist among the instructional materials. The system should be seen more as a collection of resources to be accessed and freely interpreted in

order to enable certain processes or interactions to take place rather than as a collection of materials to be learned. Teachers are expected to develop by themselves the models on the web, according to the ontology and the modelling approach described in the paper.

MMforTED seems to meet general acceptance. One reason is that it is perceived as an elaboration of an actual practice: the RARECO (i.e. Representation, Analysis, RElation, COmmunication) model. This is an heuristic method widely used in the primary school and in the first two years of the secondary school to support the construction of knowledge about artefacts and the production of technical texts.

### Acknowledgements

The author would like to thank the referees of the paper for their suggestions.

### References

- Andriessen, J and Sandberg, J: 1999, Where is education heading and how about AI?, *IJAIED* **10**: 130-150.
- Ball, N, Matthews, P and Wallace, K: 1998, Managing conceptual design objects, in JS Gero and F Sudweeks (eds), *Artificial Intelligence in Design '98*, Kluwer, Dordrecht, pp. 67-86.
- Chittaro, L, Guida, G, Tasso, C and Toppino, E: 1993, Functional and teleological knowledge in the multimodelling approach for reasoning about physical systems: a case study in diagnosis, *IEEE Trans. on Systems, Man, and Cybernetics* **23**(6): 1718-1751.
- Chittaro, L, Tasso, C and Toppino, E: 1994, Putting functional knowledge on firmer ground, *Applied Artificial Intelligence* **8**(2): 239-258.
- Coyne, RD, Rosenman, MA, Radford, AD, Balachandran, M and Gero, JS: 1990, *Knowledge-Based Design Systems*, Addison-Wesley, Reading, MA.
- Falkenberg, ED, et al: 1998, FRISCO - A Framework of Information System Concepts - The FRISCO Report IFIP WG 8.1 Task Group FRISCO, Web version: <ftp://ftp.leidenuniv.nl/pub/rul/fri-full.zip>.
- Fujii, H and Gero, JS: 2000, A situated design agent to assist in the explanation of situatedness in designing, Poster Abstract, *Artificial Intelligence in Design '00*, pp.18-20.
- Gadamer, H: 1976, *Philosophical Hermeneutics*, University of California Press, Berkley, CA.
- Gilbert, JE, Hubscher, R and Puntambekar S (eds): 2001, *AIED 2001 Workshop Papers: Assessment Methods in Web-Based Learning Environments & Adaptive Hypermedia*, San Antonio, Texas.
- Gruber, TR: 1993, A translation approach to portable ontology specification, *Knowledge Acquisition* **5**(2): 199-220.
- Haymaker, J, Ackermann, E and Fischer, M: 2000, Meaning mediating mechanism, in JS Gero (ed.), *Artificial Intelligence in Design '00*, Kluwer, Dordrecht, pp. 691-715.
- Heylighen, A and Verstijnen, IM: 2000, Exposure to examples, in JS Gero (ed.), *Artificial Intelligence in Design '00*, Kluwer, Dordrecht, pp. 413-432.
- Hieb, MR and Michalski, RS: 1993, Multitype inference in multistrategy task-adaptive learning: dynamic Interlaced Hierarchies, in RS Michalski and G Tecuci (eds),

- Proceedings of the 2<sup>nd</sup> International Workshop on Multistrategy Learning*, Harpers Ferry, West Virginia, pp.3-17.
- Iacucci, C and Pain, H: 2001, Making sense of open representations through a constructive hypertext: how to evaluate learning?, in JE Gilbert, R Hubscher and S Puntambekar (eds), *AIED-2001 Workshop Papers: Assessment Methods in Web-Based Learning Environment & Adaptive Hypermedia*, pp. 14-20.
- Laurillard, D: 1993, *Rethinking University Teaching*, Routledge, London.
- Mackellar BK and Peckham, J: 1998, Multiple perspectives of design objects, in JS Gero and F Sudweeks (eds), *Artificial Intelligence in Design '98*, Kluwer, Dordrecht, pp. 87-106.
- Marijomaa, E: 1997, Aspects of relevance in information modelling. methodological principles and conceptual problems, *Filosofisia Tutkimuksia Tampereen Yliopistosta* **63**.
- MQ&D Group: 1995, Qualitative reasoning: a survey of techniques and applications, *AICOM* **8**(3/4): 3-17.
- Noy, NF and Hafner, CD: 1997, The state of the art in ontology design: a survey and comparative review, *AI Magazine* **18**(3): 53-74.
- Ohlsson, S: 1993, Learning to do and learning to understand: a lesson and a challenge for cognitive modelling, in P Reimann and H Spada (eds), *Learning in Humans and Machines*, Pergamon Press, Oxford, pp. 37-62.
- Perrussel, L, Charrel, PJ and Rothenburger, B: 1997, A formalism for inter-perspective relationships checking, in H Kangassalo, JF Nilsson, H Jaakkola and S Ohsuga (eds), *Information Modelling and Knowledge Bases VIII*, IOS Press, Amsterdam, pp. 1-10.
- Protzen, J, Harris, D and Cavallin, H: 2000, Limited computation, unlimited design, in JS Gero (ed.), *Artificial Intelligence in Design '00*, Kluwer, Dordrecht, pp. 43-52.
- Rosenman, MA and Gero, JS: 1996, Modelling multiple views of design objects in a collaborative CAD environment, *INCIT'96 Proceedings*, pp. 49-61.
- Sim, SK and Duffy, AHB: 1998, A foundation for machine learning in design, *Artificial Intelligence for Engineering Design, Analysis and Manufacturing* **12**: 193-209.
- Simoff, SJ and Maher, ML: 1998, Designing with the activity/space ontology, in JS Gero and F Sudweeks (eds), *Artificial Intelligence in Design '98*, Kluwer, Dordrecht, pp. 23-43.
- Spiro, R: 1997, Knowledge acquisition for application-cognitive flexibility and transfer in complex content domains, in BK Britton and SM Glynn (eds), *Executive Control Processes in Reading*, Erlbaum Assoc, Hillsdale, NJ, pp. 177-199.
- Toppano, E: 1999, Using graph transformations to support multilevel reasoning in engineering design, *Machine Graphics & Vision* **8**(3): 395-425.
- Toppano, E.: 1997, Multistrategy modelling: a case study in design, in H Kangassalo, JF Nilsson, H Jaakkola and S Ohsuga (eds), *Information Modelling and Knowledge Bases VIII*, IOS Press, Amsterdam, pp. 31-46.
- Toppano, E: 1996, Rational model selection in large engineering knowledge bases, *Applied Artificial Intelligence* **10**(3): 191-224.
- Uschold, M: 1998, Knowledge level modelling: concepts and terminology, *The Knowledge Engineering Review* **13**: 5-29.
- van Joolingen, W: 1999, Cognitive tools for discovery learning, *JAIED* **10**: 385-397.
- Varejao, FM, Garcia, AC. et al.: 2000, Towards an ontological framework for knowledge-based design systems, in JS Gero (ed.), *Artificial Intelligence in Design '00*, Kluwer, Dordrecht, pp. 55-75.

## FROM CONCEPT TO EMBODIMENT: CHALLENGE AND STRATEGY

ZHI GANG XU, MING XI TANG AND JOHN HAMILTON FRAZER

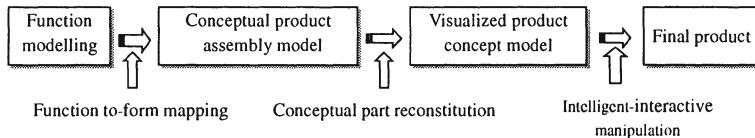
*The Hong Kong Polytechnic University  
China*

**Abstract.** Facing with the difficulty in information propagation and synthesizing from conceptual to embodiment design, this paper introduces a function-oriented, axiom based conceptual modeling scheme. Default logic reasoning is exploited for recognition and reconstitution of conceptual product geometric and topological information. The proposed product modeling system and reasoning approach testify a methodology of “structural variation design”, which is verified in the implementation of a GPAL (Green Product All Lifecycle) CAD system. The GPAL system includes major enhancement modules of a mechanism layout sketching method based on fuzzy logic, a knowledge-based function-to-form mapping mechanism and conceptual form reconstitution paradigm based on default geometric reasoning. A mechanical hand design example shows a more than 20 times increase in design efficacy with these enhancement modules in the GPAL system on a general 3D CAD platform.

### 1. Introduction

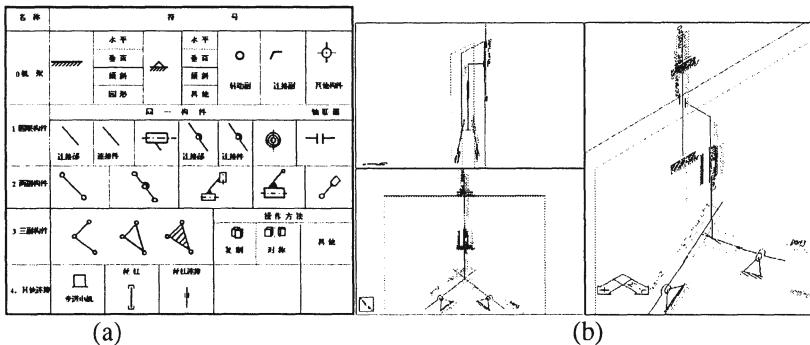
Existing approaches towards a computer aided conceptual design are insufficiently integrated with the downstream activities due to the highly ambiguous and uncertain information that has to be managed, propagated and synthesized in the embodiment design. To cope with this problem, a lot of recent researches focused on requirement management (McKay 2001), functional design (Tomiyama 1993), requirement-to-function mapping (Feng 2001) and function-to-form mapping (Roy 2001) etc. But the information generated from the above mapping processes is usually provisional, inconsistent and uncertain, making it intractable without proper arrangement of the information generated throughout the design process.

Our prior work suggested a generalized mapping strategy, Figure 1, to integrate conceptual and detail design, which consists of four main modules.



*Figure 1.* Process of generalized mapping

1) A *functional element library* is developed, Figure 2(a), to support mechanism 3D layout sketching design based on fuzzy logic, Figure 2(b).



*Figure 2.* Functional element library and 3D layout sketch

Firstly the element in Figure 2(a) is dragged and dropped on a certain 3D orientation plane to construct a physical conceptual mechanism layout, Figure 2(b). Then the symbolic/geometric fuzzy logic reasoning is exploited to deduce the transmission chain, i.e., the topological information of the conceptual product for the further establishment of conceptual assembly model, Figure 3 (a).

2) A “function-to-“functional carrier”-to-form” *mapping* paradigm is put forward to finish the conceptual embodiment design (Xu 1999 a), where each functional element is mapped onto an abstract feature for the functional requirement i.e., positioning, power/energy transmission etc. Information generated from the mapping process is arranged in an assembly model (Xu 1999 b) similar to the virtual link model reported in (Lee 1985).

3) An assembly model is in a network structure, Figure 3(a), each node of which is an information unit called functional carrier, and the arc is the constraint of qualitative and quantitative information depending on different design stages. The assembly model has the total correlation information of part-to-part, part-to-functional carrier and functional carriers (of one part)-to-functional carriers (of another part).

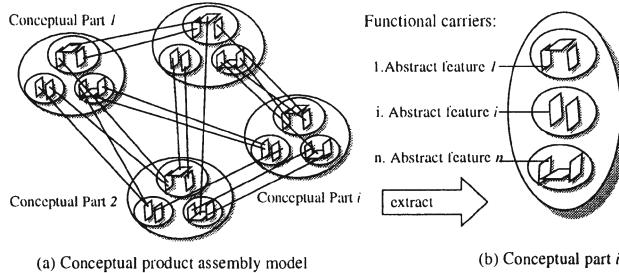


Figure 3. Conceptual product model

4) A functional carrier is stored in each part in a random order, Figure 3 (b), and the information regarding size or shape etc is in a high degree of uncertainty. So in this paper geometric reconstitution of functional carriers will be talked in detail, Figure 4. A methodology for ambiguous geometric information modeling, vague geometric information reasoning and algorithms on part reconstitution based on default geometric reasoning (DGR) are put forward.

The arrangement of this paper is as follows. Section 2 gives a review in the related area; Section 3 gives the detail explanation of conceptual part model in Figure 4 (a); Section 4 presents algorithms in Figure 4 (b) focusing on default geometric reasoning and related geometric foundations; Section 5 shows examples for DGR application; Section 6 presents a product design example and draws the conclusions of this paper.

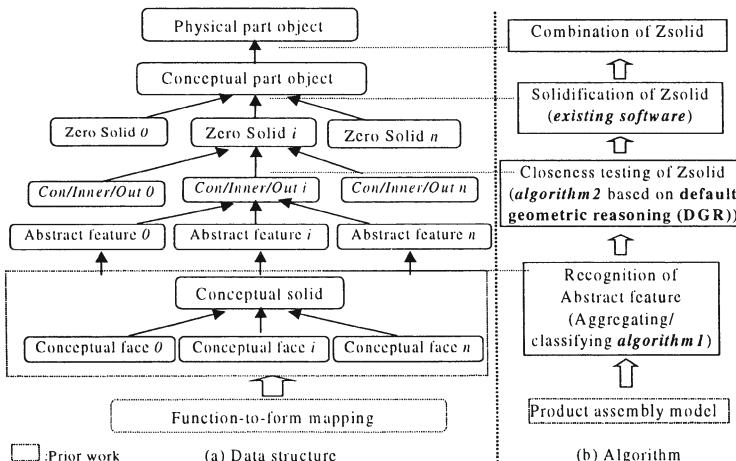


Figure 4. Conceptual synthesis related data structure and algorithm

## 2. Review

### 2.1 CONCEPTUAL DESIGN

Conceptual design and product life cycle design are concerned with a broader area of interests. From the advent of the first computer, design theory and methodologies relating to computational design technology have developed correspondingly (March 1984; Finger 1989) and fallen into various engineering perspectives, among which axiomatic design theory gained more attention and has been verified in various engineering applications (Suh 1990; Vigain 1996).

Axiomatic design theory (Suh, 1990) puts forward two axioms: the Independence Axiom and the Information Axiom. They are stated in declarative form as follows: Axiom 1, The Independence Axiom: Maintain the independence of functional requirements (FRs). Axiom 2, The Information Axiom: Minimize the information content.

The suggested conceptual part model in this paper, such as zero solid (Figure 4(a), as detailed in Section 3.2) is fundamentally based on Suh. N. P's axioms (1990), and it is a new trial to apply the Axiom to a CAD system for conceptual design information management.

Up till now, reports on conceptual design research increased enormously (Krause 1995; Screenivasa 1996) due to the reason that as much as 70-80% of the budget are determined at this stage, when drawbacks are easier to be notified and rectified based on the concurrent DFX philosophy. Research efforts are undertaking from different perspectives. Concurrent engineering (AI-Hakim 2000) is explored and the approaches relating to reliability and other functional perspectives at the conceptual stage are developed using graph theory of "tree" and "forest". Design verification (Deng 2000) is carried out automatically through the use of constraint-based functional design approach based on a variable dependency graph. The constraints are propagated over graphs and the values of the design variables are checked against these constraints. Sketching (Lipson 1995) has the immediate usage, while fuzzy knowledge based sketching (Qin 2000) seems more promising, where sketched mouse input is identified, precise 2D primitives are generated, and the 2D relationship is inferred. Novel researches of using Features as information carriers to the downstream applications are put forward (Brounnett 2000), where feature models that support conceptual design are suggested. Schutle (1993) noticed that at the early design phase, it is impossible and unnecessary to make clear all the details of feature constraints like size, shape, precise position etc, usually the mechanical part has only a conceptual frame and/or effective functional area, so "functional feature" is suggested and verified. Accordingly in this paper, the concept of

an “abstract feature” is suggested, Figure 4 (a) and the algorithms, Figure 4 (b), are verified to support conceptual design.

## 2.2 GEOMETRIC REASONING

Nowadays’ CAD systems have been developed to support the modeling of product geometry, they often require complete, precise, or detailed specification on the geometry, which is usually unavailable until later stage of the design process. The traditional geometric reasoning approaches are largely divided into three types: numerical iterative method, symbolic methods and Knowledge-based methods.

Light and Gossard (Light and Gossard 1982) in a numerical analysis-based approach proposed a prototype 2D CAD system. They solve non-linear simultaneous equations derived from geometric constraint relationships using Newton-Raphson method. Recent researches use homotropic methods (Lamure 1996) to solve equations in a single variable. This method avoids common problems associated with other numerical approaches, such as the need for identifying an initial point and quite often the resulting unstable and chaotic behavior. Symbolic approaches use computer algebra techniques. Owen (Owen 1991) described a system for algebraically solving geometric constraints represented as quadratic equations, which include two-dimensional geometries consisting of points, lines, and circular arcs. Bouma et al. (1995) described a two-dimensional geometric constraint solver, similar to Owen’s, that also deals with quadratic geometric constraint equations. The algorithm is applied recursively to *place* clusters into larger *super clusters*. The knowledge-based geometric reasoning techniques presented by Lee, (1996) utilize graph methods. Their geometric reasoning algorithm is completed using representation of a well-constrained sub graph based on one-to-one graph inference (OOGI) and two-to-one graph inference (TOGI), which are the foundation of the geometric inference process. The algorithm identifies sub graphs using OOGI and TOGI, and rewrites them in simpler forms until the graph is fully constrained. Serrano (1991) applied this approach to automatic generation of alternative dimensioning schemes for CAD drawings.

Non-manifold geometry modeling proposed by (Masuda 1996) tends to tackle the insufficient geometric problems, while spatial reasoning (Rajneet 1994) is profitable for defining the qualitative information as “front-back”, “left-right”, “up-down”, which appear frequently at conceptual design stage. Qualitative Reasoning uses descriptors like  $(-, 0, +)$  instead of real numbers to do its work. Rowan (1991) applied geometric reasoning algorithms in feature based CAD/CAM systems for feature spatial relations arrangement. Shuichi Shimizu (1996) declared that traditional CSG models, whose

geometric and topological models are fixed, do not fit with conceptual design where trial and error occurred frequently. Assumption-Based Truth Maintenance System (ATMS for short) is introduced to the 3D modeling algorithms. Actually ATMS utilizes redundant reasoning and it is so time-consuming in some cases that it is considered not fit for insufficient solid modeling requirement.

### 2.3 DEFAULT LOGIC

A logic for default reasoning is proposed by Reiter (1980), based on the convention that “what we know about the world is ‘almost always’ true”, which is similar to the problem aroused in conceptual design, when information is incomplete. Birds that can fly (like sparrow) or can’t fly (like penguin) are used to show the default rules.

$\text{BIRD}(x) : \text{MFLY}(x) / \text{FLY}(x)$ .

Means “If  $x$  is a bird, then in the absence of any information to the contrary, it infers that  $x$  can fly”. It is not always true. However the interpretation we adopted is that “if  $x$  is a bird and it is consistent to believe that  $x$  can fly, then it infers that  $x$  can fly”. The exception to flight is then given a standard first order representation.

( $x$ ).  $\text{PENGUIN}(x) \supset \neg \text{FLY}(x)$

( $x$ ).  $\text{OSTRICH}(x) \supset \neg \text{FLY}(x)$  etc.

At the early conceptual design stage, geometric information is ambiguous and uncertain, and assumptions need to be given, from which some deductions and decisions can be made, and the hypothesis can then be verified and testified by design instances to modify the fact database. If only two opposite faces are available, what kind of conclusion should it derive? Sometimes it is regarded as a “slot” feature, while there is an exception, a “cavity” for example also has two opposite faces. Analogous to the bird interpretation, feature interpretation is given as follows.

$\text{TWO\_OPPOSITE\_FACES}(x) : \text{MSLOT}(x) / \text{SLOT}(x)$ .

The exception to slot feature is then given a standard first order representation.

( $x$ ).  $\text{CAVITY}(x) \supset \neg \text{SLOT}(x)$  etc.

This default reasoning paradigm is proved to be effective in coping with conceptual design problems aroused in generalized mapping process, when geometric information is incomplete.

In summary, the above mentioned constraint satisfaction approaches and ATMS etc. are effective in solving traditional 2D and 3D constraint problems. However, geometric reasoning for conceptual design goes far

beyond just solid modeling problems, it is too much complex to make these approaches prohibitive. So hybrid approaches of modern AI technology with traditional geometric reasoning methods are promising.

### 3. Definition of Conceptual Geometric Data Structure

#### 3.1 DEFINITION OF CONCEPTUAL GEOMETRIC DATA STRUCTURE

*Functional face* is defined in Table1.

TABLE .1 Hierarchical definitions of conceptual faces.

|           | Conceptual face: $F_i.type$ | Concept and definition                                                                           |
|-----------|-----------------------------|--------------------------------------------------------------------------------------------------|
| Evolution | Directional face $DC$       | Described by outer vector (or axis of symmetry), a kind of limited face                          |
|           | Oriented face $OC$          | Described by outer vector (or axis of symmetry) and a positioning point, a kind of limited faces |
|           | Qualitative face $QC$       | Described by outer vector (or axis of symmetry), a positioning point and a functional area.      |
|           | Boundary face $BC$          | Has the geometric information of boundary, vertex, edge, loop etc                                |
|           | Closed face $CC$            | Traditional displaying face                                                                      |

*Conceptual Solid (CS)* is composed of a group of functional faces (table 1)  $F_i, i=1,2,..n$ , and/or position correlations, denoted by  $S$ .

*Closeness of Conceptual Solid*, for the Conceptual Solid  $S$ , if  $F_i, i=0,1,..n$ , under a certain positioning condition can constitute a 3D closed space, then the solid  $S$  is called closed. A closed solid is denoted by  $G$ , which is called a complete bond graph.

If sub-graph  $G_i$  is the graphs of related conceptual faces belonging to  $G$ ,  $\tilde{G}_i$  is a set of unrelated functional faces, then  $S \subset (G_b, \tilde{G}_i)$ .

*Symbol  $S_{;Fi.Type}$*  means that conceptual solid is composed of the functional faces of the type  $F_{i.Type}$ ,  $F_{i.Type} \subset (DC, OC, QC, BC, CC)$  (from table 1).

For example,  $G_{i:DC}$  means the conceptual solid being composed of related  $DCs$ , while  $\tilde{G}_{i:QC}$  means the solid being composed of unrelated  $OCs$ .

Analogous to reference proposed by Douglas (1994), five concave/convex relationships have been summarized in Figure 5 (a)-(e).

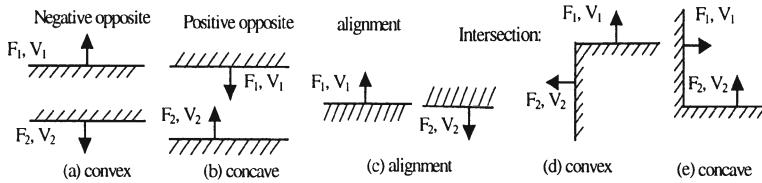


Figure 5. Conceptual solid definition based on concave/convex of faces

Tables 2 to 6 are situations between planar and rotational faces.

TABLE 2. Relationship of one planar and one cylindrical relationship: (c1: planar, c2: cylinder)

| Type                       | Inner  |         | Outer  |         |
|----------------------------|--------|---------|--------|---------|
|                            | Convex | Concave | Convex | Concave |
| Relationship with key_face |        |         |        |         |
| Figures                    |        |         |        |         |

TABLE 3. Relationship of one plane and two cylinders (c1, c2: inner cylinder)

| Type     | Relationship with key_face, √ means legible |           |         |
|----------|---------------------------------------------|-----------|---------|
| c1:Inner | Convex                                      | Convex √  | Concave |
| c2:Inner | Convex                                      | Concave √ | Concave |
| Figures  | NULL                                        |           | NULL    |

TABLE 4. Relationship of one plane and two cylinders (c1, c2: outer cylinder)

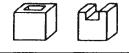
| Type     | Relationship with key_face, √ means legible |           |         |
|----------|---------------------------------------------|-----------|---------|
| c1:Outer | Convex                                      | Convex √  | Concave |
| c2:Outer | Convex                                      | Concave √ | Concave |
| Figures  | NULL                                        |           | NULL    |

**An Abstract Feature** is a special region attached to the base part of a component to express functional information and geometric implication of the component. It is an abstract information group composed of a number of faces and/or their relationships belonging to the domain of a conceptual solid.

TABLE 5. Relation of one plane and two cylinders:(c1: outer, c2: inner cylinder)

| Type     | Relationship with key_face, √ means legible |         |           |
|----------|---------------------------------------------|---------|-----------|
| c1:Outer | Convex √                                    | Convex  | Concave √ |
| c2:Inner | Convex √                                    | Concave | Concave √ |
| Figures  |                                             | NULL    |           |

TABLE 6. Definition of abstract feature

| Number of faces | Relationship and/or |         | Geometric semantics                                                               | Reconstitution difficulty | Default shape |
|-----------------|---------------------|---------|-----------------------------------------------------------------------------------|---------------------------|---------------|
| 1               | Plane               |         | Ambiguous                                                                         | Very large                | Block         |
|                 | Cylinder            |         | Definite                                                                          | Small                     | Cylinder      |
| 2               | Plane               | Convex  |  | Larger                    | Lower         |
|                 |                     | Concave |  | Larger                    | Lower         |
| > 2             | Plane               |         | Easier                                                                            | Lower                     | Higher        |

### 3.2 ZERO SOLID

Zero Solid (Zsolid) is put forward based on the following considerations:

**Zero height:** The functional face is regarded as an object with zero height, and it varies in size and shape.

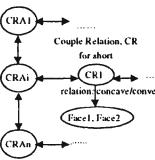
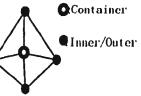
**Minimization of information:** The initial information contained in a conceptual part is minimized based on the direct and simple mapping. As the design example shows, the part has two critical elements: positioning cylinder for screw. So two Zsolids are deduced to fit for the above-mentioned functional requirement. The two Zsolids comprising the initial conceptual part have no redundant information, complied with Suh's axioms (Suh 1990).

**Independent of information:** There is no function overlapping between these two Zsolids, and no redundant information.

**Variation in design:** The reconstitution results of these two Zsolids are divergent. By interactive and intelligent manipulation of the Zsolid models, creative part design is easy to realize and the initial functional requirements of the part are also met.

**Zero Solid (Zsolid),** means variation in solid size, boundary, shape and topological information on the basis of the following data structure, Table 7.

TABLE 7. Data structure of Zsolid

| Z SOLID | Inner information            |  | Arrangement of inner features                           | Correlation between part 1,2,3                                                                                                                                                                                                                                        |
|---------|------------------------------|--|---------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|         | 1.Container, primary feature |  |                                                         |                                                                                                                                                                                                                                                                       |
|         | 2.Inner                      |  | First feature<br>Second feature<br>.....<br>Nth feature | Data structure of the arrangement of abstract features,CoupleRelaion Array, CRA for short:                                                                                                                                                                            |
|         | 3.Outer                      |  | First feature<br>Second feature<br>.....<br>nth feature |  <p>Container is the base feature to hold Inner/Outer features. Abstract feature net is set up</p>  |

There are three parts: **Container**, **Inner** and **Outer** in each ZSolid, where the **container** feature supports the **Inner** and **Outer** features. A data structure called **couple relation array** (**CRA**) is exploited to record the information related to abstract feature belonging to **Container/Outer /Inner** in the design process. The **couple relation** (**CR**) records the coupling faces and their relations in the format of **CR (f1, f2, Relation, 1: positive; -1: negative; 0: means no relation)**, which is recorded in **CRAi**.

A **conceptual part** is a network of zero solids, denoted by ZsolidNet Table 8.

TABLE 8. Definition of a conceptual part.

| Name | Composite of conceptual part | Inner structure of conceptual part                                                  | Algorithm                                         |
|------|------------------------------|-------------------------------------------------------------------------------------|---------------------------------------------------|
| Part | ZSolid1                      |  | Algorithm on closeness test of ZSolid             |
|      | ZSolid2                      |                                                                                     | Algorithm on adding supplementary faces to ZSolid |
|      | .....                        |                                                                                     | Solidification of ZSolid                          |
|      | ZSolidn                      |                                                                                     | Model change in size and shape                    |

#### 4. Algorithm on Default Geometric Reasoning

Part reconstitution is a process of adding supplementary faces and their topological relationships to a component to constitute a geometrically and topologically consistent object. In the rest of this paper, the theory and application related to geometric reasoning focusing on default geometric reasoning are talked about in detail.

#### 4.1 ALGORITHM1: AGGREGATING AND CLASSIFICATION

In order to describe the intersecting tendency between faces, the concept of “aggregating and classifying coefficient” is put forward, denoted by  $coe(f_i, f_j)$ , the definition is as the follows.

$$coe(f_i, f_j) = \frac{\bar{\theta}(\vec{v}_{f_i}, \vec{v}_{f_j})}{90^\circ}, \text{ where } \bar{\theta} = \begin{cases} \theta, & 0^\circ \leq \theta \leq 90^\circ \\ 180 - \theta, & 90^\circ < \theta \leq 180^\circ \end{cases}$$

$coe(f_i, f_j)$  means the extending of intersecting correlation between  $f_i$  and  $f_j$ , it is easier to get that:  $0 \leq coe(f_i, f_j) \leq 1$ .

The closer  $coe(f_i, f_j)$  is to 1, the stronger the intersecting tendency of these two faces have. Prior to the “aggregating” process, grouping is necessary, where the correlated faces are identified and put into corresponding Zsolids. It is obvious that when  $coe(f_i, f_j)$  is near to 1,  $f_i, f_j$  are more likely to intersect. Considering the geometric complexity, the coefficient is assigned a parametric range, say 0.8~1. Here is the definition of correlated faces.

**Correlated face**, when  $f_i$  is called the correlated face of  $f_j$ , denoted by  $f_i.RelatedFace$ , when and only when  $f_i$  meets the following requirements:

1. i)  $f_i, f_j$  have the correlation of concave/convex, and  $coe(f_i, f_j) > 0.8$ ;
2. ii)  $f_i, f_j$  are correlated by other face. “Correlated by other face” means: if there exists a face  $f_m$ , that  $f_i, f_m$  meet the criteria i), in the mean while  $f_m, f_j$  also meet the criteria i).

The analysis process is rather complex, Figure 6 gives only a simplified algorithm.

#### 4.2 RESEARCH ON DEFAULT GEOMETRIC REASONING

**Default Geometric Reasoning (DGR)**, means when any of the geometric elements in a “solid” such as: vertex, edge, face etc are incomplete or indefinite, supplement information is inferred for the missing geometric elements and the relationship among the geometric elements to construct a unambiguous informational consistent object under default logic. Here consistency means validity, completeness and uniqueness of a three-dimensional object.

##### 4.2.1 Theoretical Foundations

A **Path** as shown in Figure 7, can be a line or a curve that is continuous. When the start and end of a path is connected, then it is called a Closed-Path. Figure 7 (a) is not a path while (b), (c), (d) and (e) are paths.

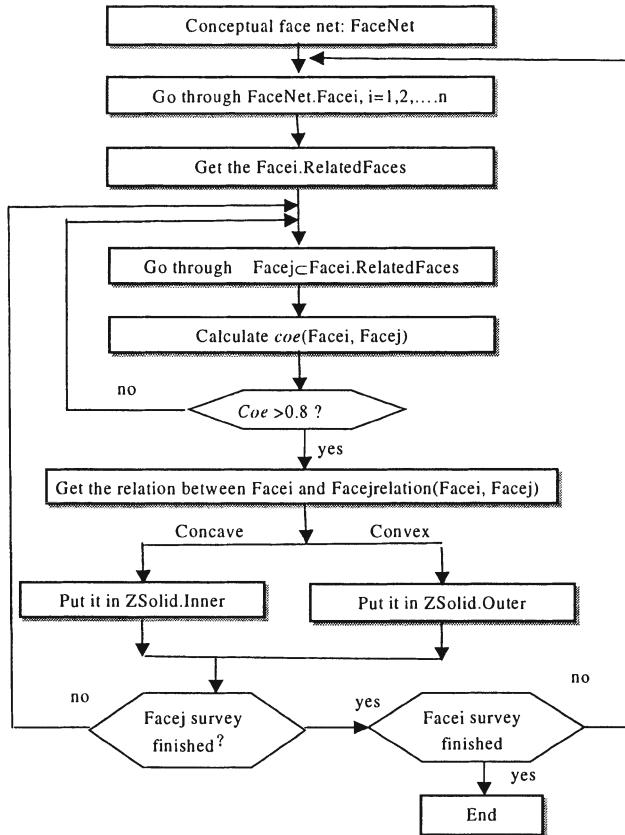


Figure 6. Algorithm on classifying/grouping

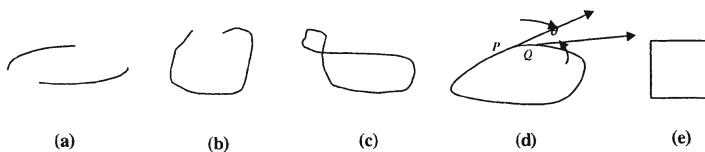


Figure 7. Examples of path turning

**Total turning**, as in Figure 7(d) is defined as the angle the oriented tangent vector turns around as the path stepped along.

**Theorem 1** (Closed-Path): A total turning is a topological invariant for a closed path. For any closed path the total turning is an integer multiple of  $2\pi$  (Jablokow, 1996).

**Theorem 2** (Simple Closed-Path): The total turning for a simple (non-intersecting) closed path is  $\pm 2\pi$  (clockwise or counter-clockwise, Jablokow, 1996).

Figure 7 (c) is not a simple path, (d) and (e) are simple paths.

From the above theories and definitions, the following theories are deduced.

**Theorem 3**, If conceptual solid *CS* can form one or more simple closed paths in every three orthogonal sections under a certain positioning condition, it is called closed.

**Theorem 4** (Whitney-Graustein): Two co-planar closed paths can deform between each other, if and only if they have the same total turnings (Jablokow, 1996).

On the basis of Theorem 4, deduction1 can be made.

**Deduction1:** Adding or deleting features on a conceptual solid did not affect its closeness.

#### 4.2 ALGORITHM BASED ON DEFAULT GEOMETRIC REASONING

The algorithm based on default geometric reasoning consists of two steps:

Step1: classifying the existing functional faces according to the out normals:

Given that the normal vector  $V_{11}$  on  $F_1$  and the normal vector  $V_{12}$  on  $F_2$  have formed a primary normal plane called main\_plane1 and two other orthogonal normal planes main\_plane2 ( formed by  $V_{21}, V_{22}$ ) and main\_plane3 ( formed by  $V_{31}, V_{32}$ ) as shown in Figure 8, then decide whether the normal  $V_3$  of a third plane  $F_3$  is co-planar with either of the main plane main\_plane*i*,  $i=1,2,3$ . The scheme is to calculate their mixed product  $\text{fabs}(\text{mixed-product}(V_i, V_j, V_3))$ , the smallest one is the main plane.

1. Rationalize  $V_i, V_j, V_3$ , and calculate their mixed product,  $V_i \subset (V_{11} V_{21} V_{31})$ ,  $V_i \subset (V_{12} V_{22} V_{32})$

$$M(V_i, V_j, V_3) = \text{mixed-product}(V_i, V_j, V_3) = (V_i \times V_j) * V_3 = \begin{vmatrix} v_{i,x} & v_{i,y} & v_{i,z} \\ v_{j,x} & v_{j,y} & v_{j,z} \\ v_{3,x} & v_{3,y} & v_{3,z} \end{vmatrix}$$

2. Then the main\_plane with the smallest value of  $M(V_i, V_j, V_3)$  is what  $V_3$  belongs to, Figure 8.  
step 2, closeness testing and supplementary faces addition:

- As shown in Figure 9, using any three orthogonal sections to “cut” the solid and get at least three directional 2D section profiles, Figure 9 (a);

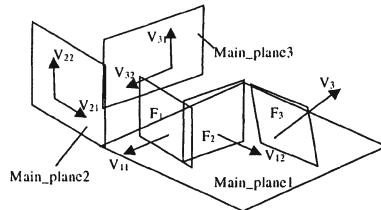


Figure 8. Main vector discrimination

- Decide whether any one of the 2D section profiles is a simple path;

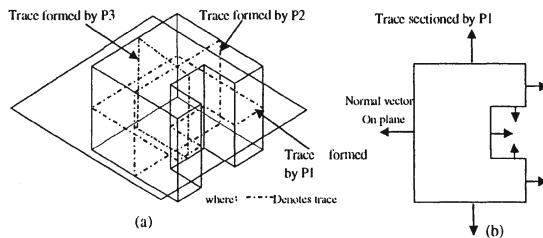


Figure 9. Closeness of conceptual solid

- If the answer is no, then add one complementary face at the middle position of the two faces, then go to step 2; If the answer is yes, then go back to step1.

In order to simplify the problem, three orthogonal section planes are utilized to decide whether three 2D sectional profiles are closed or not. Without losing generality, take the solid in Figure 9(a) as an example; using three sections  $P_1$ ,  $P_2$ ,  $P_3$  vertical to the ridges to cut the solid, three 2D sections are generated. Taking any one of the sections, Figure 9(b), calculate the total turning of the 2D profile, the scheme is like this:

$$\theta_{total} = \sum_{i,j=1, i \neq j}^n \theta(v_i, v_j), \quad \text{where } \theta(v_i, v_j) \text{ is the inclination of the vector } V_i \text{ and } V_j.$$

According to theorem 2 and theorem 3, any total turning of the 2D simple path should satisfy the following equation:  $\vartheta_{total} = \pm 2\pi$ , if it is not so, a supplementary face should be added between them. For instance, if the inclination (denoted by *seta*) of two normal vectors:

*seta* (*face1.normal*, *face2.normal*) = 180. Then a supplementary *face3* should be added between *face1* and *face2*.

Let,  $\text{seta}(\text{face1.normal}, \text{face3.normal}) = \text{seta}(\text{face3.normal}, \text{face2.normal})$

$$= \text{seta}(\text{face1.normal}, \text{face2.normal})/2.$$

The above total turning calculation algorithm is implemented to test the closeness of the conceptual solid, and provide the deterministic evidence for adding supplementary faces to construct a consistent object.

## 5. Design Example

The complexity of the reconstitution process lies in the fact that there are always too many correlations among faces, like that of planar to planar, Figure 5 and planar to conic, Tables 2 to 5. The main process is to find the “key face”, which has the most correlations with the other faces. Table 9 is the input face and their correlations. In fact the input faces have no boundary, size etc, they have only outer vectors and positioning points.

TABLE 9. Input faces and correlation.

| Func<br>face   | m_ff<br>Type | m_connec<br>t type | Normal<br>/axis of<br>Symmetry | Correlated faces and<br>convex/concave relation                                                                                                                  |
|----------------|--------------|--------------------|--------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| F <sub>1</sub> | Plane        | none               | (0, -1, 0)                     | (F <sub>2</sub> , convex), (F <sub>3</sub> , convex), (F <sub>6</sub> , concave)                                                                                 |
| F <sub>2</sub> | Cylinder     | Inner              | (0, 1, 0)                      | (F <sub>1</sub> , convex), (F <sub>3</sub> , convex), (F <sub>5</sub> , convex), (F <sub>6</sub> , convex), (F <sub>7</sub> , convex), (F <sub>9</sub> , convex) |
| F <sub>3</sub> | Plane        | none               | (0, 1, 0)                      | (F <sub>2</sub> , convex), (F <sub>4</sub> , concave), (F <sub>5</sub> , concave), (F <sub>1</sub> , convex)                                                     |
| F <sub>4</sub> | Cylinder     | Inner              | (0, 1, 0)                      | (F <sub>3</sub> , concave), (F <sub>5</sub> , concave)                                                                                                           |
| F <sub>5</sub> | Plane        | none               | (0, -1, 0)                     | (F <sub>4</sub> , concave), (F <sub>2</sub> , convex), (F <sub>3</sub> , concave)                                                                                |
| F <sub>6</sub> | Plane        | none               | (0, 1, 0)                      | (F <sub>2</sub> , convex), (F <sub>7</sub> , convex), (F <sub>1</sub> , concave)                                                                                 |
| F <sub>7</sub> | Plane        | none               | (0, -1, 0)                     | (F <sub>2</sub> , convex), (F <sub>8</sub> , concave), (F <sub>9</sub> , concave), (F <sub>6</sub> , convex)                                                     |
| F <sub>8</sub> | Cylinder     | Inner              | (0, 1, 0)                      | (F <sub>7</sub> , concave), (F <sub>9</sub> , concave)                                                                                                           |

|       |       |      |         |                                                          |
|-------|-------|------|---------|----------------------------------------------------------|
| $F_0$ | Plane | none | (0,1,0) | ( $F_2$ ,convex),( $F_8$ ,concave),( $F_7$ ,co<br>ncave) |
|-------|-------|------|---------|----------------------------------------------------------|

The analysis process is as the followings:

Step1: Find the key\_face. After analysis, the functional faces  $F_1$  and  $F_7$  are correlated with more faces than the others, so they are taken as the key\_faces.

Step2: Count the number of correlated cylinders to key\_face:  $F_3$  and  $F_7$ . Go through the correlated faces of  $F_3$ , and count the number, the number is 2, take  $F_3$  as the decisive face to create ZSolid0. After analyzing, the orientation and orientation condition of  $F_2$ ,  $F_3$  and  $F_4$  is as Figure10. Detailed analysis showed the correlation condition of  $F_1$ ,  $F_2$ ,  $F_3$ ,  $F_4$ , and  $F_5$  is as Figure10.

On the basis of the above analysis, the abstract feature of "Inner" part in ZSolid0 has three feature groups, i.e. CRAi,  $i=1,2,3$ , each CRAi has one or more CRj ( $j=1,2,3$ , Table 10), Table 10.

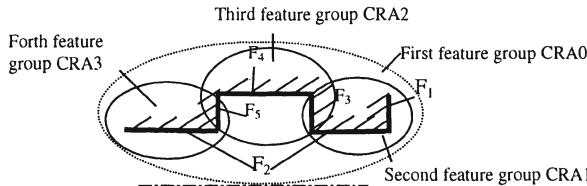


Figure 10. Illustration of the relative positioning

After detail analysis based on algorithm1 and algorithm2, the structure is deduced to be symmetric based on the key faces:  $F_7$ ,  $F_3$ . Analogous to  $F_3$ , the  $F_7$ -based ZSolid1 is also constructed, Table 11.

TABLE 10. Data Structure of Zsolid0

| CRA0.type == Container | CRA1.type == INNer First feature |                   | CRA2.type == INNer Second feature |                   | CRA3.type == INNer Third feature |
|------------------------|----------------------------------|-------------------|-----------------------------------|-------------------|----------------------------------|
| CR1                    | CR1                              | CR2               | CR1                               | CR2               | CR1                              |
| ( $F_1$ ,NULL)         | ( $F_1$ , $F_2$ )                | ( $F_2$ , $F_3$ ) | ( $F_3$ , $F_4$ )                 | ( $F_4$ , $F_5$ ) | ( $F_5$ , $F_2$ )                |

TABLE 11. Data Structure of Zsolid1

| CRA0.type<br>==<br>Container | CRA1.type ==<br>INNer First feature |             | CRA2.type ==<br>INNer Second<br>feature |             | CRA3.type == INNer<br>Third feature |
|------------------------------|-------------------------------------|-------------|-----------------------------------------|-------------|-------------------------------------|
| CR1                          | CR1                                 | CR2         | CR1                                     | CR1         | CR1                                 |
| (F6, NULL)                   | (F6,<br>F2)                         | (F2,<br>F7) | (F7,<br>F8)                             | (F8,<br>F9) | (F9,F2)                             |

The reconstructed result is as Figure 11, and a surfacing algorithm is introduced.

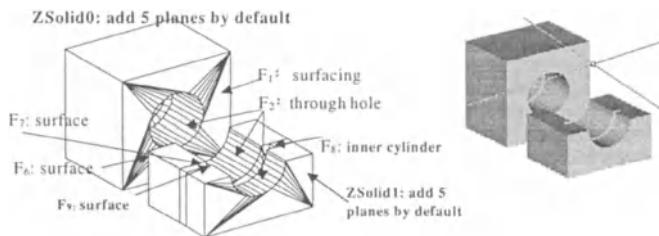


Figure 11. Reconstitution result of the part

## 6. Conclusions and Future Works

With regard to the issues of conceptual product modeling technology, this paper puts forward a hierarchical data structure of geometric solid model and corresponding algorithms based on default geometric reasoning. The proposed methodology is implemented in the software system GPAL (Green Product All-Life Cycle) system, developed based on VC++, MDT, MCAD API etc. A knowledge base called GPAL\_KN is implemented based on Sybase SQL 5.02 for the intelligent design support, which include default logic based feature reconstitution rules and F-F mapping criteria etc.

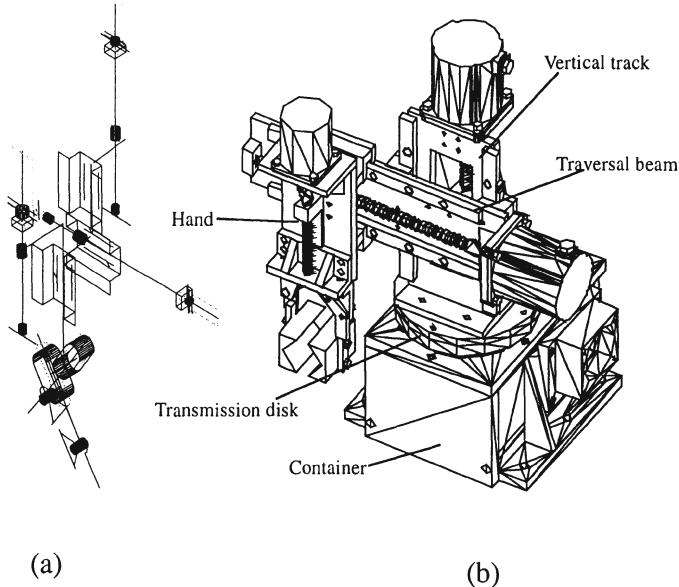
Figure 12 is a typical design example based on the methodology. The main design process is as the follows: 1) after functional design (Xu 1999a) the transmission chain, Figure 12(a), of the mechanism is deduced, and the correlated information of parts and functional faces is recorded in the above mentioned assembly model. Taken any one component, say “container”, Figure 12(b), from the chain for example, algorithm 1 and 2 are exploited to finish the mapping process and part 3D reconstitution. The other component

is reconstructed one by one in the same way to finish the whole product design.

The above example demonstrated a paradigm from concept to embodiment with the following reasoning requirements and information propagation:

- “Top-down” design starting from product sketch layout to detailed components design;
- From product global layout to local part shapes;
- From conceptual skeleton to detailed embodiment design;
- From qualitative/ambiguous to quantitative/definite information based on default rules.

A comparison of the GPAL system with traditional 3D CAD systems is shown in Table 12. The result showed a 20-80 times increase in design efficacy with the new design paradigm implemented in GPAL system.



*Figure 12. Design example for the new paradigm*

The potential limitation of DGR (Default Geometric Reasoning) algorithm and GPAL systems is as the follows:

- The algorithm is based on default logic, the input elements are faces of planar and conic types; the input face should have positioning point and outer vectors, the correlation between faces is not the prerequisite. And

the less the information input, the more complex the algorithms are.

- Till now DGR needs to be more robust and flexible;
- Potential usage of GPAL system is limited by the number of the elements defined in the element library;
- More detailed considerations to be added to the model;
- More friendly in the drag/drop human computer interface is needed.

So the future work should be:

- Combination of default logic with ATMS etc. to make the reasoning process more robust and efficient;
- Heuristics machine learning mechanism should be added to the element library based on design histories;
- The other modules like LCA (Life-Cycle Assessment), assembly /disassembly analysis and sequencing design in the early design phase is now under further testing;
- Part reuse design based on Zsolid is also under further testing.

TABLE 12. Comparison of GPAL with generative 3D CAD systems.

| Systems    |           | 3D CAD (no assembly)                                                                                                     | 3D CAD (assembly)                                                                | GPAL                                                |
|------------|-----------|--------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------|-----------------------------------------------------|
| Schedule   | Part      | 64 parts (24: slots; 24: nuts):<br>70-80 hours                                                                           | 64 parts (24: slots; 24: nuts): 36-48 hours                                      | 40 parts: 1-2 hours                                 |
|            | Assembly  | Need to pre-define position of each part. 100-150 hours to move, 3Drotate each part to fit for the assembly constraints. | Interactive assembling, need: 24-48 hours                                        | Automatic assembling, Need: 1-2 hours               |
|            | Modificat | Time-consuming, equal to re-drawing                                                                                      | 2-4 hours, some change is impossible. I.e. change the model from need redrawing. | Modifying the sketch layout interactively, 1 hours. |
|            | Prepar    | 20-30 hours to calculate 3D coordinate position of each part.                                                            | 1 hour to schedule the product layout.                                           | 0 hours                                             |
| Total cost |           | 190 – 252 hours (students)                                                                                               | 62 – 98 hour (students)                                                          | 3-4 hours (students)                                |

## Acknowledgements

The above-mentioned research work began as early as 1995, when the rudiment design automation system GPAL was developed based on the theory of "generalized mapping" supported by Chinese high-tech project fund. Recently the above methodologies have been implemented in other advanced design systems (Xu, 2001c) in The Design Technology Research Centre in the School of Design of The Hong Kong Polytechnic University. Special thanks are given to the colleagues who dedicated to the development of this system and the methodologies.

## References

- Aldefeld, B:1988, Variation of geometry based on a geometric-reasoning method, *Computer Aided Design* 20(3): 117-126.
- Al-Hakim, L, Kusiak, A and Mathew, J: 2000, A graph-theoretic approach to conceptual design with functional perspectives, *Computer Aided Design* 32(14): 867-875.
- Bouma, W, Fudos, I, Hoffman, C, Cai, J and Paige, R: 1995, Geometric constraint solver', *Computer-Aided Design* 27: 487-501
- Brunetti, G and Golob, B: 2000, A feature-based approach towards an integrated product model including conceptual design information, *Computer Aided Design* 32(14): 877-887.
- Deng, Y-M, Britton, GA and Tor, SB: 2000, Constraint-based functional design verification for conceptual design, *Computer Aided Design* 32(14): 889-899.
- Driling, P, et al: 1996, A unified approach to free-form and regular feature modeling, *Annals of the CIRP* 45(1): 125-128.
- Feng, CX: 2001, Fuzzy mapping of requirements onto functions in detail design, *Computer Aided Design* 33(6): 425-437.
- Finger, S and Dixon, JR: 1989, A review of research in mechanical engineering design, Part 1: descriptive, prescriptive, and computer-based model, model of design process, *Research in Engineering Design* 1: 51-67.
- Harutunian, V, et al: 1996, Decision making and software tools for product development based on axiomatic design theory, *Annals of the CIRP* 45(1): 135-139.
- Jablokow, AG, Vicker Jr, JJ, and Turcic, DA: 1996, Topological and geometric consistency in boundary representations of solid models of mechanical component, *Journal of Mechanical Design* 115: 762-768.
- Krause, FL: 1995, Conceptual modeling for industrial design, *Annals of CIRP* 44(1): 123-128.
- Lamure, H and Michelucci, D: 1996, Solving geometric constraints by homotopy, *IEEE Transaction on Visualization and Computer Graphics* 2: 22-34
- Lee, K and Gossard, DC: 1985, A hierarchical data structure for representing assemblies: Part1, *Computer Aided Design* 17(1): 15-19.
- Lee,JY: 1996, Geometric reasoning for knowledge-based parametric design using graph representation, *Computer-Aided Design* 28: 831-841
- Light, R and Gossard, D: 1982, Modification of geometric models through variational geometry, *Computer Aided Design* 14(4): 209-214.
- Lipson, H and Shpitai, M: 1995, A new interface for conceptual design based on object reconstruction from a single free hand sketch, *Annals of the CIRP* 44(1): 133-136.

- MacCallum, KJ, Duffy, A and Green, S: 1993, An intelligent concept assistant, in H Yoshikawa and EA Warman (eds), *Design Theory for CAD*, North Holland, Amsterdam.
- March, L: 1984, The logic of design, in N Cross (ed.), *Developments In Design Methodology*, John Wiley, London, pp. 265-276.
- Masuda, H: 1993, Topological operators and Boolean operations for complex-BASED non-manifold geometric models, *Computer Aided Design* 25(2): 119-129.
- McKay, A, de Pennington, A and Baxter, J: 2001, Requirements management: a representation scheme for product specifications, *Computer Aided Design* 33(7): 511-520.
- Owen, JC: 1991, Algebraic solution for geometry from dimensional constraints, (eds) *ACM Symposium Foundations of Solid Modeling*, ACM SIGGRAPH, Austin, Texas, pp. 397-407.
- Qamhiyah, AZ, Venter, RD, et al: 1996, Geometric reasoning for the extraction of form features, *Computer Aided Design* 28(11): 887-903.
- Qin, SF, Wright, DK and Jordanov, IN: 2000, From on-line sketching to 2D and 3D geometry: a system based on fuzzy knowledge, *Computer Aided Design* 32(14): 851-866.
- Reiter, R: 1980, A logic for default reasoning, *Artificial Intelligence* 13: 81-132.
- Roy, U, Pramanik, N, Sudarsan, R, Sriram RD and Lyons KW: 2001, Function-to-form mapping: model, representation and applications in design synthesis, *Computer Aided Design* 33(10): 699-719.
- Schulte, M, et al. 1993, Function features for design in mechanical engineering, *Computer in Industry* 23: 15-24.
- Screenivasa, R, et al: 1996, From symbol to form: a frame work for conceptual design, *Computer Aided Design* 28(11): 853-870
- Serrano, D: 1991, Automatic dimensioning in design for manufacturing, *Proceedings of Solid Modeling Foundations and CAD/CAM Applications* ACM SIGGRAPH, Austin, Texas, pp. 371-378
- Shimizu, S and Numao, M: 1996, Constraint-based design for 3D shapes, *Artificial Intelligence* 76: 194-205.
- Shum, SSP, Lau, WS, Yuen, MMF and Yu, KM: 2001, Solid reconstruction from orthographic views using 2-stage extrusion, *Computer Aided Design* 33(1): 91-102.
- Sodhi, R and Turner, JU: 1994, Relative positioning of variation part models for design analysis, *Computer Aided Design* 26(5): 366-378.
- Suh, NP: 1990, *The Principles of Design*, Oxford University Press, New York.
- Tang, MX: 1995, Development of an integrated AI system for conceptual design support, in J Sharpe (ed.), *AI System Support for Conceptual Design*, Springer-Verlag, Berlin, pp. 153-169.
- Toimiyama, T, Umeda, Y and Yoshikawa, H: 1993, A CAD for functional design, *Annals Of The CIRP* 42(1): 143-146.
- Vaco, D and Kim, YS: 1994, Geometric reasoning for machining features using convex decomposition, *Computer Aided Design* 26(6): 477-489.
- Xu, ZG and Huang, KZ: 1999, Product net work model supporting conceptual design, *Chinese Journal of Engineering Graphics* 2: 67-74.
- Xu, ZG, et al.: 1999, Green product design automation based on the theory of generalized mapping, *Proceedings of Second International Conference on CAID&CD'99*, International Academic Publishers, Bangkok, pp. 310-315.
- Xu, ZG, Frazer JH and Tang, MX: 2001, Development of product-oriented user-centered design tool, in P Yunhe, et al (eds), *Fourth International Conference On Computer Aided*

- Industrial Design And Conceptual Design*, International Academic Publishers, Jinan, pp 98-104.
- Yoshikawa, H: 1989, Design philosophy: the state of the art, *Annals Of The CIRP* **38**(2): 579-586

## **LEARNING FROM HUMAN DESIGNERS**

---

*Using protocol analysis to investigate collective learning in design*  
Zhichao Wu And Alex Duffy

*5.8 analogies per hour*  
Pierre Leclercq and Ann Heylighen

*Towards computational tools for supporting the reflective team*  
Andrew W Hill, Andy Dong and Alice M Agogino

## ARTIFICIAL INTELLIGENCE FOR THE DESIGN AND GRADING OF PRECIOUS STONES

TONY HOLDEN AND MATEE SEREARUNO

*University of Cambridge  
United Kingdom*

**Abstract.** This paper describes how the artificial intelligence techniques of rule-based knowledge representation, fuzzy logic and genetic algorithms were integrated to produce a system capable of grading diamonds and gemstones. The design of the cutting and polishing sequences for a rough stone is highly dependent on its innate quality and so the initial grading must be carried out accurately. Part of the grading process involves identifying design opportunities that will raise the economic value of a stone. An interesting aspect of this work is the representation of subjective knowledge as it relates to the appraisal of aesthetic qualities.

### 1. Introduction

A key stage in the design of an ornamental diamond or gemstone is the selection of the optimal grade of rough stone that, when cut and polished, will conform to an intended specification. This specification, designed by an expert lapidarist following examination of the stone in its initial rough state, describes those geometric and visual qualities of the final article that will cause it to be perceived as a beautiful and valuable object. These qualities include its brilliance, colour, scintillation, size and clarity. Presently, this task is undertaken manually with little mechanical assistance. Given that precious stones are both physically intricate and that acquiring the expert knowledge to assess them well takes many years, the absence of automation that can assist with the grading task is holding up not only the advancement of the industry but also the greater availability of precious stones in the wider market.

This paper describes work undertaken to address these issues that brings together the artificial intelligence techniques of rule-based knowledge representation (Cios, et al. 1998), fuzzy logic and genetic algorithms. (Li

and Yen 1995; Goldberg 1989) The result is a tool, entitled “*iGem*”, that can perform the grading task automatically.

This paper will start by summarising the importance of precious stone grading and explaining the context in which it is currently carried out. Following this, the nature of the features or inclusions that can mar a stone’s appearance and influence its grading will be described. The reader will then be taken through the process by which the, presently implicit, knowledge pertaining to precious stone grading was extracted from expert lapidarists, represented using rules and then examined to take advantage of innate patterns and complexity-reducing properties that were able to reduce the size of the rule sets.

In order to improve the precision with which the system was able to grade stones, the parameters present in the rule antecedents were augmented using fuzzy logic. A challenge here was to design the associated fuzzy sets in a way that optimised both the sensitivity and specificity of the rules when performing the grading classification. A means was devised using a genetic algorithm that ‘bred’ optimal fuzzy set parameters, through a process of progressively refining and selecting top-performing rule segments (Cios et al. 1998).

Finally, the performance of the complete system is described and an explanation given as to how, through the availability of this faster automated technique, it is possible to spot opportunities to select, through judicious cutting and removal of inclusions, rough stones that would hitherto have been classified in a low category but can now be ‘*upgraded*’ to a higher and more valuable one. The economic benefits of this are significant; the ability to upgrade a stone by one category can lead to an increase in value of several hundred dollars per carat.

## 2. The Need to Grade Precious Stones

As an object mined from nature, a diamond or gemstone (generically referred to as a precious stone) will normally possess flaws. These flaws, technically known as inclusions or features, can take many forms. These include rough patches, chips, internal fractures as well as dark or clouded regions. Figures 1a and 1b shows the types of inclusion that a stone may possess and the official nomenclature used to describe and illustrate them. Using this, it is possible to draw a diagram of a stone that accurately shows all of its features (Figure 2). Additionally, each stone will have a ‘colour’, such as blue or yellow that describes the tinge or hue present in the stone’s appearance. Pure white (colourless) stones with no imperfections are extremely rare and therefore very valuable (Watermeyer and Michelsen 1994).

|    |                                         |  |                                           |
|----|-----------------------------------------|--|-------------------------------------------|
| .  | Light Pinpoint Inclusion                |  | Large Black or Coloured Inclusion         |
| •  | Group of Pinpoint Inclusions            |  | Dark or Coloured Inclusion with a Cloud   |
| ◆  | Dark Inclusion                          |  | Small Cleavage, Tension or Fracture Crack |
| ◆◆ | Group of Dark Inclusions                |  | Large Cleavage, Tension or Fracture Crack |
| ●  | Cloud of Very Small Light Inclusions    |  | Fringed Girdle                            |
| ◊  | Colourless Crystal                      |  | Cleavage Crack in the Girdle with a Cloud |
| ◊◊ | Group of Colourless Crystals            |  | Colourless Growth or Twinning lines       |
| ⤒  | Colourless Crystal with Cleavage Cracks |  | Coloured Growth or Twinning lines         |

Figure 1a. Graphical symbols for internal features of a precious stone

For the majority of cases therefore, it is necessary for each rough stone to be examined by a skilled lapidarist and a sequence of cuts and polishes designed for it that will maximise its aesthetic qualities according to the intrinsic potential of the stone, the specification of the final design and the need to maximise monetary value through the minimisation of cut material. Given that there are many different types of design for a final cut stone, the task is therefore one of a complex many-to-many mapping from rough stone to finished product.

The traditional process of precious stone grading was developed hundreds of years ago. Despite the fact that geological understanding and mining technology has improved significantly and the jewellery market has flourished in recent decades, gemstone grading techniques remain virtually unchanged and are presently highly manual and subjective, relying upon the knowledge and experience of a human expert (Watermeyer and Michelsen 1994).

According to the most widely accepted standard, the clarity grade for ornamental diamonds consists of nine categories, from the finest to the most inferior category (Lenzen 1983). These are described in Figure 3. Figure 4

shows how the price of 'G' colour (almost white) diamonds varies according to their weight and clarity grade.

|  |                            |  |                          |
|--|----------------------------|--|--------------------------|
|  | Small Natural              |  | Scratch                  |
|  | Large Natural with Trigons |  | Polishing Lines          |
|  | Natural Curved             |  | Percussion Mark          |
|  | Nick in the Girdle         |  | Growth Or Twinning Line  |
|  | Roughened Girdle           |  | Rough (unpolished) Culet |
|  | Fringed Girdle             |  | Damaged Culet            |
|  | Group of Small Pits        |  | Extra Facet              |
|  | Large Pit or Cavity        |  | Roughened Facet Edge     |

Figure 1b. Graphical symbols for external features of a precious stone

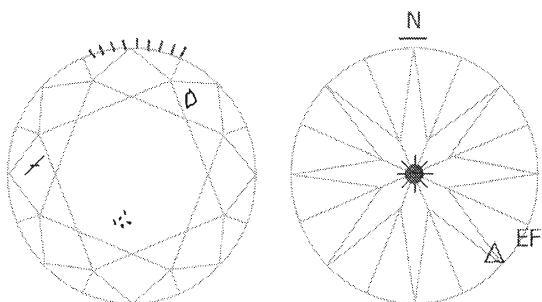
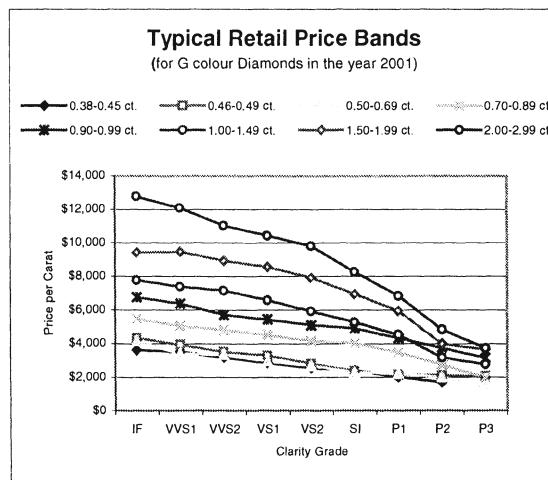


Figure 2. An example of a gemstone drawing

| Clarity     | (Qualitative) Description                                                                                                                                                                                                        |
|-------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| IF          | Internal feature(s) <u>not visible</u> to the trained expert with a <i>10x loupe</i> .                                                                                                                                           |
| VVS1, VVS2  | Very very small feature(s), <u>very difficult</u> to <u>difficult</u> to see for the trained expert with a <i>10x loupe</i> .<br>Size, position and number of internal features determine the distinction between VVS1 and VVS2. |
| VS1,<br>VS2 | Very small internal feature(s), <u>fairly difficult</u> to <u>easily</u> seen for trained expert with a <i>10x loupe</i> .<br>Size, position and number of internal features determine the distinction between VS1 and VS2.      |
| SI          | Small internal feature(s), <u>very easily</u> seen for trained expert with a <i>10x loupe</i> .                                                                                                                                  |
| P1          | Internal feature(s), which are, for the trained expert, <u>difficult</u> to see with the <i>naked eye</i> through the crown side of the diamond.                                                                                 |
| P2          | Large and/or many internal feature(s), which are, for the trained expert, <u>easily</u> seen with the <i>naked eye</i> . The stone's brilliancy is slightly degraded.                                                            |
| P3          | Large and/or many internal feature(s), which are, for the trained expert, <u>very easily</u> seen with the <i>naked eye</i> . The stone's brilliancy is very much degraded.                                                      |

Source: IDC proposals, May 1979 in (Lenzen, 1983)

Figure 3. The relationship between quality grade and features for a precious stone



Source: <http://www.PriceScope.com>, 2001

Figure 4. Suggested retail price band for G-colour diamonds

## 2.1 THE ECONOMIC IMPLICATIONS OF PRECIOUS STONE GRADING

Completely colourless stones without any visible flaws are extremely rare. Additionally, only about 20%-25% of all rough diamonds are suitable for ornamental diamonds (Lenzen 1983). Furthermore, amongst these, up to 60% of the valuable material can be lost during the cutting process. Also, the finest clarity grade can contain 150 times fewer inclusions than that of the poorest. Assuming these distributions, a proportion of less than one-sixth of a percent of the total precious stone population is potentially of the finest, and therefore most valuable, clarity grade (Krishnapillai 1995).

The large difference in the price gap between each clarity grade implies that a misclassification in the clarity grading, could lead to significant monetary loss. Typically, a misclassification of greater than plus or minus one clarity band is unacceptable.

## 2.2 THE CERTIFICATION OF PRECIOUS STONES

In response to the need for a consistent and authoritative means of grading stones, a number of professional gemstone appraisal institutions have been established in many of the major markets centres for precious stones, notably in the USA and Europe. These institutions will evaluate the quality of a precious stone in return for an appraisal fee. They have laboratories equipped with modern instruments and employ well-trained experts in gemmology and mineralogy.

Following examination, the appraisal laboratory will issue a report (or certificate) to accompany the stone as it passes between owners. A gem report is normally an A4 sheet of paper, inscribed with the details of the certified stone, including external measurement, shape, cut, proportions, clarity, colour grade, and comments on its appearance. Figure 5 shows an example of such a certificate.

Gem certificates are becoming increasingly important in the precious stone business. They are the only means to assure potential buyers of the quality of the stone being sold. Without these certificates, a customer only has the trustworthiness of the jeweller from whom the stone is being purchased. There is therefore potential for genuine error and deliberate fraud.

Indeed, it could be postulated that the precious stone grading and the certification are indeed the key to the growth of large-scale gemstone trading. In addition, these certificates could also be used as a means to identify lost or stolen stones, since the intrinsic inclusions for each stone are virtually impossible to reproduce.

According to the Diamond Trading Council, the global precious stone business is currently worth hundreds million dollars a year and is set to

increase. This will increase demand for precious stone appraisal services. Since the existing laboratories are already operating at full capacity, they will be hard pressed to cope with any increase in demand (Lenzen 1983).

It is therefore necessary that some or all of the current manual grading process be automated.

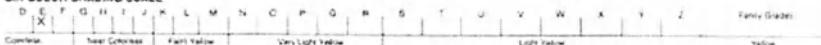
|                                                                                                                                                                                                                                                                                                                                                                                     |                                                                                               |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------|
|  <b>GIA GEM TRADE LABORATORY</b><br>A Division of GIA Enterprises, Inc.<br>A Wholly Owned Subsidiary of the Nonprofit Gemological Institute of America, Inc.                                                                                                                                       |                                                                                               |
| 10412525                                                                                                                                                                                                                                                                                                                                                                            | 5355 Armada Drive<br>Carlsbad, California 92008-4899<br>(619) 436-4430<br>FAX: (619) 436-4434 |
| JUL 01 1998                                                                                                                                                                                                                                                                                                                                                                         | <b>DIAMOND GRADING REPORT</b>                                                                 |
| <small>THE FOLLOWING WERE AT THE TIME OF THE EXAMINATION, THE CHARGED<br/>TRIGRITS OF THE DIAMOND DESCRIBED HERIN BASED UPON 10X<br/>MAGNIFICATION (FULLY CORRECTED TRIPLET LOUPE AND BINOCULAR<br/>MICROSCOPE), CHAMONITE AND MASTERT COLOR COMPARISON<br/>DIAMONDS, U料Violet LAMP, MILLIMETER GAUGE, CARAT BALANCE,<br/>MICROSCOPE, AND ANOTHER INSTRUMENTS AS NECESSARY.</small> |                                                                                               |
| <small>RED SYMBOLS INDICATE INTERNAL CHARACTERISTICS; INCLUSIONS;<br/>GREEN SYMBOLS INDICATE EXTERNAL CHARACTERISTICS; IRREGULARITIES;<br/>SYMBOLS INDICATE TYPE, POSITION AND APPROXIMATE SIZE OF<br/>CHARACTERISTICS. DETAILS OF FINISH ARE NOT SHOWN. DIAGRAM MAY<br/>BE APPROXIMATE.</small>                                                                                    |                                                                                               |
| <b>KEY TO SYMBOLS</b>                                                                                                                                                                                                                                                                                                                                                               |                                                                                               |
| FEATHER<br>CLOUD<br>NATURAL                                                                                                                                                                                                                                                                                                                                                         |                                                                                               |
| <b>SHAPE AND CUTTING STYLE</b>                                                                                                                                                                                                                                                                                                                                                      | ROUND BRILLIANT                                                                               |
| Measurements                                                                                                                                                                                                                                                                                                                                                                        | 5.04 x 5.10 x 3.25 MM                                                                         |
| Weight                                                                                                                                                                                                                                                                                                                                                                              | 0.51 CARATS                                                                                   |
| <b>PROPORTIONS</b>                                                                                                                                                                                                                                                                                                                                                                  |                                                                                               |
| Depth                                                                                                                                                                                                                                                                                                                                                                               | 69.7 %                                                                                        |
| Table                                                                                                                                                                                                                                                                                                                                                                               | 58 %                                                                                          |
| Girdle                                                                                                                                                                                                                                                                                                                                                                              | MEDIUM TO SLIGHTLY THICK, FACETED                                                             |
| Culet                                                                                                                                                                                                                                                                                                                                                                               | NONE                                                                                          |
| <b>FINISH</b>                                                                                                                                                                                                                                                                                                                                                                       |                                                                                               |
| Polish                                                                                                                                                                                                                                                                                                                                                                              | VERY GOOD                                                                                     |
| Symmetry                                                                                                                                                                                                                                                                                                                                                                            | GOOD                                                                                          |
| <b>CLARITY GRADE</b>                                                                                                                                                                                                                                                                                                                                                                | VS1                                                                                           |
| <b>COLOR GRADE</b>                                                                                                                                                                                                                                                                                                                                                                  | E                                                                                             |
| Fluorescence                                                                                                                                                                                                                                                                                                                                                                        | NONE                                                                                          |
| <b>COMMENTS</b>                                                                                                                                                                                                                                                                                                                                                                     | PINPOINTS ARE NOT SHOWN.                                                                      |
| ORIGINAL                                                                                                                                                                                                                                                                                                                                                                            |                                                                                               |
|  <b>GEM TRADE LABORATORY</b><br><i>GIA Gem Trade Laboratory</i>                                                                                                                                                                                                                                  |                                                                                               |
| <b>GIA CLARITY GRADING SCALE</b><br><br>This report is not a guarantee, valuation or appraisal. The recipient of this report may wish to consult a credentialed jeweler or Gemologist about the importance and interrelationship of cut, color, clarity and carat weight.                        |                                                                                               |
| <b>GIA COLOR GRADING SCALE</b><br><br>Copyright © 1989 - 1997 GIA Gem Trade Laboratory                                                                                                                                                                                                           |                                                                                               |
| <b>NOTICE: IMPORTANT LIMITATIONS ON REVERSE</b>                                                                                                                                                                                                                                                                                                                                     |                                                                                               |

Figure 5. An example of a gem certificate from Gemmological Institute of America

### 3. The Development of an Automatic Grading System

The property of clarity, unlike the other properties of the stone such as weight or geometry, cannot be measured directly. On the contrary, it can be seen that a clarity grade is a largely subjective assessment. The following sections describe how, using a combination of artificial intelligence and related techniques a system, entitled “*iGem*”, was constructed to automate this process. An interesting aspect of this work is the codification of initially subjective and aesthetic factors. Each of the stages of development for *iGem* will be described in the proceeding sections (Pyle 1999). However, to orientate the reader they are briefly summarised here:

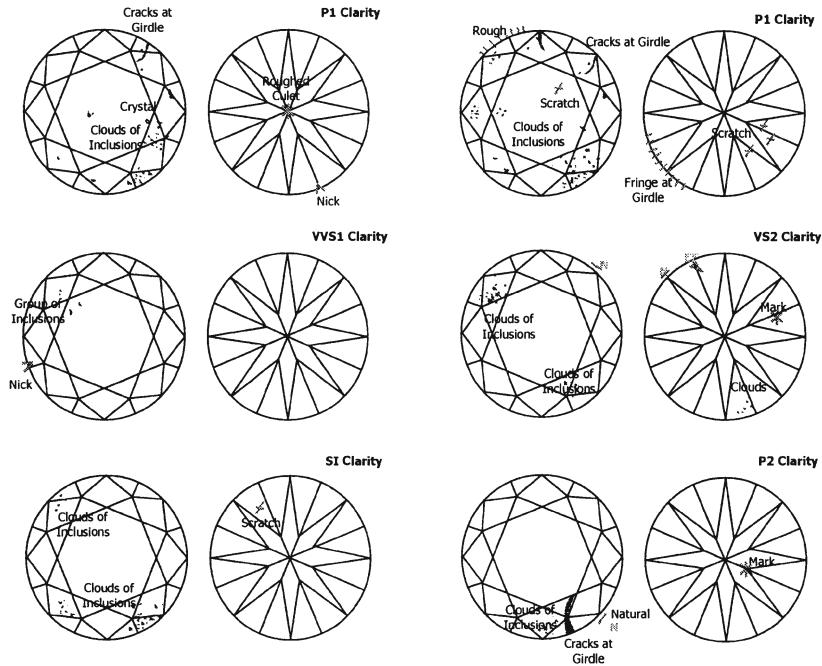
1. *Generation of raw data.* Using known distributions of precious stone features, a series of 503 ‘virtual gemstones’ were generated by computer. The purpose of this was to provide a meaningful number of examples of each type of feature for the experts to assess in the next stage.
2. *Knowledge elicitation from experts.* Here, a team of four expert lapidarists were shown the gemstone drawings and asked to classify each into the recognised nine grades.
3. *Statistical analysis for rule simplification.* An analysis of the location and nature of the stones’ features showed that, in fact, only a minority of the some thirty or so features available actually correlated with a final classification. This knowledge allows for simpler rules and a smaller, computationally more efficient, ruleset to be developed.
4. *Rule generation using a decision tree algorithm.* Having obtained a simplified correlation between features and grade, a set of IF-THEN rules were inferred using a variant of Quinlan’s ID3 algorithm.
5. *Improvement of rules with fuzzy logic.* In order to mitigate against the effect of imprecision in the original dataset and also the loss of financial opportunity resulting from borderline classification, the rules’ logical operators were replaced with comparative linguistic terms based upon a fuzzy representation.
6. *Generation of fuzzy sets with a genetic algorithm.* To discover the definitions for the fuzzy sets, a genetic algorithm was used to generate and test candidates that led to rules providing a useful and discriminating classification.
7. *System evaluation and test.* Using examples held back from the original set of virtual stones generated, the system performance was evaluated.

IF-THEN rules are chosen as the preferred method for representing the domain knowledge extracted from the raw precious stone data because such statements are clearly defined, highly structured and comprehensible to people. Fuzzy logic and genetic algorithms, employed in the later process of

generating *fuzzy* rules, is useful when dealing with the uncertainties and imprecision of precious stone data during the highly subjective data acquisition process. Finally, a Knowledge Discovery in Databases (KDD) technique is employed as a framework to integrate and coordinate the various techniques and implementation stages. These include the two important phases characterising the KDD framework – data preparation and data mining (Fayyad, et. al. 1996).

#### 4. Generation of Raw Data and Virtual Stones

In order to provide the experts with a sufficiently large number of examples to classify and so allow a statistically meaningful correlation to be drawn, a total of 503 ‘virtual stones’ were generated by computer. Figure 6 shows a sample of these.



*Figure 6.* Computer generated examples of ‘virtual’ stones

Each stone diagram was described by a variable length dataset where each element contained sufficient information to describe each feature. Datasets describing stones of a higher grade (containing less flaws) will

generally contain fewer elements. Each of these data elements was represented in relational form, Figure 7.

The use of artificially generated ‘virtual’ stones as opposed to diagrams of real stones was preferred for the following reasons:

1. Data for real stones is often kept confidential and so obtaining a large enough sample would be difficult.
2. It was necessary to ensure that each grade was sufficiently represented in the total population and that, within the grade, there was a good distribution of different features. Given the rarity of some grades of stone, it would require excessive effort to obtain reliable data. The production of such data by computer ensures generation of a wide and meaningful spread of examples.

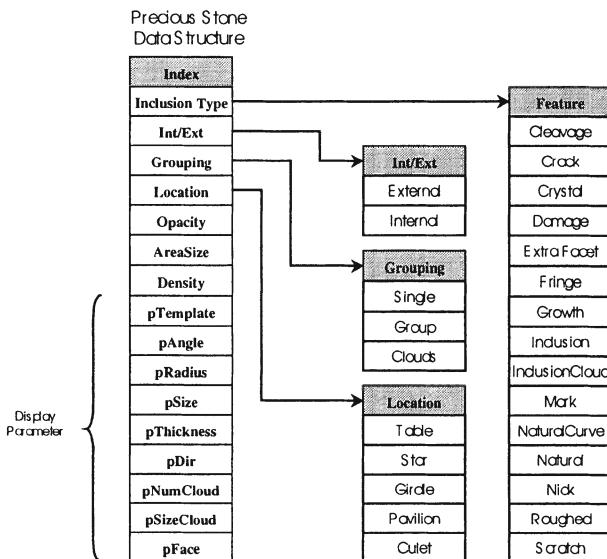


Figure 7. The relational data element used to represent a stone's feature

The primary source for feature data was the book “*Diamond Grading*” by Lenzen (Lenzen 1983). This contains sample diagrams for stones of various clarities as it is a textbook primarily intended to be used in the diamond grading course at the German Gemmological Training Centre. As a result, the reliability of this data source is highly trustworthy.

## 5. Knowledge Elicitation from Experts

Having generated datasets for the 503 example stones, the visual representations were then produced and presented to a team of four expert lapidarists who were asked to classify each drawing. In order to ensure the quality of the dataset, at least three out of four experts had to be in agreement on the clarity classification before the result was accepted (Pyle 1999).

## 6. Statistical Analysis for Rule Simplification

This exercise provided a database containing tables describing the features of a significant number of stones together with their corresponding expert classifications. In itself, this was sufficient to generate a set of rules, representing the experts' hitherto implicit knowledge that would allow classification of a precious stone. Each rule would be of the classic IF...THEN form, viz:

**IF (Table > 1.45) AND (Star <= 3.1) AND ..... THEN Category = VVS1**

With the present dataset however, there are a potentially large number of antecedents for each rule. This can lead to both computational slowness and also uneconomic expression of knowledge. In an attempt to discover opportunities for greater economy of expression, a statistical analysis was performed that related feature types and location with eventual classification. Figures 8 and 9 show the results of this.

This analysis reveals that many attributes are sparsely distributed. For example, the Nick, Natural, Percussion Mark, Twinning Line, Fringe-at-Girdle, and Damaged Culet have virtually no effect in the clarity identification. The prominent features are, however, 'Inclusion' and 'Cleavage', both of which are of features internal to a stone.

Although inclusions are very prominent, however, this feature suffers from widespread fluctuation in value. In fact, a large degree of swing seen in many data records was discovered to be irrelevant to the respective clarity grade. Furthermore, inclusions are often hard to distinguish from other features. As a result, the 'inclusion' feature is not practical as a way of classifying a stone.

However, the location aspect has only five attributes, namely *Table*, *Star*, *Girdle*, *Pavilion* and *Culet* (these being recognised locations on a stone's surface) and, of itself, appears to provide almost sufficient information, along with feature size, to allow a grading. In the final scheme, the two additional domain variables of 'Internal' and 'External' were introduced and

refer to inclusions that exist inside the stone or on one of its facets respectively.

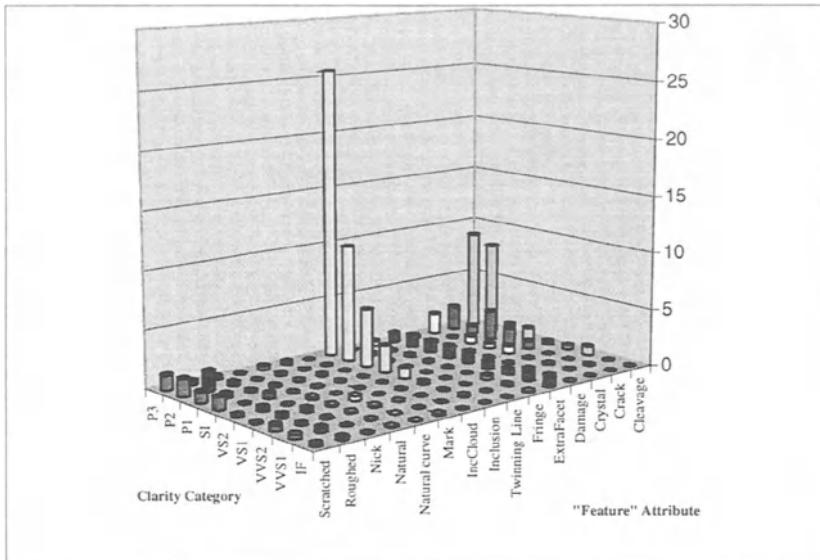


Figure 8. The relationship between a stone's feature types and its final grade

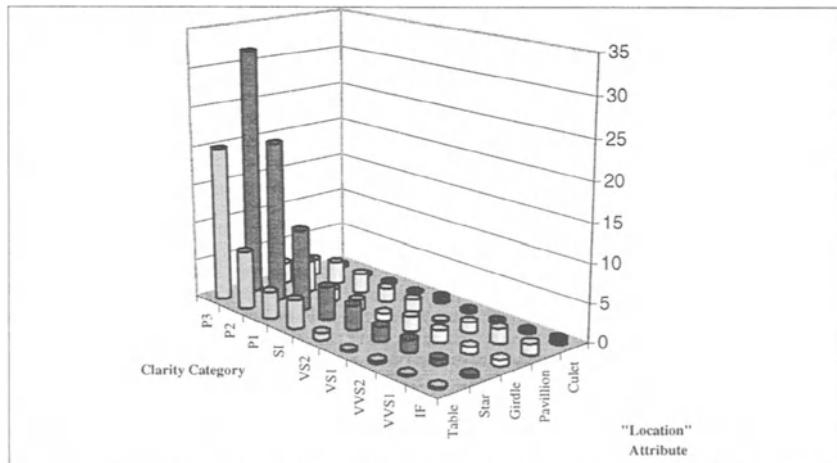


Figure 9. The relationship between the location of a stone's features and its final grade

The benefit of this analysis is clearly that rules may now be simplified since there is now only a maximum of seven potential antecedents to a classification rule. In addition, the location attribute is more recognisable and can be easily distinguished with manual inspection, promoting comprehensibility of the discovered ruleset.

## 7. Rule Generation Using a Decision Tree Algorithm

Having identified those antecedents necessary and sufficient to produce a useful ruleset, the ruleset itself was produced by means of a decision tree based on the ID3 algorithm (Quinlan 1986). *Iterative Dichotomiser 3* or *ID3* is a well known algorithm based on a decision tree approach. ID3 uses information entropy as a factor to measure the degree of chaos in a data set and tries to partition the data set in such a way that the entropy gain is maximal.

Decision tree algorithms basically work by partitioning the data sets in two halves. i.e. separating the data items that are satisfied and not satisfied with respect to the split criterion. The partitioning is continued recursively upon the split node until termination criteria are met. This occurs when all (or most of) the data items associated with that end nodes share the same characteristics. As a result, by tracing from the leaf node back to the root node, the combination of such conditions forms a rule that describes the properties of the data items at the leaf node.

In Figure 10, the first condition “Table > 1.45” separates the data items that possess a “Table” feature whose value is greater than 1.45 and assigns them to a child node. The remaining data items are allocated to another child node. The same algorithm is recursively applied to the two child nodes, resulting in further splits with conditions “Star <= 3.1” and “Girdle > 2.7”.

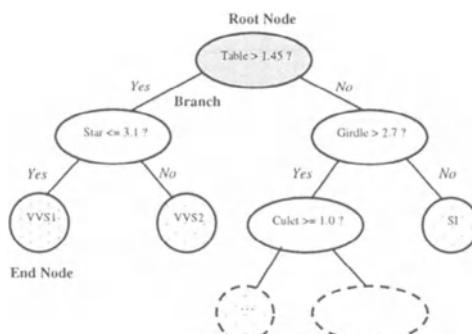


Figure 10. Illustrating classification using a decision tree

Tracing from the leaf nodes upwards, the classification rules may be derived. For example, the rule to classify a VVS1 stone is:

**IF (Star <= 3.1) AND (Table > 1.45) THEN Category = VVS1**

## 8. Improvement of Rules with Fuzzy Logic

The rules developed thus far were found to give satisfactory performance when classifying rough stones. However, the rules used were only as good as the examples from which they were generated. If another stone appears which is exactly the same as one of the training examples, it will be guaranteed to be categorised, according to the experts' consensus, correctly. However, no two stones are ever the same and indeed, expert lapidarists use a degree of subjectivity in their judgement; one expert may judge an inclusion (say a 'cloud') to be of size 1.2, whereas another may assert it to be of size 1.3. There is a risk, therefore, that rules may incorrectly categorise some stones, particularly those on the borders of two grades.

There is another reason why it is helpful to identify borderline stones. These represent stones that, through judicious cutting of imperfections, could be raised to a higher grade. In fact, according to the value table shown in Figure 11, a mis-classification of one clarity grade could end up in 3 to 23% loss of profit.

|              | IF      | VVS1             | VVS2             | VS1              | VS2              | SI                | P1                | P2                 | P3                |
|--------------|---------|------------------|------------------|------------------|------------------|-------------------|-------------------|--------------------|-------------------|
|              | \$7,770 | \$7,375          | \$7,145          | \$6,573          | \$5,916          | \$5,285           | \$4,551           | \$3,470            | \$2,760           |
| Increase (%) | -       | \$395<br>(5.08%) | \$230<br>(3.11%) | \$572<br>(8.00%) | \$657<br>(9.99%) | \$631<br>(10.67%) | \$734<br>(13.89%) | \$1081<br>(23.75%) | \$710<br>(20.46%) |

Figure 11. Illustrating the relationship of stone value to grade ('G' colour, 1.0 carat stone)

To ameliorate this possibility, a fuzzy data representation was introduced for the rule parameters. This replaces the conventional logical operators (e.g.  $>$ ,  $<$ ,  $=$ ) with comparative linguistic terms. This fuzzy logic operation gives a degree of membership, rather than a yes/no result (Li and Yen, 1995). As a result, all data has a better chance of being covered by the rule set, Figure 12.

In the implementation, three fuzzy operators were used, namely *Less-Than*, *More-Than* and *Range-Of*. The fuzzy logic expressions contained in the rules were represented using the graphs shown in Figure 13.

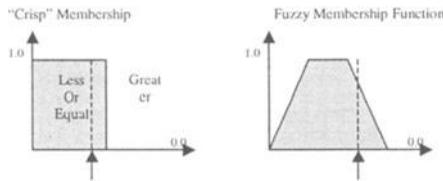


Figure 12. Crisp vs fuzzy data representation

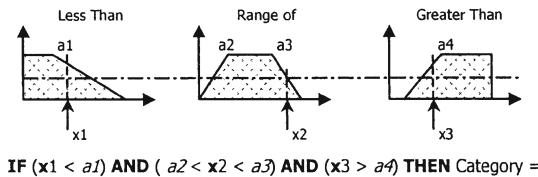


Figure 13. Illustrating operators using fuzzy linguistic terms

Each operator is described by a 4-point trapezoid, described by the points P1, P2, P3, P4, as shown in Figure 14. The slope of the function gradient can easily be calculated from these values. Each of a rule's fuzzy operators is then represented by a simple binary string as shown in Figure 14.

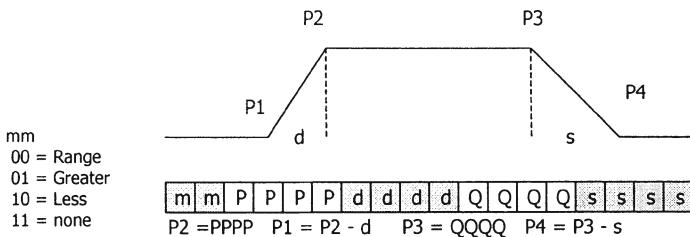


Figure 14. The encoding of fuzzy parameters for an operator

A complete rule is represented by a concatenation of individual strings, with 126 bits (18 bits/feature x 7 features) in total, Figure 15.

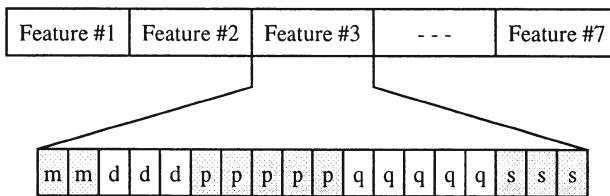


Figure 15. The encoding of a complete rule

## 9. Generating the Fuzzy Sets with a Genetic Algorithm

The key to optimal performance of the rulesets lies in the values chosen to represent the trapezoid describing the fuzzy operator. Rather than try to ‘guess’ these or to elaborate them by some manual technique, they were derived using a genetic algorithm.

The genetic algorithm works on the principle of the survival of the fittest. The evolution continuously exchanges and mutates “genes” (in this case, the parts of the feature bitstring representing a trapezoidal apex) amongst a population for some period of time, until a candidate gene evolves which is judged to be the best survivor (Goldberg 1989). Survival of the fittest in this case equates to how well the rule of which the parameter is a part can discriminate between grades of stone. The process can be summarised as follows:

1. The system is seeded with a set of randomly generated bitstrings, using the encoding scheme as described above. These bitstrings (or population) thus represent a set of candidates for possible fuzzy membership functions to be incorporated in the fuzzy rules.
2. Then the fitness of each bitstring is determined. This is done in two steps. Firstly, the bitstring is decoded (using the same encoding scheme) to construct the corresponding fuzzy membership functions. Then, these functions are used to construct the fuzzy rule set in the decision-tree generation process described earlier. Finally, the generated rule set is tested against the training data, in which the degree of success is considered to be the discrimination quality of the bitstring.
3. Once all the bitstrings have been evaluated for their fitness, the conventional genetic operations of *reproduction*, *crossover*, and *mutation* are used to generate new bitstrings. Higher fitness candidates will be given a greater chance to survive in the next generation. The crossover function randomly shuffles the “gene” (i.e. a portion of bitstrings) among the selected candidates, in the hope that the good part of one bitstring would integrate with another good part of another

bitstring. Finally, mutation functions are employed to blatantly complement sections of bitstrings in order to provide a mechanism that prevents higher-fitness bitstring genes from dominating an entire generation.

4. This evolution process will return to the evaluation step, and the process will repeat itself until the termination criterion is met (i.e. the performance of the fuzzy rule set, generated from the “best” bitstring and the decision tree, is sufficiently satisfied).

An example of fuzzy rule set to classify a stone is shown in Figure 16.

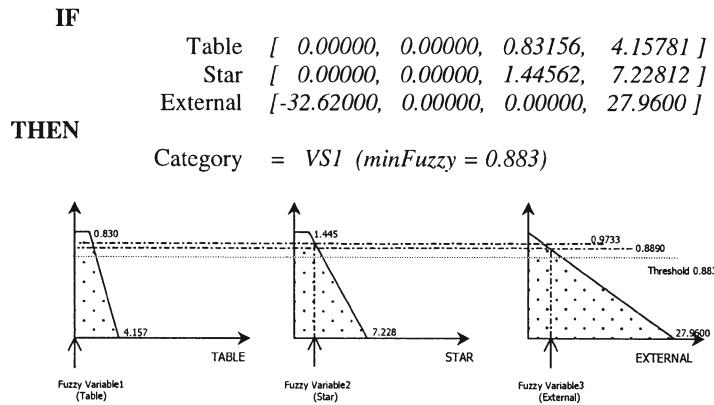


Figure 16. Illustrating fuzzy rule quantification and reasoning

## 10. System Evaluation and Test

Figure 17 shows how the system graded a number of *unseen* stones compared to the classifications given by experts. It was found that the number of rules discovered by the decision tree algorithm grew proportionally with the size of training dataset. However, a set of 200 data samples describing mixed categories of stones was found experimentally to be a reasonable size to generate a useful number of rules since the percentage of correct classification with more training data samples increased at a much slower rate above this number. In total, a set of 58 fuzzy rules (instead of 80 rules from the classical decision tree ID3 algorithm) was obtained from the rule generation process. Note that the number of marginal and incorrect classifications, as well as that of the correct classified category, is shown.

From testing with the 303 remaining unseen data vectors, the table shows that, the VVS1, VS2, SI, P1, and P2 classifications are moderately accurately classified, whereas the VVS2 and VS1 clarity grades are less so. Most of the VS1 samples were wrongly identified as VS2, since their properties are hardly distinguishable. In other words, the rulesets for VS1 category are inefficient in comparison to the neighbouring category of VS2. Using the measures of sensitivity, specificity and predictive accuracy espoused by Cios (Cios et al. 1998), we can measure the performance of *iGem*. From ten repeated experiments, the overall average percentage of completely correct categorisation is 57.80%, although a percentage of 89.87% was achieved when a tolerance of plus or minus one clarity grade is allowed.

The measure of *specificity* is also used to determine how well the ruleset eliminates neighbouring category data from the classification. The average specificity of *iGem* is an impressive 95.0%, although the rulesets for VS1 and P1 are both below the average. The other measure used is *predictive accuracy*, which calculates the overall precision of the classification. The average accuracy for this given set of testing data is found to be 91.0%.

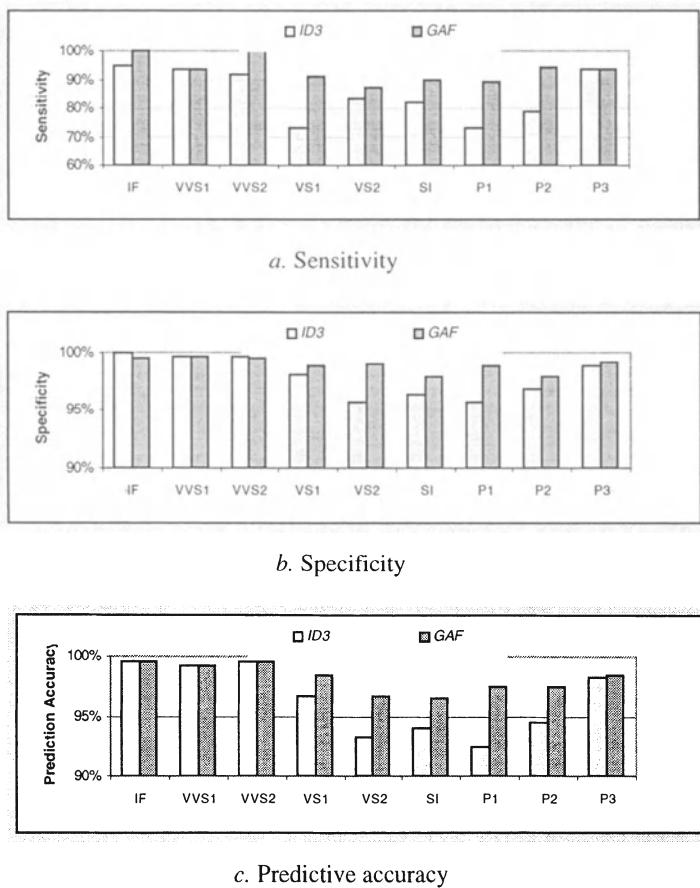
| Expert Grading | Classified by <i>iGem</i> as: |      |      |     |     |    |    |    |    |
|----------------|-------------------------------|------|------|-----|-----|----|----|----|----|
|                | IF                            | VVS1 | VVS2 | VS1 | VS2 | SI | P1 | P2 | P3 |
| IF             | 27                            | 2    |      | 1   | 2   |    |    |    |    |
| VVS1           | 1                             | 7    | 1    | 1   | 5   |    |    |    |    |
| VVS2           |                               | 1    | 3    | 1   | 4   |    | 1  |    |    |
| VS1            |                               | 2    | 4    | 8   | 8   | 1  |    |    |    |
| VS2            |                               |      | 1    | 11  | 37  | 9  | 4  | 1  |    |
| SI             |                               |      |      | 3   | 7   | 21 | 3  | 5  |    |
| P1             |                               |      |      | 4   | 1   | 4  | 15 | 9  | 1  |
| P2             |                               |      |      | 1   | 1   | 1  | 9  | 28 | 8  |
| P3             |                               |      |      |     |     |    | 1  | 8  | 30 |

Figure 17. Sampled classification performance of *iGem*, evaluated with 303 “unseen” data vectors

When dealing with imprecision in the data, an evaluation with a range of datasets deliberately distorted to a degree between 1 and 15% reveals that the fuzzy rulesets are consistently more superior in performance to conventional “crisp” rulesets. For illustration, the classification performance with 10% distortion level is shown in Figure 18. Both “crisp” (generated by

the ID3 algorithm) and “Fuzzy-Genetic” rule sets are trained with 200 data samples and then tested with 4,000 samples of 10% distorted “seen” data

It can be seen that a significant increase in sensitivity, especially for those stones in the VS1, VS2, SI, P1 and P2 categories is observed, Figure 18a. In addition, the specificity, Figure 18b, which is the capability to reject and avoid classification of irrelevant samples, is noticeably improved in those mid-range categories. Such improvement is also seen in a higher percentage of predictive accuracy, as shown in Figure 18c.



*Figure 18. Evaluation with distorted datasets.*

## 11. Conclusions and Opportunities for Economic Upgrade

Figure 19 shows a screen dump of *iGem*. We believe the system has performed to a strong degree of accuracy. Whilst there is room for improvement in the specificity and accuracy of categorisation, the system would be of significant practical value in automating the grading of the majority of low-to-medium value stones. High value stones (those in the IF and VVS1 categories) would probably always require human grading.

From the results, it can be seen that there are two opportunities for economic benefit arising from the system's existence. The first is simply that grading of many stones can now be automated, thereby removing the manual bottleneck described at the start of the paper.

The second comes from the fact that 'borderline' stones – those that with careful removal of damaging features could be made more valuable – can be more easily spotted and the steps necessary to perform the upgrade clearly stated. For example, the ruleset in Figure 20 shows that a stone with a VVS1 clarity grade could be upgraded to IF clarity, if its table inclusions were re-polished to be smaller than 0.22 and it had no internal inclusions.

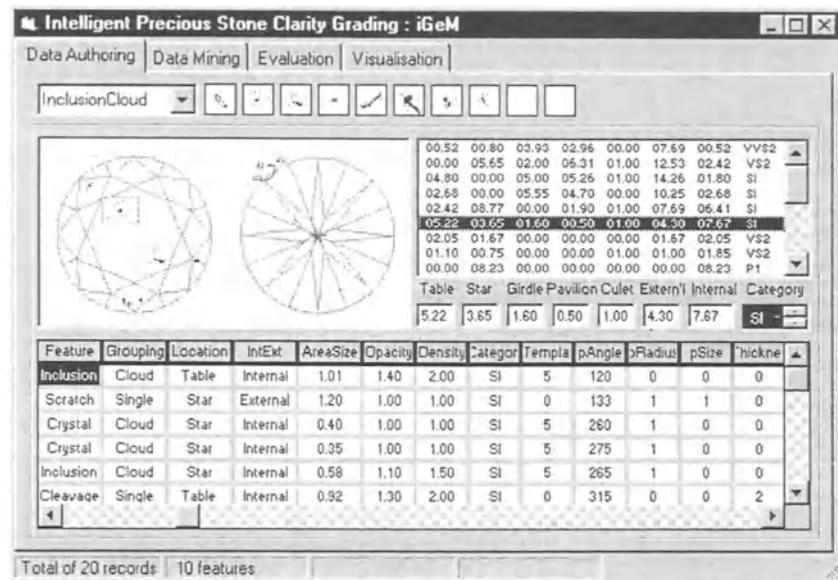


Figure 19. An *iGem* screen, illustrating the data authoring tool

| Rule | Table                | Star                | External                | Internal         | Grade |
|------|----------------------|---------------------|-------------------------|------------------|-------|
| 1    | Table <= 0.22        | 0.00 < Star <= 0.84 | 0.30 < External <= 5.53 | Internal <= 0.00 | IF    |
| 2    | 0.22 < Table <= 1.05 | Star <= 0.84        | 0.00 < External         | Internal <= 0.18 | VVS1  |

Figure 20. Illustrating opportunities for upgrading a stone

Further work in developing *iGem* is investigating the potential for direct interfacing with X-ray scanning machines. This technology uses x-ray tomography to non-destructively investigate the internal flaws inside a stone. As a result, the manual subjective data acquisition process is eradicated and replaced with the automatic method that provides a richer and more consistent data set for the knowledge discovery process. The inclusions can be described in three dimensions, as opposed to the current two dimensions. In principle, the integration of the automatic scanning machine and the *iGem* grading software would result in an automated system that enables a customer to easily verify the gem stone clarity grade. Once fully developed, any stone can easily be assessed, even in a jewellers shop. This would help increase confidence levels in diamond and gemstone trading.

For industrial scale production, integration with “shape-fitting” software would allow a rough stone to be designed in a way that permitted the shape, pattern and other of the stone’s final properties, including the clarity grade, to be predicted, even before the stone has actually gone through the cutting process. Once the design was made, a computer-controlled robot could perform the cutting task automatically. The obvious benefit is that the manufacturing plant becomes more flexible and able to produce a product to a specification that is currently in demand, and so increase business opportunities.

## References

- Cios, K, Pedrycz, W and Swiniarski, R: 1998, *Data Mining Methods for Knowledge Discovery*, Kluwer Academic Publishers, London.
- Fayyad, UM, Piatetsky-Shapiro, G and Smyth, P: 1996, From data mining to knowledge discovery : an overview, in Fayyad, UM, Piatetsky-Shapiro, G, Smyth, P and Uthurusamy, R (eds), *Advances in Knowledge Discovery and Data Mining*, AAAI Press / The MIT Press, pp. 1-34.
- Goldberg, D: 1989, *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, Reading.
- Krishnapillai, A: 1995, *An Automated System for the Processing of Precious Stones*, Department of Engineering, University of Cambridge, Cambridge
- Lenzen, G: 1983, *Diamonds and Diamond Grading*, Butterworths, London.
- Li, HX and Yen, VC: 1995, *Fuzzy Sets and Fuzzy Decision-making*, CRC Press.
- Pyle, D: 1999, *Data Preparation for Data Mining*, Morgan Kaufmann, San Francisco.

- Quinlan, JR: 1986, Induction of decision trees, *Machine Learning* **1**: 81-106.  
Watermeyer, B and Michelsen, S: 1994, *The Art of Diamond Cutting*, Chapman & Hall,  
London.

## USING PROTOCOL ANALYSIS TO INVESTIGATE COLLECTIVE LEARNING IN DESIGN

ZHICHAO WU AND ALEX DUFFY

*University of Strathclyde  
UK*

**Abstract.** It is postulated that in a collaborative design environment, when agents interact with each other, they may learn from each other. In this paper, the phenomena of collective learning in team working are investigated using protocol analysis. Through such an investigation, a model of collective learning in design has been extended from an existing model of learning in design.

### 1. Introduction

Design and learning are two interlinked activities (Persidis and Duffy 1991). Designers learn during the design process (Persidis and Duffy 1991; Sim 2000). However, most previous research in learning in design is confined to investigating individual learning, that is, how an individual agent learns (Persidis and Duffy 1991; Duffy and Duffy 1996; Sim 2000; Gero 1990; Smithers and Troxell 1990; Grecu and Brown 1996a, 1996b; Reich and Fenves 1989). In a design environment like team or distributed working, agents interact with and from each other. If the phenomena of collective learning in design exist, what is the nature of it? That is, what can be learnt collectively? When will collective learning occur? How do agents learn collectively? This paper reports an investigation into these phenomena.

The meaning of “collective” is defined in the *New Oxford Dictionary of English* as “belonging or relating to all the members of a group” (Pearsall 1998). It is suggested that there are different forms that make collective learning unique from individual learning:

- Combining knowledge: The knowledge in different agents can be combined and new knowledge can be extracted and learned from the combination.
- Common Learning: All agents in a team or a group can learn the same thing at the same time.

- Mutual evolution: When agents interact with each other in a collaborative design environment, they can learn from each other. Such interaction can result in such a mutual knowledge evolving process that could not be gained through individual learning.

To be able to understand the nature of collective learning in design, current related work in organisational learning and computer science research community is presented. Based on these theories, an understanding of collective in design is derived. A protocol analysis of a design team is carried out in order to evaluate the hypothesis of collective learning in design. Finally, a model of collective learning, extended from a current model of learning in design (Sim 2000), is developed from the analysis that to some degree reflects the nature of collective learning in design. It is intended that the model will be elaborated upon through further protocol studies and the development of an agent-based, collective learning, computing environment.

## 2. Related Work

Learning in design has been investigated within the artificial intelligence in design community for over a decade. Sim developed a model of learning in design based on individual learning and described what is learnt, when learning occurs and how knowledge is learnt (Sim 2000). However, the model does not explain the phenomena of how a group of agents learn from each other in a context such as team working.

Brown and Grecu have summarised some issues in agent-based learning, such as triggers, elements, knowledge types, availability of knowledge, methods and consequences (Grecu and Brown 1996b). Their agent-based learning system was developed using Single Function Agents (SiFAs) (Grecu and Brown 1996a; Dunkus et al., 1995). Karni et al (1992) considered learning in concurrent engineering design. They focus acquiring knowledge from production data that would facilitate better design and improve production management. Their method also enabled learning from data in the recycle stage that may be used in the early stages of design. Their work presents some aspects of collective learning in design.

Duffy and Duffy developed the concept of *Shared Learning* (1996). Traditional computational design systems reflect viewpoints defined by knowledge engineers and present perceptions of the knowledge needs of designers. In shared learning, the designer and computing system, acting as a learning assistant, co-learn. That is, a designer learns new knowledge from a learning assistant that automatically learns and presents previously implicit, and therefore unrepresentative knowledge. *Shared Learning*

supports the interaction of learning between a designer and a computer, and presents an aspect of collective learning.

While the above work presented aspects of collective learning in design, they haven't developed a comprehensive model of collective learning in design.

However, effort has been directed at investigating collective learning or related work in the areas of organisational theory and computer science, particularly in Distributed Artificial Intelligence (DAI).

Within the last few decades, there has been considerable research on individual learning, team learning, and organisational learning in organisational studies (Neergaard 1994; Cross and Israelit 2000). *Team learning* is oriented towards an effort to improve collaboration, integrate specialised functional and technical knowledge, and increase the responsiveness to demanding stakeholders (Cross and Israelit 2000). Team learning focuses on determining how knowledge enters a team and also how such knowledge may come to be shared by a group as a whole (Cross and Israelit, 2000). An organisation may also learn. Organisational learning is understood as members of the organisation respond to changes in the internal and external environments of the organisation, and thus to change the norms, strategies and assumptions in the organisation (Argyris and Schön 1977). The organisation itself does not learn. Members of the organisation, as agents of the organisation, undertake the task of learning, and they share their knowledge, which is said to result in organisation learning. Team learning and organisation learning studies how a group of agents learns instead of one, which is related to collective learning in design.

Organisational learning has been investigated in the computer science research community, particularly in multi-agent systems (Ho and Kamel 1998; Ono and Fukumoto 1997) to provide adaptability and robustness to the organizations and their systems. Through organisational learning the role of the agents, the knowledge states of the agents, the coordination strategies, etc, are learnt. Multi-agent learning systems in DAI are developed, such as systems in (Prasad et al. 1997; Ono and Fukumoto 1997; Sen et al. 1994; Haynes and Sen 1996; Prasad and Lessor 1997). Those agent-based learning systems in DAI, orienting to investigate how agents learn in the system, can be used as an aid to develop computational design system with collective learning capability.

### 3. The Concept of Learning

Different researchers have different definitions for the concept of learning in different fields. For example, from neurobiology, Shepherd (1988) stated that *learning is an adaptive change in behaviour caused by experience*. In psychology, *learning* was defined from six perspectives as a result of an (Marton and Booth): Learning as increasing one's knowledge; Learning as memorising and reproducing; Learning as applying; Learning as understanding; learning as seeing something as a different way; and Learning as changing a person. In the view of organisational study, learning is understood in two levels: operational and conceptual. Learning address (Kim 1993): (1) the acquisition of skill or *know-how*, which implies the physical ability to produce some action, and (2) the acquisition of *know-why*, which implies the ability to articulate a conceptual understanding of an experience. To be able to carry out effective action, both know-how and know-why are needed. In the artificial intelligence community, Simon (1983) states that:

“Learning denotes changes in the system that are adaptive in the sense that they enable the system to do the same task or tasks drawn from the same population more efficiently and more effectively the next time”.

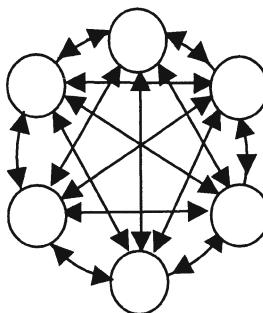
In the area of engineering design, Persidis and Duffy (1991) argued that designers learn when they encounter knowledge that is sufficiently different from their present state of knowledge. Learning in design is understood as the change of the human's state of knowledge that can have a direct influence on the ability of a human to solve a problem. In this paper, learning is defined as a process of gaining and storing knowledge through interaction with other agents.

### 4. Collective Learning in Design

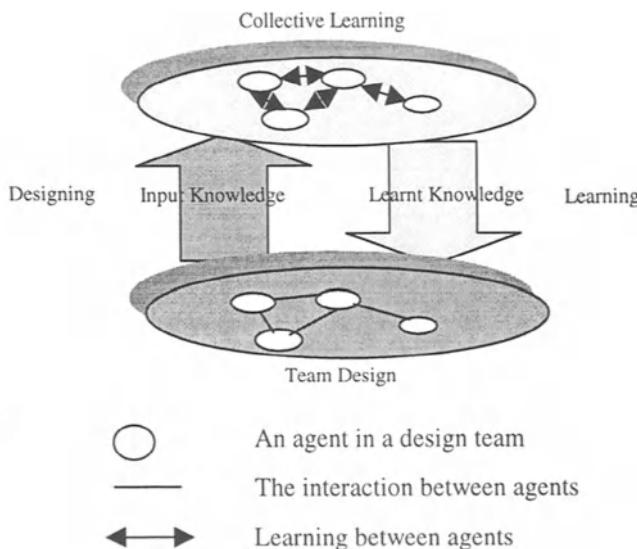
Collective learning in design is considered in this work in the context of collaborative design. Collective learning is not a new concept. In his work in organisational study, Neergaard stated collective learning is such kind of learning, where a group of individuals has learnt collectively through their interaction (Neergaard 1994). In this paper collective learning is described as agents in a team learn collectively through their interactions. Agents in the team can learn from any of others inside or outside the team, see Figure 1.

Design and learning are two activities linked together. Duffy and Duffy (Duffy and Duffy 1996) discussed design and learning loop. Collective learning can be considered in the same manner as having team design and

collective learning loop. In a team-working environment, design agents work together for a common goal, they learn from each other in the design process and what is learnt is applied to the current or future design practice, which is assumed that there is a team design and collective learning loop, Figure 2.



*Figure 1. Learning from one another*



*Figure 2. A loop of team design and collective learning*

Based on the model of learning in design (Sim 2000), the assumption made for collective learning in design includes: There may have input knowledge and output knowledge for team design activities and collective learning activities. There may have knowledge transformers that link input

knowledge and output knowledge, which may imply an epidemic link between team design and collective learning. Collective learning may occur before, during or after the team design activity representing provisional, in-situ, and retrospective triggers respectively, which indicates temporal link between team design and collective learning. There may have goals for team design activities and collective learning activities, and these two types of goal may interact with each other.

It is also postulated that there may have different modes of collective learning, depending on where the input knowledge comes from. For example, one agent may learn from another agent directly through acquiring the knowledge, or may derive new knowledge from input knowledge of several agents. Or new knowledge may be produced through deriving from the agent's own knowledge and other agents' knowledge.

## 5. Protocol Analysis Approach

Protocol analysis is used to evaluate the hypothesis of collective learning in design made in section 4. The assumption made in the protocol analysis is that “the verbalisable cognitions can be described as states that correspond to the information that is in the focus of attention”, and that “the information vocalized is the verbal encoding of the information in short-term memory” (Ericsson and Simon 1984). In this paper, a meeting of a design team that were tasked with designing a fluid delivery system for a 3D printer, is recorded using video camera. There are five team members from the 5<sup>th</sup> year, Masters of Engineering, of a product design course. Each member has at least four years team working experience. The team members are identified as Gi, Pa, Ma, Da, and Je in the protocol. The whole design process lasted one hour and thirteen minutes.

The process of the protocol analysis includes data segmentation, coding of data, analysis of data, and interpretation of data (Gero and Mc Neill 1998). In the following sections, the segmentation and coding scheme, the method to segment the protocol into team design activities, and results of the protocol analysis are presented.

### 5.1 SEGMENTATION AND CODING SCHEME

The segmentation of the team working activities was based upon the types of design activities in the formalism of generic design activities in (Sim 2000), Table 1. It is assumed that team design activities can be categorised into such types of design activities as indicated in Table 1 and there may have more than one agents participated in the design activity, which is different from individual design activities.

TABLE 1. Types of design activities (Sim 2000)

| Classification               | Activities and transcript codes                                                                                                                                                            |
|------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Design definition activities | Abstracting (Abs), Association (Ass), Decomposition (Dec), Generating (Gen), Detailing (Det), Defining (Def), Standardising (St), Structuring/Integrating (Str/Int), Synthesis (Syn),      |
| Design evaluation activities | Analysis (Anl), Modelling (Mod), Evaluation (Ev), Decision making (Dm), Testing /Experimenting (Tes/Exp)                                                                                   |
| Design management Activities | Constraining (Con), Information gathering (Inf), Identifying (Id), Exploring (Exp), Searching (Sch), Resolving (Res), Selecting (Sel), Prioritising (Pr), Planning (Pl), Scheduling (Schd) |

Based on the formalism of generic design activities in (Sim 2000), the elements for the team design activities includes:

- Team design goal,  $D_g(T)$ . It is assumed that the design goal of the team is the common goal for the members.
- Input knowledge for team design activities,  $I_k(T)$ .
- Output knowledge from the team design activities,  $O_k(T)$ .

Modes of collective learning ( $CL_m$ ) were analysed. Based upon the assumption made in section 4, following elements of collective learning are included in the protocol analysis:

- Input knowledge for collective learning,  $I_k(\text{agent}_i)$ .
- Output knowledge through collective learning,  $O_k(\text{agent}_i)$ .
- Knowledge transformers for collective learning,  $K_t(\text{agent}_i)$ .
- Learning triggers,  $T_r(\text{agent}_i)$ .
- Knowledge change,  $\Delta K(\text{agent}_i)$ .
- Collective learning goal,  $L_g(\text{agent}_i)$ .

The protocol is encoded using the elements of team design activity and collective learning activity, and used to evaluate the hypothesis of collective learning.

## 5.2 SEGMENTATION INTO TEAM DESIGN ACTIVITIES

Key words, phrases and the nature of the design activities are used as the criteria to segment the team design activities. The protocol was segmented into 71 parts. Not all the design activities formalised in Sim's model are identified in the protocol, Table 2.

TABLE 2. The design activities identified in protocol of team working (✓)

| Types of Design Activities   | The design activities identified in the protocol                                                                                                                 |
|------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Design definition activities | Abstracting, Association (✓), Decomposition, Generating, Detailing (✓), Defining, Standardising, Structuring/Integrating, Synthesis                              |
| Design evaluation activities | Analysis (✓), Modelling, Evaluation (✓), Decision making (✓), Testing /Experimenting                                                                             |
| Design management Activities | Constraining (✓), Information gathering (✓), Identifying (✓), Exploring (✓), Searching (✓), Resolving, Selecting (✓), Prioritising, Planning (✓), Scheduling (✓) |

## 6. Elements of Collective Learning in Design

### 6.1 DESIGN ACTIVITIES

#### 6.1.1 Design goals

Team design goals in many design activities seem to be verbalised at the beginning of the design activity. The keywords to identify the team design goals are like “Gi: We need to design a lid...”, “Gi: We need to design something to seal the tube...”, “Ma: I still think we need something to stop the flow....”, etc. Team design goals can be inferred from these keywords. The goals within these design activities are “design a lid”, “design something to seal the tube”, and “design something to stop the flow”. Team design goals can also be inferred from the questions verbalised by the participants. For example, the questions are like “Gi: Where are we going to get a printer from...?”, “Pa: So what are they expecting us to do then? ...”, “Gi: Is there any need for a tank? ...”, “Gi: Right, what about the mounting of the tank then? Do you think it should be on the gantry or moving with it?”. The team design goals within those team design activities are “get a printer”, “clarify their design tasks”, “decide the location to mount the tank”. The team design goals can also be inferred from the other elements of team design activities, like input knowledge or output knowledge. For example in Table 3, Gi provides input knowledge, the tank should be sealed along the edges and the silicon gel can be used, Pa provides the input knowledge, a small gasket may be used, Da provides the input knowledge, the tank should be sealed across the three compartments, and Ma provide the input knowledge, if the tank is cut flat the silicon stuff can be used. From those input knowledge, it can be inferred that design goal of this team design activity is to decide the location and the materials to seal the tank.

TABLE 3. An example of team design goal inferred from other elements of team design activities

|                                                                                                                                                                                  |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ...Gi: I think it should be <b>sealed along those edges so that the whole thing is sealed.</b>                                                                                   |
| Pa: What about a small gasket?                                                                                                                                                   |
| Gi: What do you mean?                                                                                                                                                            |
| <b>Pa: Just a small gasket that seal around, but I don't know how you'd get that done and how you'd seal it.</b>                                                                 |
| <b>Gi: Even just use silicon gel round the top.</b>                                                                                                                              |
| Da: But because it's 3 compartments...Easier if it's just a single compartment. But because it's the <b>three you'd have to have rigid edges round here and a seal across...</b> |
| Pa: Seal across here and here... you could possibly have it coming in ...                                                                                                        |
| Gi: Could you not have the silicon binder stuff just going along all these surfaces and just stick the top on?                                                                   |
| <b>Ma: If you cut it well enough you should get it pretty close.</b>                                                                                                             |
| Ma: As long as it's flat.                                                                                                                                                        |

#### 6.1.2 Input knowledge

Input knowledge often follows the verbalisation of the team design goal. Different from one agent's design activity, in team working there may be more than one agent who provides input knowledge for the design activity. Like the design activity in Table 3, four agents, Gi, Pa, Da, and Ma, provide input knowledge. There are only a few examples that only one agent inputs knowledge. For example, only Pa provides input knowledge in the team design activity described in Table 4. Pa explains its design idea to the rest of the team members. The protocol indicated that Gi learns that idea from Pa, and it is possible that the rest of the team members may learn that knowledge through Pa's sharing of his design idea.

#### 6.1.3 Output knowledge

Output knowledge can be inferred or derived from the input knowledge and the goal of the team design activities. For example, in the design activity in Table 3, the output knowledge is derived from Gi, Pa, Da, and Ma's input knowledge, through sharing, and the output knowledge, the edges of the tank should be sealed and across the 3 compartments, and the silicon stuff can be used if the edge is cut flat, is derived. In some examples, the output knowledge of the team can be derived from only one agent's input knowledge. In the team design activity in Table 4, only one agent, Pa, input knowledge and through sharing the team learns his design idea and the output knowledge in that design activity is the understanding of Pa's idea of mounting the tank.

TABLE 4. Example of only one agent providing input knowledge for a team design activity

...Pa: (drawing) Set it on to two clips and just press it in. There you go there's the design.  
 Gi: (pointing) What's that?  
 Pa: That's a hole.  
 Gi: (Laughs)  
 Pa: That's the hole for the tube so just tuck it in - snap fit, just push out.  
 Gi: Yes, I get that bit.

## 6.2 COLLECTIVE LEARNING

Collective learning can be identified through the keywords or phrases or sentences. For example, the key phrase in “Gi: We have to use cartridge because we are using the cradle. Because *Ge* (an agent outside the team) *said so....*” indicates that Gi learns the team’s design requirement from Ge. There are other examples like “Gi: ... Can we not have one of those remote control syringe things that *Ry* (an agent outside the team) *was talking about?....*”, where Gi learns about the remote control from Ry. In another example, the format of question is used, like “...Pa: Where do we put a grub screw at the side to keep it tight? Da: With a plastic sleeve around it, it would not have too much definition...”. In this example, it can be inferred that Pa can learn from Da how the screw can be kept tight. There are some other examples in the protocol where collective learning occurs through asking questions, such as “Gi: Is there anyway that we can incorporate the valve that operate and that is also run by the processor in the computer? So that when that is reaching out it allows the flow to reach in. Is there any other way to do it when it is being controlled? Ma: Just put a valve on it or something.”, “...Gi: What do you mean? Pa: Just a small gasket that seals around, ...”, “...Ma: What you mean have just a thing up here (pointing to gadget) or what? Da: No, just design a system like ...”. In some examples, the keywords like “yeah”, “I think so”, “oh, yes”, “Yes, I get that bit”, “Eh, that’s a good point”, “he (*Ge*) told me”, “*Yes, that idea*”, “I know”, “Oh right I didn’t know you could get that as one thing”, indicates that one agent learns from other agents. Collective learning can also be inferred through the communication between the members. For example, “... Ma: (drawing) That’s the gantry there. You can have it sitting up there. Gi: No, no, no but remember you’ve got a Z direction. That whole bit there (pointing) is moved back ...” from which it can be inferred that Ma can learn from his own knowledge and being reminded by Gi that his design of the location of the

gantry is infeasible due to his forgetting to take consideration of the Z direction.

### *6.2.1 Input knowledge*

In the protocol analysis, the input knowledge for collective learning can be verbalised after the keywords or phrases of design goal, or after the questions. For example, in Table 13, the input knowledge from Da and Ma is followed by the key phrases “we’re talk about size here” and key words “volume” “stability” which implies that the design goal of the volume and the stability of the tank. In another example in Table 11 the input knowledge from Ma and Gi is provided after Pa’s question “what are they expecting us to do then”.

### *6.2.2 Output knowledge*

In collective learning, output knowledge of one agent can be the input knowledge from another agent, or can be acquired/derived from the input knowledge from itself and other agents, or from other agents’ input knowledge without its own input knowledge. For example, in Table 10, the output knowledge of Ma is acquired from Gi. The knowledge is only transferred from one agent to another agent. In Table 12, Gi’s output knowledge, that her option to design the top of the cartridge is not feasible, can be derived from her own knowledge and Da’s input knowledge. In Table 11, the output knowledge of Pa, that their design still depends on Gerry’s programming, can be derived from Gi and Ma’s input knowledge. In these two examples, new knowledge is created through deriving the input knowledge from itself and other agents, or just from other agents.

### *6.2.3 Knowledge transformers*

Knowledge transformers serve as a tool to change the input knowledge into output knowledge. It is difficult to identify the knowledge transformers in collective learning which link input and output knowledge from different agents. Through the protocol analysis three types of knowledge transformers are identified: Acquisition/Sharing, Derivation (Reformulation), and Requesting/Replying. When an agent acquires new knowledge from one or more of other agents, the knowledge transformer is Acquisition/Sharing, and when new knowledge is derived from other agents or from both the agent’s own and other agents’ knowledge, the knowledge transformer is derivation. Beside these two knowledge transformers, another knowledge transformer, Requesting/Replying, is used where the agent request knowledge from one or more other agents, in the form of question, and the knowledge is acquired through other agents’ replies. Requesting/ Replying can be understood as a

special knowledge transformer of acquisition/sharing, or derivation (reformulation), depending on whether new knowledge is derived from other agents. One example of acquisition acting as knowledge is like the design activities in Table 5, where Ma acquired the knowledge from Gi directly. In the design activity in Table 11, derivation is identified as knowledge transformer, where Pa's knowledge that making a model still depends on Gerry's work is derived from the input knowledge of both Gi and Ma. The design activity in Table 6 can be used to illustrate requesting/replying acting as knowledge transformer in the collective learning activity, where Ma learns from Da that the cartridge and tank can be replaced at once, and what the size of the tank could be in consideration of the useful life of the printer head through his explanation of his design idea.

TABLE 5. An example of acquisition acting as a knowledge transformer

|                                                                                                                                                                                                                                                                                                                                                                                   |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p>...Ma: That's the thing we could design almost anything but it's not going to get made downstairs because they can't make much out of plastic, especially that size.<br/>         Gi: It's pretty minimal - it's like cut a bit of plastic, drill three holes in it and put some silicon gel on the top.<br/>         Ma: Oh yes, I know I mean anything like complicated.</p> |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

#### *6.2.4 Learning triggers*

Collective learning triggers can be divided into two subcategories: temporal triggers that reflect the links between team design and collective learning in the aspect of time, and rational triggers that are the different reasons that trigger collective learning.

Temporal triggers are identified, which is the same result as Sim's analysis based upon a single agent: Provisional triggers, In-situ triggers and Retrospective triggers. In some team design activities, one agent can learn a piece of knowledge from others before their team design and share that knowledge with the team members.

TABLE 6. An example of Requesting/Replying acting as a knowledge transformer in collective learning

|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p>...Ma: What do you mean have just a thing up here (pointing to gadget) or what?<br/>         Da: No, just design a system like what we're talking about just now but instead of one that would take the tank off and having air get into the system replacing the cartridge at certain stages, replacing the whole lot now at once. If you say the print head's got a useful life of 40 builds or whatever; if it only uses a wee tiny bit each build, if we design a system that's gonna use that much.</p> |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

For example, in the design activity in Table 7, Gi learns from Gerry (an agent outside the team) the piece of knowledge that they have to use a cartridge because they are using the cradle before the design, and shares this knowledge with other team members during, which is a kind of provisional collective learning. Collective learning can also occur in a retrospective way. In a design activity, an agent can retrospectively recall the knowledge that is learnt from other agents and use that knowledge to the current design situation. For example, in the protocol in Table 13, Ma retrospectively learnt the knowledge that a detachable tank can be used in the cartridge from Pa and apply the learned knowledge to the current design. There are also examples of in-situ collective learning, like the collective learning activities in Table 10, Table 11, Table 12, and Table 5.

TABLE 7. Provisional collective learning

|                                                                                                                                                             |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Gi: Right, we have to use a cartridge because we are using the cradle. <b>Because Gerry said so.</b><br>So we have to design a little bit to go on the top. |
| Da: <b>Yeah.</b> It is not just the tube into the ink recess or whatever it is called.                                                                      |
| Ma: <b>Yeah</b>                                                                                                                                             |
| Da: What about pull straight into the ...                                                                                                                   |
| Gi: Straight into the hole at the bottom of the ...                                                                                                         |
| Da: Feeder.                                                                                                                                                 |

There are different reasons that trigger collective learning, which is named as *Rational Triggers*. Through protocol analysis, five types of rational triggers are identified: one agent or other agents' new or different design idea(s), the failure of the design, the success of the design, the disagreement of other agents' design, and the request of knowledge. One agent can learn, triggered by other agent's new or different knowledge. For example in Table 7, Da learns from Gi triggered by her idea that they have to design "a little bit of the top" of the cartridge. The failure of the design or the potential problem in the design can also trigger the agent to learn. For example, in the design activity in Table 8, the failure of the design that members of other teams may not know the tank may be put in the top of the cartridge, triggers Gi to learn from Pa that they may clip something on the side. The success in one agent's design idea can also trigger another agent or other agents to learn. For example, in Table 13, the design idea from Pa triggers Ma to learn. Conflict may occur in a team-working environment (Klein and Lu 1989; Klein 1991). When disagreement occurs among the agents, it may trigger collective learning. For example, in Table 9, Gi has the idea that they don't have to seal the tank but with the disagreement of Da and Pa and their explanation of the results without sealing the tank Gi's

idea has been changed and Gi can learn a new piece of knowledge that the tank has to be sealed. Collective learning can also be triggered through the request of knowledge in one agent. One agent may request knowledge from other agents, or want other agents to confirm or disconfirm a design idea, or want other design agents to explain their ideas, etc. In the form of asking questions, the agent can learn from other agents triggered by their reply. The collective learning activities in Table 10 and Table 13 are triggered by requests for knowledge.

TABLE 8. The failure of the design triggered collective learning

|                                                                                                                                                                                                                                                                                         |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Gi: <b>But they might design it not knowing - that's the problem as well - not knowing that they're putting a big tank at the top.</b> All that we need to worry about is they're mounting something on the gantry arm and we were possibly going to mount something on the gantry arm. |
| Pa: (drawing) <b>Something, even that and just clip something onto the side like that and I mean that's the dimensions of it so that's just about all you would need for one big build, for maybe for two or...</b>                                                                     |

TABLE 9. The disagreement of other agents trigger one agent's collective learning

|                                                                                                                                                                                                                                                                                   |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Gi: <b>That doesn't have to be sealed (pointing to rawing)</b> because as long as that water level's there you're not gonna get any water in it. They're not gonna get any air in it rather. As long as it's down to like there, know what I mean, as long as that bit's covered. |
| Da: <b>Even there, if it's running through a sponge then you're not gonna get any air through it anyway.</b>                                                                                                                                                                      |
| Pa: <b>Don't want it contaminated with dust as well,</b> you know you want to keep it quite ...                                                                                                                                                                                   |
| Gi: <b>Yes, it would definitely have to be a sealed...</b>                                                                                                                                                                                                                        |

### 6.2.5 Learning goals

It is difficult to identify collective learning goals, as the individuals did not verbalise such words as “the goal I learn from him/her is ...”, etc. Similarly with the identification of the team design goal, the goal of collective learning could be identified through keywords or phrases or key sentences, and it could also be inferred from the team design goals, input knowledge and output knowledge of the team design activities. For example, in the team design activity in Table 9 the collective learning goal of Gi, that whether the tank needed to be sealed or not, is inferred from the keywords “Gi: That doesn't have to be sealed...”, and the input knowledge of both Da and Pa. In another example in Table 6 Ma's learning goal to understand Da's design idea is identified through Ma's question “What do you mean have just a thing up here (pointing to gadget) or what?” and Da's answer to his question.

## 7. Four Modes of Collective Learning

Through protocol analysis, four modes of collective learning have been classified by modes refer to the relationships between the agents' interactions: In the first mode, the input knowledge for collective learning comes from one of the other agents. For example, in Table 10, Ma learns from Gi that two valves can be used as filtering system. The second mode is that the agent learns through input knowledge from more than one of the other agents. In this mode, there is more than one of other agents providing input knowledge for the agent's learning activity. In Table 11, Pa learns from both Ma and Gi in that they may make a prototype model for their design. The third mode is that an agent learns through both input knowledge from itself and others. For example, in Table 12, Gi learns that her own design idea of the top of the cartridge is infeasible, derived from her own knowledge and being reminded by Da. In the final mode, one agent learns the knowledge from another agent and then shares that knowledge with other agents. For example, in Table 13, Ma learns the knowledge from Pa that the tank can be designed to be detachable and then share this knowledge with Da. In this learning, it can be understood that Da can learn from Ma directly and can learn from Pa indirectly through Ma.

TABLE 10. First mode of collective learning: an agent learns through input knowledge from another agent

|                                                                                                                                                                        |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Ma: I think we should just keep it pretty simple because the tank that's here at the moment is just ... (Gillian smiles) and it's got no filters or pumps or anything. |
| Gi: No, well it's got two one-way valves in the system to prevent stuff getting passed through.                                                                        |
| Ma: Yes, we can put two valves on this.                                                                                                                                |

TABLE 11. Second mode of collective learning: the agent learns through input knowledge from more than one of the other agents

|                                                                                                                                                                              |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Pa: So what are they expecting us to do then?                                                                                                                                |
| Ma: I don't know.                                                                                                                                                            |
| Gi: Make a model as best we can. It's only gonna be like a kind of prototype. It's not like to be sold.                                                                      |
| Ma: Or we can just design anything then make a model out of whatever, wood or something.                                                                                     |
| Gi: We just have to make a representation.                                                                                                                                   |
| Pa: It would be good if we had a prototype. Again we are dependent on Gerry coming up with the goods.                                                                        |
| Ma: I think we should just work out a way of doing it without Gerry because to wait on him when he hasn't done the programme in like three weeks, then we're in big trouble. |
| Pa: I think we need to go down and speak to him today again.                                                                                                                 |

TABLE 12. Third mode of collective learning: the agent learns through input knowledge of itself and other agents

|                                                                                                                                                                                                                                                                                                                       |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p>Da: I think we're talking about size here, we're all talking about ... what about (drawing and describing) still have the same volume. What's gonna give you more stability?</p> <p>Ma: Why don't we just make something that you can stick anywhere and have it, like, detachable, like Paul (Pa) was saying?</p> |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

The four modes of collective learning are shown in Figure 3. It is noted that Mode 4 can be a repeated process of mode 1 or other modes or the combination process of the three other modes. That is, the agent can learn a piece of knowledge from one or more than one agent and shares that knowledge with another or more than one other agents, the agent's own knowledge can also be involved in the collective learning process.

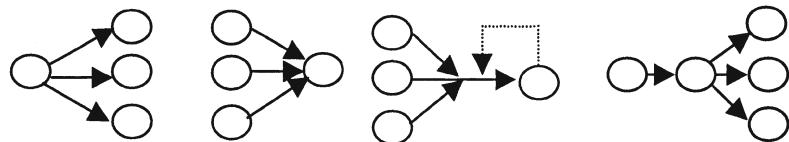
TABLE 13. Fourth mode of collective learning in design: one agent learns the knowledge from another agent and then shares that knowledge with other agents

|                                                                                                                                                                                                                                                                                                                                                |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p>Gi: (pointing) That is one option (Drawing) or having three compartments and a lid with three tubes in the top of there where each tube goes in.</p> <p>Da: We have the same problem as stuff coming out the side.</p> <p>Gi: The sponge should soak it up. This is going back to the same ... what are we designing here? Is it a lid?</p> |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

## 8. Forms of Collective Learning

Within each mode there can be different forms of collective learning. To date, three forms of collective learning have been identified:

- Mutual learning,
- Common learning, and
- Combining knowledge.



Mode 1: one to many    Mode 2: many to one    Mode 3: many to one plus itself    Mode 4: combination of different modes

*Figure 3. Four modes of collective learning in design*

### 8.1 MUTUAL LEARNING

Through collective learning the knowledge states of agents can evolve mutually. The mutual learning process can include four phases (see Figure 4): the agent<sub>1</sub> learns a design idea from agent<sub>2</sub>, and then produces a new

design idea. Agent<sub>2</sub> learns that idea and produces another new idea. Agent<sub>1</sub> and agent<sub>2</sub> can interact with other agents and mutually evolve their knowledge.

In the protocol, mutual evolution has been observed. For example, in the protocol in Table 14, Da and Gi evolved their idea mutually. Da had a vague idea to design a cap fitted into the top of the cartridge, and that idea was clarified reminded by Gi's idea, and in turn Gi learned that design idea.

## 8.2 COMMON LEARNING

All the agents in the team may learn the same piece of knowledge at the same time or at different times, namely common learning. In the protocol, the common learning activity can be identified using key words like "yeah", and the context of the communication. For example, in Table 15, Gi learned from Gerry that they have to use cartridge because they are using the cradle, and she shared this piece of knowledge to the rest of the team members. It can be inferred that Da and Ma learned this piece of knowledge through their verbalisation of the key words "yeah". It can also be inferred that the rest of the members learned this knowledge from the context of the protocol, for in the following communication, they started to discuss how to design the cartridge. Common learning can also be identified by the verbalisation. For example, the verbalisation, "Gi: Gerry told us what the maximum volume was for that kind of build...", indicates that all the team members can learn the same piece of knowledge, the maximum volume of the flow, from Gerry.

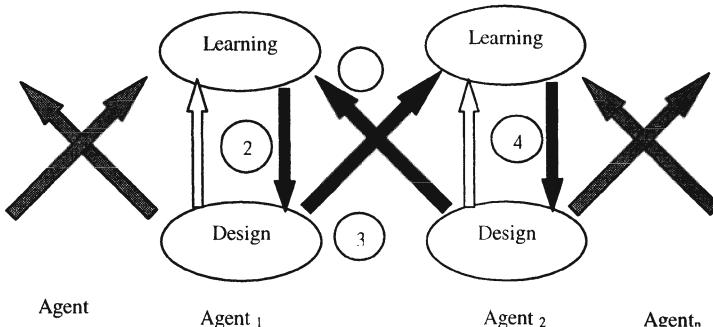


Figure 4. Mutual knowledge evolutions

## 8.3 COMBINING KNOWLEDGE

When different knowledge sources are put together, holistic knowledge can be produced. For example, in Table 16, the combined knowledge that *the*

*connector made by other groups cannot be long* is produced when putting the team's design and another team's together.

TABLE 14. An example of mutual learning

|                                                                                                                  |
|------------------------------------------------------------------------------------------------------------------|
| Da: But I reckon if you, instead of using these caps, if you just have (drawing)...                              |
| Gi: Yeah                                                                                                         |
| Da: That just fits in the top - I'm trying to think of an example. You know the kind of thing I'm talking about. |
| Gi: Yeah, like the steradent tops?                                                                               |
| Da: Yes, is that the stuff they use for dentures?                                                                |
| Gi: Yeah.                                                                                                        |
| Da: Yes, that idea.                                                                                              |

TABLE 15. An example of common learning

|                                                                                                                                                                          |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Gi: Right, we have to use cartridge because we are using the cradle, because Gerry said so. So we have to design a little bit to go on the top of this about this topic. |
| Da: Yeah. It is not just the tube into the ink recess or whatever it is called.                                                                                          |
| Ma: Yeah                                                                                                                                                                 |

Through collective learning the knowledge states of agents will change, and they may learn knowledge that cannot be learned in individual learning. For example, in the team meeting, Gi learned the knowledge of the *design of caps for the cartridge* from Da, the knowledge of *elaborative design of the cartridge* from Pa, and the *knowledge of design bards for the tube* from Ma.

TABLE 16. An example of combining knowledge

|                                                                                             |
|---------------------------------------------------------------------------------------------|
| Gi: So we'd need to do something with the fluid connector bit so that it doesn't tangle up. |
| Pa: Clarity                                                                                 |
| Ma: And we'll just have to hope that the other groups haven't made it that long.            |
| Gi: We know what that depth is, Gerry told me what that depth is.                           |

There are different types of knowledge that can be only learnt through collective learning: meta-knowledge (i.e. the knowledge of the knowledge states of other agents), common knowledge, and holistic knowledge. For example, in Table 17, other team member learned the knowledge that Gi had the knowledge of the volume of cartridge. Common knowledge and holistic knowledge can be acquired through common learning and combining knowledge, such as examples in Table 15 and Table 16.

TABLE 17. An example of learning meta-knowledge

|                                                 |
|-------------------------------------------------|
| Gi: We need to do it as an experiment.          |
| Ma and Pa: I thought he done that.              |
| Gi: All he did is gives us the volume.          |
| Ma: What was the volume then?                   |
| Gi: I've probably got it on an email somewhere. |
| Ma: because we can work it out from that.       |

## 9. Team Design and Collective Learning Interactions

### 9.1 THE EPISTEMIC LINK

Three types of knowledge transformers have been identified in the protocol, Acquisition/Sharing, Derivations (Reformulation), and Requesting/Replying that link the input knowledge and output knowledge in team design activity and collective learning activity. In Sim's model of learning in design (Sim 2000), the mapping relationship between the type of the knowledge transformers and the type of design activities was observed. Such a mapping relationship was also postulated to exist in collective learning. The mapping result is shown in Table 18. It is found that the transformer acquisition/sharing is more likely used in the team design activities like detailing, decision-making, analysis, exploring, identifying, information gathering, and planning. The transformer Derivation (Reformulation) was more likely used in the design activities like decision making, analysis, exploring, identifying, and selecting, and the transformer Requesting/Replying was more like used in the design activity like exploring.

TABLE 18. Mapping between knowledge transformers of collective learning and categories of team design activities inferred in the protocol analysis

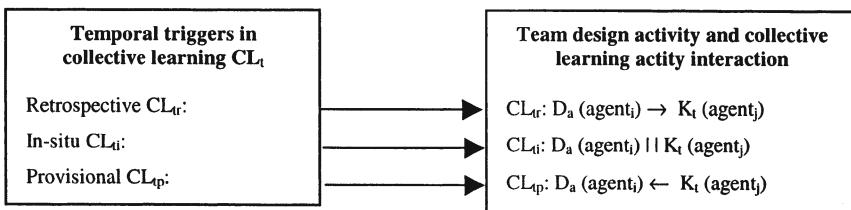
| Knowledge transformer                     | Design definition activities | Design evaluation activities | Design management activities                            |
|-------------------------------------------|------------------------------|------------------------------|---------------------------------------------------------|
| Acquisition/Sharing                       | Detailing                    | Decision making, analysis    | Exploring, identifying, information gathering, planning |
| Derivations (Reformulation)/Randomisation |                              | Decision making, Analysis    | Exploring, Identifying, Selecting                       |
| Requesting/Replying                       |                              |                              | Exploring                                               |

### 9.2 THE TEMPORAL LINK

In Section 6.2.4 three types of temporal triggers in collective learning are identified: retrospective, in-situ, and provisional trigger. Collective learning can occur before, during, or after the team design activities. Similar to individual learning, there is temporal link between team design activities and collective learning, Figure 5.

### 9.3 THE TELEOLOGICAL LINK

In Sections 6.2.5 and 6.1.1, the goal of collective learning and the goal of the team working were identified. In this section, the relationship between them is investigated.



*Figure 5.* The temporal link between team design activity and collective learning activities

In individual learning, two types of relationship between learning and design were identified and substantiated: the learning goal precedes the design goal and the design goal precedes the learning goal (Sim, 2000). It was assumed that these types of interaction between collective learning goal and team design goal exist. Through the protocol analysis these two types were identified. For example, in Table 12, Pa's learning goal, to clarify the design task in their team, is inferred at the start of the team design activity that preceded the team design goal, to decide their design tasks. In another example in Table 19 the team design goal, designing something to seal the tube, preceded Pa's learning goal, acquiring the place to put a grub screw at the side to keep it tight. In section 6.2.4, different types of rational triggers for collective learning are identified that link with collective learning and thus with collective learning goal.

TABLE 19. Design goal precedes learning goals in collective learning

|                                                                                                                                                                          |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Gi: We need to design something to seal the tube so it doesn't leak, whether it is choosing a component like a rubber grommets, because we cannot just ram the tubes in. |
| Da: Grub screws?                                                                                                                                                         |
| Pa: Where do we put a grub screw at the side to keep it tight?                                                                                                           |
| Da: With a plastic sleeve around it, it would not have too much definition. (Drawing a design option)                                                                    |

## 10. A Model of Collective Learning in Design

Through the protocol analysis, a model of collective learning in design is developed, Figure 6, which extend the model of learning in design (Sim 2000). The model to some degree reflects the nature of collective learning in design.

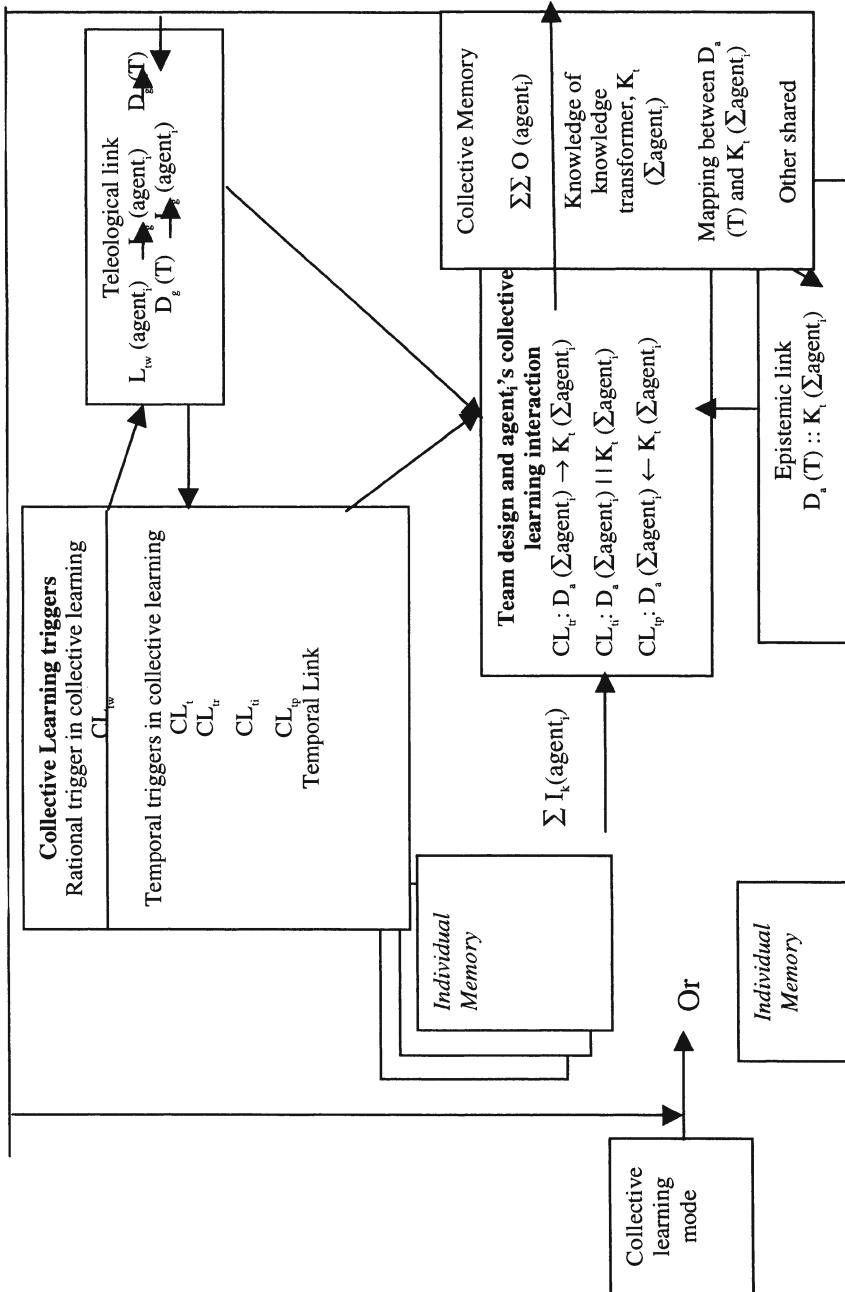


Figure 6. A model of collective learning in design

Learning and memory are interconnected – what we already have in our memory affects what we learn and what we learn affects our memory (Kim, 1993). To correspond to *collective learning*, the term of *collective memory* is used, which is defined as *the sum of the knowledge in agents in a design team and the knowledge resources that shared by all the agents in the team*. Collective memory plays the role of storage of knowledge in collaborative design

The modes of collective learning decide the mode of input knowledge: whether only one or more than one agent provides input knowledge for another agent's learning activity, or whether the agent's own knowledge, together with other agents' input knowledge, are provided in its learning activity, or the output knowledge of one agent's collective learning activity serves as input knowledge for another agent's design activity through sharing. The input knowledge is stored in individual memories. Agents learn from one another through interaction. There are inextricable links between team working and collective learning. Three types of links between team design activities and collective learning activities are identified: the epistemic link, teleological link and temporal link. There are reasons for one agent learns from other agents, called rational triggers, which trigger collective learning. What is learnt collectively may be shared with other agents in the actual or future design activities, such as the design Table 13 where the knowledge learned from Pa is used in current design and it is possible that what is learnt can be used in the future design, which provides evidence for a team design and collective learning loop.

The model is developed based upon the protocol analysis of team design and an existing model of learning in design (Sim 2000). Such a model can to some degree describe the phenomena of collective learning in design and in the future work the model will be further formalized and evaluated.

## 11. Conclusion

To be able to use computers to support design, the nature of design should be understood before hand. Learning in design has been investigated since the 1980's and models and theories have been developed. While most of the research is focused on individual learning in design, no comprehensive model of collective learning in design has been found prior to this paper. In current literature, there is some work related to collective learning in organisational theory and computer science, and some work in design that reflects some aspects of collective learning in design. However, a comprehensive model of collective learning is not presented in their work.

In this paper, the phenomena of collective learning in design have been investigated using protocol analysis. The elements of team design and collective learning, the links between team design and collective learning, modes of collective learning, and different forms of collective learning have been analyzed, and a model of collective learning in design has been proposed. There are different forms of collective learning, namely mutual learning, common learning and combining knowledge, which make it unique from individual learning.

In future work, the proposed model of collective learning in design will be formalized and evaluated, and the investigation of collective learning will focus on how a design team as a whole learns and how collective learning and individual learning relate to each other. Computational supported means for collective learning will also be investigated.

## Acknowledgements

We are indebted to Ms Christin Rankin who spent many hours to transcribe the tape for us and to the fifth-year students in DMEM, University of Strathclyde who were co-operative in the recording. Special thanks also go to the anonymous reviewers of this paper who had insightful comment on it.

## References

- Argyris, C and Schön, D: 1977, *Organizational Learning: A Theory of Action Perspective*, Vol. 9: Addison-Wesley, Sydney.
- Cross, R and Israelit, S (eds): 2000, Introduction, *Strategic Learning In A Knowledge Economy, Strategic Learning in a Knowledge Economy: Individual, Collective and Organizational Learning Process*, Butterworth Heinemann, Oxford.
- Duffy, S and Duffy, A: 1996 Sharing the learning activity using intelligent CAD, *AI EDAM* **10**: 83-100.
- Dunskus, B, et al: 1995, Using single function agents to investigate conflict, *AI EDAM* **9**: 299-312.
- Eriesson, K and Simon, H: 1984, *Protocol Analysis: Verbal Reports as Data*, MIT Press, Cambridge, MA.
- Gero, JS: 1990, Design prototypes: a knowledge representation schema for design, *AI Magazine* **11**: 26-36.
- Gero, JS and Mc Neill, T: 1998, An approach to the analysis of design protocols, *Design Studies* **19**: 21-61.
- Grecu, D and Brown, D: 1996a, Learning by single function agents during spring design, in JS Gero and F Sudweeks (eds), *Artificial Intelligence in Design '96*, Kluwer, Dordrecht, pp. 409-428.
- Grecu, D and Brown, D: 1996b, Dimensions of learning in agent-based design, *AID'96 ML in Design Workshop*.
- Grecu, D and Brown, D: 2000, Expectation formation in multi-agent design systems, in JS Gero (ed.), *Artificial Intelligence in Design '00*, Kluwer, Dordrecht, pp. 651-652

- Haynes, T and Sen, S: 1996, Co-adaptation in a team, *International Journal of Computational Intelligence and Organizations* 1(4): 240-268.
- HO, F and Kamel, M: 1998, Learning coordination strategies for cooperative multiagent systems, *Machine Learning* 33: 155-177.
- Karni, R, et al: 1992, Towards learning in concurrent engineering design, *A workshop of AID'92 Machine Learning in Design*.
- Kim, D: 1993, The link between individual and organizational learning, *Sloan Management Review* Fall: 37-50.
- Klein, M, and Lu, S: 1989, Conflict resolution in cooperative design, *Artificial Intelligence in Engineering* 4(4): 168-180.
- Klein, M: 1991, Supporting conflict resolution in cooperative design systems, *IEEE Systems, Man and Cybernetics* 21(6): 1379-1390.
- Marton, F and Booth, S: 1997, *Learning and Awareness*, Lawrence Erlbaum, Mahwah, NJ.
- Neergaard, C: 1994, *Creating a Learning Organisation: A Comprehensive Framework*, Department of Production, Aalborg University, Denmark.
- Ono, N and Fukumoto, K: 1997, A modular approach to multi-agent reinforcement learning, in G Weib (ed.), *Distributed Artificial Intelligence Meets Machine Learning In Multi-Agent Environments*, Springer, Berlin, pp. 25-39.
- Pearsall, J (ed): 1998, *The New Oxford Dictionary of English*, Oxford University Press, Oxford.
- Persidis, A and Duffy, A: 1991, Learning in engineering design, in H Yoshikawa, F Arbab and T Tomiyama (eds), *Intelligent CAD, III*, Elsevier, 251-272.
- Prasad, M, et al: 1997, Learning organizational roles in a heterogeneous multi-agent system, *International Journal of Human-Computer Studies* 48(1): 51-67.
- Prasad, M and Lessor, V: 1997, *Learning Situation-Specific Coordination in Cooperative Multi-Agent Systems*, Department of Computer Science, University of Massachusetts, Amherst, MA.
- Reich, Y and Fenves, S: 1989, The potential of machine leaning techniques for expert systems, *AI EDAM* 3(3): 175-193.
- Sen, S, et al.: 1994, Learning to coordinate without sharing information, *Proceeding National Conference on Artificial Intelligence*, Seattle, Washington, pp. 426-431.
- Shepherd, G: 1988, *Neurobiology*, Oxford University Press, Oxford.
- Sim, SK: 2000, *Modelling Learning in Design*, Ph.D. Thesis, CAD Centre. University of Strathclyde, Glasgow.
- Simon, H: 1983, Why should machines learn?, in RS Michalski, et al (eds), *Machine Learning: An Artificial Intelligence Approach*, Morgan Kaufmann , San Mateo, CA, pp. 25-37.
- Smithers, T and Troxell, W: 1990, Design is intelligent behaviour, but what's the formalism, *AI EDAM* 4: 89-98.

## 5. 8 ANALOGIES PER HOUR

*A designer's view on analogical reasoning*

PIERRE LECLERCQ  
*University of Liège*  
*Belgium*

AND

ANN HEYLIGHEN  
*KU Leuven*  
*Belgium*

**Abstract.** Several studies have pointed out that analogy is used in architectural design. By contrast, this paper reports a study that sheds more light on how this use actually takes place, i.e. at what point in the design process, for what purpose, in what range, etc. In order to study these questions, three architects were asked to design the interior of a sailing yacht while being exposed to material on small spaces, whether or not related to boats. Analysis of these design sessions unmasks analogical reasoning as a concrete, often spontaneous, and generally profitable design strategy that is used frequently, if not constantly throughout the design process.

### 1. Introduction

Throughout the design literature, analogy has been ascribed a key role in the design of architectural masterpieces (Broadbent 1973; 1980; Lawson 1994; Rowe 1982). Examples can be found in the oeuvre of no less than Frank Lloyd Wright, Rem Koolhaas and Gerrit Rietveld – to name only a few. The columns of Wright's Johnson Wax building, for instance, would have been shaped by analogy to water lilies, while Koolhaas' project for the Sea Trade Terminal in Zeebrugge suspiciously looks like a bollard to moor ships at the quay. Rietveld's Schröder House in Utrecht, for its part, strongly resembles a Mondriaan painting, both in the formal delineation of its façade and in its interior plan.

Whether based on water lilies, a bollard or a Mondriaan, the use of analogy in architecture is typically discussed because of the added value it gives to the building, in other words because of the design product. To our knowledge, however, this use has rarely been considered from the perspective of the design process. Therefore, through an experiment, this paper will try to depict the use of analogy in architecture from the point of view of an architect being involved in designing. For instance, when/at what point in the design process do architects resort to analogy? Why do they resort to it, and in what context? Do they use the analogy deliberately or spontaneously, and what (if any) are the differences between both? Under what conditions does the analogy lead to success and under what to failure? These and other questions are addressed in the experiment reported below.

After a short introduction to analogy and an overview of related research, the paper will describe the setting and procedure of the experiment, followed by the methodology and results of the analysis. It concludes with lessons learned from this experiment and topics for future research. The results of our experiment may shed a surprisingly new light on the CBR model as advocated in the early 1990s (Kolodner 1993).

## 2. Related Research

### 2.1 ANALOGY

Derived from the Greek *ana logon* (“according to a ratio”), the term analogy originally denotes a similarity in proportional relationships, e.g. between two triangles that differ in scale (Britannica.com 2001). In general, analogy can be defined as a likeness of relations, such as ‘A is related to B like C is related to D’, or in short, ‘A:B :: C:D’. Such likeness implies the existence of a higher-order abstraction that holds equally well for A:B (the *source* or *base*) and C:D (the *target*). In case of *within-domain* analogies, source and target stem from the same domain, whereas in *between-domain* analogies, source and target stem from different remote domains (Vosniadou and Ortony 1989).

Obviously architects do no hold absolute sway over analogical reasoning. According to George Polya, analogy “pervades our thinking, our everyday speech and our trivial conclusions as well as artistic ways of expressions and the highest scientific achievements” (1973). Observations of these pervading analogies have inspired several theories, which mainly differ in how they characterize the relationship between source and target. Some tend toward the ‘structural’ point of view, others towards a more ‘semantic’ one.

Dedre Gentner's Structure Mapping Theory, for instance, considers an analogy as a mapping of knowledge from one situation onto another, supported by a system of syntactic relations that is transferred from the source objects to the target objects (Gentner 1983; 1989).

By contrast, Keith Holyoak and his colleagues assert that analogical mapping is activated when trying to achieve a specific goal, and a source analog provides a more efficient solution procedure than on hand knowledge (Gick and Holyoak 1980; Holyoak 1991; Holyoak and Thagard, 1989). By consequence, mapping is directed entirely by the importance of the – largely semantic – predicates to one's current goals, whereas structural principles play second fiddle.

## 2.2 IN ARCHITECTURAL DESIGN

In architecture, as in other design domains, design problems are typically ill-defined or wicked. This very wickedness makes reasoning by analogy a potentially powerful design strategy, as it can bring forth valuable knowledge from a known situation (the source A:B) to the ill-defined design situation at hand (the target C:D). Architects may reason and learn about their design by relating it to a more familiar situation that can be viewed as structurally or semantically parallel.

Empirical research on analogy in architectural design has focused primarily on the use of visual analogies. A case in point is the work by Gabriela Goldschmidt, who has pointed out that designers may identify visual displays as candidate source analogs and establish mappings through structural or surface relations (1994; 1995). In order to investigate the impact of such displays, a series of experiments was set up in which novice and expert architects were asked to solve various design problems, with and without exposure to visual displays, and with and without explicit requirements to use analogy (Casakin 1997; Casakin and Goldschmidt, 1999). The findings of these experiments suggest that visual analogy is used and across the board improves the quality of the design result, particularly in the case of novice designers. A question that remains largely unanswered, however, is how this use takes place.

Indeed, if we are to develop efficient design tools for architects, we must know more about analogy in architectural design than that it affects the quality of the result. We need to develop a more differentiated understanding of how, when and why architects resort to analogy during design. That is exactly what the following experiment tries to do.

### 3. Protocol of the Experiment

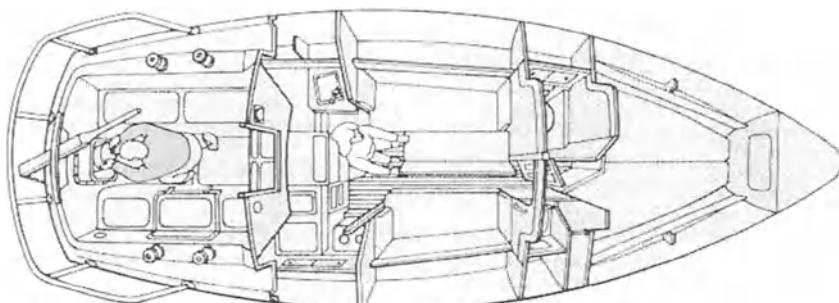
Let us first describe the specific conditions of the experiment set up to open up the discussion on the questions mentioned above.

Recall that this experiment simply aims at placing some markers in the study of analogical phenomena in architectural design. Taking into account the sample (see 3.3 below), it is out of the question to identify through statistical analysis what parameters lead to success when designers resort to references external to the design problem in order to solve it. Instead our objective consists in generating elements of qualitative replies to the questions raised.

#### 3.1 ASSIGNMENT

The assignment was purposely developed to place the subjects observed in a condition that forces them to resort to analogy during design. The duration of the experiment is limited to 3 or 4 hours; the objective is precise, limited in size and directly links up with the subjects' experience, yet it comes from an unusual domain: they are asked to design the configuration of the interior spaces of a sailing boat (yacht for 6 to 8 people, cruising on the Aegean sea).

The problems inherent to this type of design are closely related to problems in architectural design, in which the subjects are skilled. Yet, besides the exiguity of spaces, 2 constraints are completely new: the functions' overlap in 3 dimensions (see Figure 1) and, especially, the deterioration of the reference to a horizontal floor (when navigating, the sailing boat pitches in the direction of its route and lodges strongly on port or starboard), thus challenging the verticality and work in plan.



*Figure 1.3D overlap of functional spaces*

### 3.2 PROCEDURE

The experiment consists of 2 sessions, separated by 2 days.

During the 1<sup>st</sup> session, the assignment is presented to the subjects and observers (see 3.3 below). Some general and specific information is given, drawing the attention on the functional programme and the constraints specific to naval design. Subsequently the subjects have approximately one hour to study some documentation prepared in the context of this experiment, Figure 2. This information mainly consists of marine images and technical files provided by manufacturers of sailing boats, but also comprises images from other fields: sleeping-car, shuttle, Spacelab, Soyuz, catamaran, etc. It is supplemented by plans, sections and a 3D model of the yacht to be designed.

The remainder of the 1<sup>st</sup> session is devoted to the actual design work. During 2 to 3 hours, each subject works on the development of a proposal for a spatial configuration, thereby drawing from all resources s/he considers necessary – whether they come from the initial oral recommendations, the documentation provided or any knowledge derived from the subject's personal experience. Of course, subjects are invited "to think aloud" so as to allow 2 observers to capture their train of thoughts. It is important to note that subjects and observers were involved in the practice of this particular type of exercise before, thus minimizing inevitable biases of the think aloud method.



*Figure 2. Extract from the documentation provided*

The 2<sup>nd</sup> session is entirely devoted to the analysis of each subject's work. Under the direction of the 2 observers who compare their notes, subjects are invited to supplement, clarify or develop each action of their design. In particular, each episode during which an analogical transfer took place is depicted in terms of a series of criteria (origin, type, range, success, chaining, ...) that will be used later on in the overall analysis (see 4).

### 3.3 SUBJECTS AND THEIR OBSERVERS

The subjects are 3 last year students civil engineer architect of the University of Liège. As trained architects, they are no longer considered novices in designing interior spaces, nor in engineering as regards managing technical constraints. They are, however, not yet experts in their field; they do not control the market supply, or the detailed budgetary aspects of an architectural design. Yet, in the case of naval design, which interests us here, subjects find themselves, facing the new constraints, in a beginner's position. It is in this way that we push them to seek – in the documentation or in their personal experience – references to hook on to an adapted design strategy.

The 6 observers (2 per subject) are also last year students civil engineer architect. Sharing the same level of knowledge and know-how as the subjects, they are cut out to follow and re-transcribe the thoughts of their fellow students. If their role is to remain passive during the acquisition phase of the experiment, they are the principal actors in the analysis phase.

## 4. Proposal of a Diagram of Analogical Thinking

We propose to describe each analogical transfer in terms of 9 specific parameters. Figure 3a summarises them in a diagram, which also presents their relations; Figure 3b illustrates them by a concrete example. With these figures in view, let us define the different parameters.

The *target* (B) is the principal element of the analogy. It constitutes the object of attention and, in general, the problem to be solved. In our example (Figure 3b), it relates to the manual water pump in the kitchen corner. In a small sailing ship, there is no water under pressure, it should be pumped from the tank: the tap thus consists of a pumping lever.

The *trigger* (A) relates to the context of the target. It often allows the subject to identify the latter and perceive it as the next target. In our example, the photographs of the kitchen corner, extracted from the catalogue of a naval manufacturer, acted as trigger. Without having seen

these photographs, the subject would not have identified the problem of running water on board.

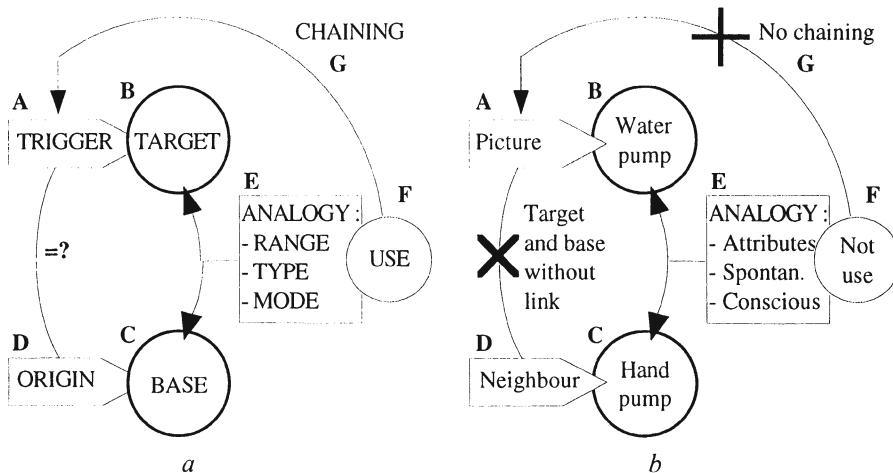


Figure 3.. Diagram and example of analogical thinking

The *base* (C) is the extra-contextual reference the target is associated with. This base constitutes the structuring element of the knowledge that will be adapted to the target. In our example, the kitchen tap-pump is associated with a lift & force pump.

The *origin* (D) is the context of the base. This context gives the base more body as a solution, by adding surrounding knowledge. In our example, the origin of the lift & force pump is situated in the subject's personal memories: his neighbour uses such a pump to sprinkle his garden.

The actual analogy (E) is the phenomenon of association taking place between target and base. First of all we characterise the analogy according to its *range*, Figure 4: pairing can be achieved by associating objects (A1), object attributes (A2) or relations between objects (A3). Target and base are linked directly when, stemming from the same domain, they relate to the same object or object attribute (e.g. the subject derives the height under the ceiling of the square directly from a value seen previously in a technical file). Target and base are linked indirectly when, coming from the same domain, they relate to different objects (e.g. to determine the height under the ceiling of the square, the subject bases itself on the size of an individual). Target and base are extra-contextual when they result from different domains (e.g. in case of the water pump).

The analogy can also be indexed according to its *type*, i.e. according to whether it is spontaneous (it emerges automatically from the subject's thought) or controlled (the subject expresses a priori the will to associate a base with the target, then searches – sometimes explicitly – an origin, identifies the base and ends up applying a knowledge transfer).

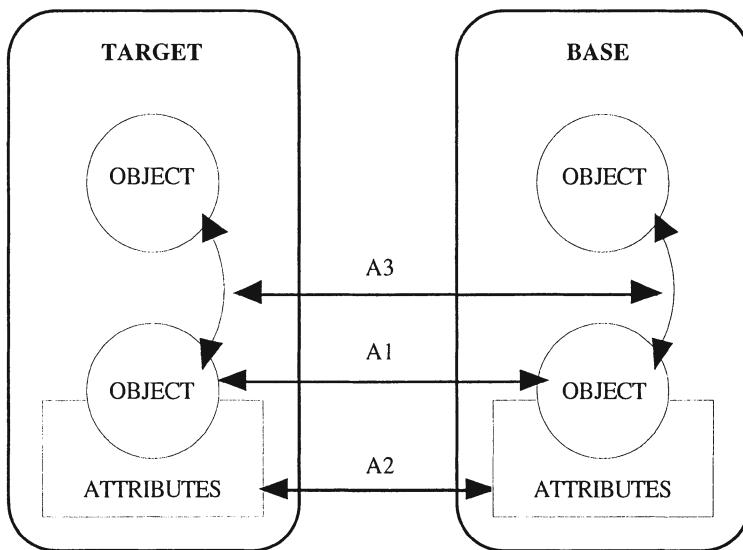


Figure 4. Range of an analogical association

In addition, it is sometimes possible to distinguish between conscious and unconscious analogies. We will refer to this as the *mode* of analogical transfer. In an unconscious mode, the subject, unlike the observers, is not even aware of the acquisition of tacit knowledge, nor of its implementation in the design process. In our example, the analogy between water pump and lift & force pump involved pairing an attribute (the shape of the lever of the 2 pumps) spontaneously and, since it was verbalised, in a conscious way.

The *usefulness* (F) of the analogy gives an account of its outcome. Did the operation lead to a success (the pairing allowed to solve the target-problem), to a failure (the adaptation between base and target field did not succeed) or to a non-use (the transfer took place, but brought nothing to the design process).

Finally, the phenomenon of *chaining* between analogies (G) allows tracking possible cascades of pairings. In these cases, the success of a first analogy establishes a new trigger, which in turn urges subjects into an analogical reflection to continue their work.

## 5. Quantitative Results

Upon completion of the experiment, we thus disposed of the handwritten description of 3 complete design processes with, for each one, the sketches and final proposal drawn by the subject, the notes of the observers and the concerted list of "analogical thoughts", analysed according to the parameters defined above. Coding these data allowed us to compile them in a dozen easily accessible summary tables. Which observations can we make from these tables?

### 5.1 ON THE NUMBER OF ANALOGIES OBSERVED

Analysis of the 3 design sessions observed reveals between 12 and 17 instances of analogy during 2,5 hours of design, Table 1. This corresponds to an average of one analogical thought every 10 minutes, which confirms the significant recourse to this design strategy. Apparently analogy concerns a frequent and permanent practice in architectural design, used on an operational and concrete level throughout the design process. This practice has nothing in common with the well-known surface and illustrative analogies – often exclusive and made up *a posteriori* – ‘justifying’ the finished design product.

TABLE 1. Number of analogies observed

| Subject           | Number |            |
|-------------------|--------|------------|
|                   |        | Time [min] |
| Julien            | 12     | 12.5       |
| David             | 15     | 10.0       |
| Cécile            | 17     | 8.8        |
| $\Sigma$ and Mean | 44     | 10.2       |

### 5.2 CHARACTERISTICS OF THE TRIGGER

Triggers are particularly difficult to capture, Table 2: a good half of them is not identifiable in our experiment. This is of course due to the impossibility of accessing the subject's thoughts instantaneously, forcing the observers to follow only their expression.

With regard to the triggers known, half of them relate to the documentation we had provided the subjects. The memory of specific information, suggested not long before the design session, thus influences directly the knowledge the designer will use.

This average observation, however, should immediately be nuanced by the case of the 3<sup>rd</sup> subject (Cecile) who resorts mainly to triggers resulting from her personal knowledge: 5 personal recourses against 2 triggers related to the documentation provided and, above all, 10 triggers not

verbalised, i.e implicit and thus probably closely related to her personal experience.

TABLE 2. Characteristics of the trigger

| Subject      | Personal | Document | Chaining | Unknown |
|--------------|----------|----------|----------|---------|
| Julien       | 1        | 5        | 0        | 6       |
| David        | 1        | 4        | 3        | 7       |
| Cécile       | 5        | 2        | 0        | 10      |
| $\Sigma$     | 7        | 11       | 3        | 23      |
| Distribution | 16%      | 25%      | 7%       | 52%     |

### 5.3 LINK BETWEEN TARGET AND BASE

Comparison of each target and the base it is associated with provides some indications of possible links between the associated domains associated. Our observation, Table 3, suggests a majority of extra-contextual associations: base and target do not belong to the same domain. Again, this average should be considered in the light of the behaviour of the 3<sup>rd</sup> subject (Cecile) who associates 15 targets out of 17 with an extra-contextual base (88%), whereas the two other subjects make only 5 associations of this type (18%), against 22 that present a direct or indirect link between target and base.

TABLE 3. Link between target and base

| Subject      | Direct | Indirect | Extra-<br>contextual |
|--------------|--------|----------|----------------------|
| Julien       | 5      | 5        | 2                    |
| David        | 9      | 3        | 3                    |
| Cécile       | 1      | 1        | 15                   |
| $\Sigma$     | 15     | 9        | 20                   |
| Distribution | 34%    | 20%      | 45%                  |

### 5.4 ORIGIN OF THE ANALOGY

On average, the origin of the analogy seems to lie more in personal references than in the documentation provided, Table 4. However, also here the influence of the 3<sup>rd</sup> subject is significant. Without her, the distribution of the origins is completely reversed: 2/3 of the origins would result from information given before the design session.

This suggests the existence of 2 analogical strategies: one based on specific knowledge directly related to the design's constraints, the other drawing its sources from personal knowledge. We will pick up this distinction again in the discussion.

TABLE 4. Origin of the analogy

| Subject      | Document | Personal |
|--------------|----------|----------|
| Julien       | 8        | 4        |
| David        | 9        | 6        |
| Cécile       | 2        | 15       |
| $\Sigma$     | 19       | 25       |
| Distribution | 43%      | 57%      |

## 5.5 LINK BETWEEN TRIGGER AND ORIGIN OF THE ANALOGY

In addition, we wished to investigate whether there exists a relation between an analogy's trigger and origin, corresponding to the contexts of its target and base respectively. Unfortunately, 7 times out of 10, this link cannot be identified, mainly because, as already mentioned in 5.2, very often the target remains implicit and is thus not verbalised.

## 5.6 RANGE OF THE ANALOGIES: OBJECT, ATTRIBUTE OR RELATION

What is the range of the analogies observed? Pairing between target and base is chiefly accomplished by associating two objects based on one of their attributes (66% of the analogies on average and 76% for 2 subjects). One time out of three, Table 5, pairing is achieved via a similar relation between several objects in target and base domain. Rare are the cases where the analogy involves a direct mapping of the target object onto a base object (5% of the analogies).

TABLE 5. Range of analogies observed

| Subject      | Objects | Attributes | Relations |
|--------------|---------|------------|-----------|
| Julien       | 1       | 9          | 2         |
| David        | 0       | 7          | 8         |
| Cécile       | 1       | 13         | 3         |
| $\Sigma$     | 2       | 29         | 13        |
| Distribution | 5%      | 66%        | 30%       |

### 5.7 SPONTANEOUS VERSUS CONTROLLED ANALOGIES

The analogical thoughts mainly appeared spontaneously (70%) and almost constantly for all subjects, Table 6. The tactics of resorting to analogy in design are thus a privileged activity of the mind: generally, subjects do not decide to resort to analogy deliberately, but are guided by their imagination.

TABLE 6. Spontaneous versus controlled analogies

| Subject      | Controlled | Spontaneous |
|--------------|------------|-------------|
| Julien       | 3          | 9           |
| David        | 4          | 11          |
| Cécile       | 6          | 11          |
| $\Sigma$     | 13         | 31          |
| Distribution | 30%        | 70%         |

### 5.8 (UN)CONSCIOUS ANALOGIES

By using the "think aloud" method, subjects are forced, as much as possible, to verbalise every single of their thoughts. Therefore we supposed this experiment to deal with completely conscious analogies only, as only those that are expressed can be captured. To our surprise, however, 2 out of the 44 analogies observed turned out to be unconscious!

These two exceptional captures were possible thanks to the sketches and the certainty that subjects had no experience with sailing. The 2 unconscious analogies were posed by the same subject in relation to the same object: they concern the shape of the deck of the sailing boat, which is drawn very accurately twice during his design session. In section, this shape is particular so as to meet 3 requirements: (i) to provide the greatest height possible in the square, under the deck; (ii) to allow the team-members to pass on the deck from the cockpit to the front for manoeuvring the veils and (iii) to quickly evacuate the water the waves leave on the deck. A designer who has never navigated cannot produce a deck of the adequate shape without a long reflection, sullied with trials and errors... unless being inspired directly by a shape seen in the documentation provided. By consequence, the fact of drawing twice, from the first strike, the correct shape of the deck without mention may be taken to betray the unconscious borrowing of specific knowledge.

### 5.9 CHAINING OF ANALOGIES

The sequence of analogical thoughts remains a marginal phenomenon in our experiment: only 10% of the cases and each time for no more than a doublet of analogies. Note, however, that these sequences only took place after the 1<sup>st</sup> analogy met with success. No sequence occurred after a failure of base-target transfer.

## 6. Discussion: About the Effectiveness of Analogies

### 6.1 GENERAL SUCCESS RATE

Table 7 lists the effectiveness of the analogies in design: 3 times out of 4, they meet with success! In these cases, the link between target and base is valid, retained and applied in an adaptation phase that fully succeeds.

Analogical reasoning is thus a generally profitable strategy during design. In our experiment, the failure rate is limited to 14% of the attempts. Our data do not allow detecting easily whether failures are due to mapping the target onto an erroneous base, or whether it is the adaptation phase that fails and thus causes the attempt to be abandoned. In the latter case, the very limited time left to the subjects to propose a complete configuration for the boat may have weighed heavily on the maximum time devoted to the adaptation. Note, however, that this failure rate is of the same range as the "not used" rate (11%): certain analogical thoughts arouse from the mind some ideas that have nothing to do with and are of no use to the design problem. For instance, upon reading the sentence "cruising on the Aegean sea", one of the subjects evoked the state of emergency issued in Macedonia, neighbouring country of Greece. No relationship whatsoever with the interior configuration of a sailing yacht!

TABLE 7. General success rate

| Subject      | Success | Failure | Not used |
|--------------|---------|---------|----------|
| Julien       | 11      | 0       | 1        |
| David        | 11      | 3       | 1        |
| Cécile       | 11      | 3       | 3        |
| $\Sigma$     | 33      | 6       | 5        |
| Distribution | 75%     | 14%     | 11%      |

## 6.2 SUCCESS AS A FUNCTION OF THE LINK BETWEEN TARGET AND BASE

Can the success of the analogies be differentiated according to the type of link between target and base domains? Table 8 suggests that success is characteristic of within- or between-domain analogies in the same proportions (39 and 36%)! One would thus obtain the same effectiveness with thoughts from the same domain as the target object as with thoughts that are completely unrelated! An analogy emerging from a direct link between target and base leads more than 8 times out of 10 to success, Table 9. This exceptional rate may be explained by the deadline imposed on the subjects: they often tried to solve a sub-problem of the design by directly borrowing a solution from documentation available at hand. We are probably, in these cases, at the limit of a real analogical thought. In our opinion, the between-domain associations are more authentic in this respect.

TABLE 8. Success as a function of the link between target and base

|          | Direct | Indirect | Extra-<br>contextual | Direct | Indirect | Extra-<br>contextual |
|----------|--------|----------|----------------------|--------|----------|----------------------|
| Success  | 13     | 8        | 12                   | 39%    | 24%      | 36%                  |
| Failure  | 2      | 1        | 3                    | 33%    | 17%      | 50%                  |
| Not used | 0      | 0        | 5                    | 0%     | 0%       | 100%                 |

These are profitable in 60% of the cases against a failure rate of 15% and a useless rejection of analogies of 25%: 1 idea out of 4 would thus be unusable, but nearly 2 out of 3 can effectively contribute to the design process.

TABLE 9. Success as a function of the link between target and base

|          | Direct | Extra-<br>contextual | Direct | Extra-<br>contextual |
|----------|--------|----------------------|--------|----------------------|
| Success  | 13     | 12                   | 87%    | 60%                  |
| Failure  | 2      | 3                    | 13%    | 15%                  |
| Not used | 0      | 5                    | 0%     | 25%                  |
| $\Sigma$ | 15     | 20                   | 100%   | 100%                 |

### 6.3 SUCCESS AS A FUNCTION OF THE RANGE

69% of the analogies established through association of an attribute lead to success, against 92% of those achieved by a corresponding relation in base and target domain: analogical thinking can thus lead to success by using complex associations, Table 10. Apparently, the success rate would even rise with the degree of complexity of the link connecting target and base! This suggests that the cognitive activity put into operation during analogical reasoning in design by far exceeds the simplistic representations of current theories. By consequence, research on case-based design (CBD) should concentrate more on sophisticated modes of pairing concepts.

TABLE 10. Success as a function of the range

|          | Objects | Attributes | Relations | Objects | Attributes | Relations |
|----------|---------|------------|-----------|---------|------------|-----------|
| Success  | 1       | 20         | 12        | 50%     | 69%        | 92%       |
| Failure  | 0       | 5          | 1         | 0%      | 17%        | 8%        |
| Not used | 1       | 4          | 0         | 50%     | 14%        | 0%        |
| $\Sigma$ | 2       | 29         | 13        | 100%    | 100%       | 100%      |

### 6.4 SUCCESS OF SPONTANEOUS AND CONTROLLED ANALOGIES

The success ratios of the spontaneous and controlled analogies are equivalent (74 and 76%, Table 11). The former, however, are twice as numerous and arise only under injunction of the subject's intentions: the spontaneous options are thus targeted and produced by the mind with the confidence of a raised effectiveness. In other words, the 'freewheeling mind' does not spout no matter what associations of ideas; it produces analogies that are as beneficial as those generated by controlled reflection.

TABLE 11. Success of spontaneous and controlled analogies

|          | Controlled | Spontaneous | Controlled | Spontaneous |
|----------|------------|-------------|------------|-------------|
| Success  | 10         | 23          | 77%        | 74%         |
| Failure  | 3          | 3           | 23%        | 10%         |
| Not used | 0          | 5           | 0%         | 16%         |
| $\Sigma$ | 13         | 31          | 100%       | 100%        |

### 6.5 SUCCESS AS A FUNCTION OF THE ORIGIN

Not counting unused attempts, analogies originating in personal experience show the same success rate (84%) as those resulting from knowledge acquired through the documentation provided before the experiment, which is, remember, directly related to the problem situation at stake. It should be noted that no analogy was established starting from the documents with extra-contextual information: neither Space Lab, nor the sleeping cars have, for example, lead to an analogical thought. The subjects thus get an idea either in their personal general knowledge, or in new knowledge, but only if they directly relate to the object of their design.

## 7. Tentative Conclusion

The use of analogy in architectural design is typically discussed and studied in terms of (its impact on) the resulting design *product*. By contrast, this paper has reported a study that adopts a different perspective on the phenomenon, by switching attention to the design *process*.

In order to identify and study analogy from this point of view, we have proposed a scheme that characterises an analogical transfer according to 9 parameters. The parameters were chosen with an eye to answering questions like when/at what point, why/for what purpose, and how/in what range or mode do architects use analogy as they design.

Subsequently, we have tried out this scheme to analyse 3 design sessions, in which architects worked on the interior configuration of a sailing yacht. As such, the scheme has proven a useful pair of glasses to observe, inventory and understand analogies throughout the design process, allowing us to formulate tentative answers to some of the questions raised above.

At what point in the design process do architects resort to analogy? The results of our analysis strongly suggest that analogical thinking is used frequently, if not permanently throughout the process. These numerous analogies may be triggered by personal knowledge in the designer's memory as well as by external information exposed to before designing. Also their origin may lie in personal or external sources. Which type prevails seems a matter of preference, yet further research is needed to clarify this issue. At this point, our results only demonstrate that architects use a mix of 2 analogical strategies: one relying on personal knowledge and experience, the other supported by external knowledge delivered 'just-in-time', i.e. not long before design starts.

As to the question why architects resort to analogy, a tentative answer inspired by our results may be: simply because it is their second nature. Indeed, a large majority of the analogies observed seem to occur spontaneously, i.e. without mention of any specific or explicit reason, suggesting analogical reasoning to be a natural, if not favourite strategy of the human mind.

Whether spontaneous or not, how exactly do these analogies unfold? According to our analysis, analogies are established both within one domain – the domain of naval design – and between different domains, the relative weight of which varies considerably across individual designers. What does not vary across individuals, however, is that the analogical transfer unfolds not so much between objects in base and target domain, but rather between object attributes or relations. The exceptionally high success rate of analogies of the latter range chimes with the more ‘structural’ theories on analogy mentioned above and may have serious consequences for the development of successful CBD systems (and other tools that aim at providing designers with relevant analog candidates). Indeed, if this success rate can be confirmed by further experiments, it suggests that the representation of cases – and other potential sources – should focus not so much on the individual objects that constitute the case, but rather on the structural relations between them.

However, we deliberately use the term *tentative* answers, for it is obvious that further evidence is needed before we can even start thinking of drawing general conclusions from our study. The experiment presented in this paper is very limited, in terms of both time (the 2.5-hour design sessions represent but a fraction of a real world design process) and number of subjects observed. Studies of this nature cannot produce results that may be generalized straightforwardly. They are nevertheless important as preliminary work in developing a more profound understanding of analogy in architectural design, which in turn should allow developing more effective design support.

As such, the study has produced useful observations and encouraged us to repeat the experiment with more and other subjects. The most encouraging observation is perhaps that on average designers resort to analogical thinking no less than 5.8 times per hour. Therefore, we cannot resist the temptation to close this paper with the (tentative) conclusion that the water lily-like ‘1 analogy per architectural masterpiece’ view, propagated by architectural and design literature, seems to do little justice to the reality of architectural design.

## Acknowledgements

The research lead by Pierre Leclercq is funded by the Research Council of the University of Liège. Ann Heylighen is a postdoctoral fellow of the Fund for Scientific Research Flanders.

## References

- Britannica.com: 2001, <http://www.britannica.com> (last visit: October 21, 2001).
- Broadbent, G: 1973, *Design in Architecture: Architecture and the Human Sciences*, Wiley and Sons, London.
- Broadbent, G: 1980, Building design as an iconic design system, in G Broadbent, R Bunt and C Jencks (eds), *Signs, Symbols and Architecture*, Wiley & Sons, New York, pp. 324–331.
- Casakin, H: 1997, *The Role of Analogies and Visual Displays in Architectural Design*, unpublished doctoral dissertation, Technion, Haifa.
- Casakin, H and Goldschmidt, G: 1999, Expertise and the use of visual analogy: implications for design education, *Design Studies* 20: 153–175.
- Finke, RA, Ward, TB and Smith, SM: 1992, *Creative Cognition*, MIT Press, Cambridge, MA.
- Gentner, D: 1983, Structure mapping: A theoretical framework, *Cognitive Science* 7: 155–177.
- Gentner, D: 1989, The mechanisms of analogical learning, in S Vosniadou and A Ortony (eds), *Similarity and Analogical Reasoning*, Cambridge University Press, Cambridge, pp. 199–239.
- Gick, ML and Holyoak, KJ.: 1980, Analogical Problem Solving, *Cognitive Psychology* 12: 485–506.
- Goldschmidt, G: 1994, Visual analogy in design, in R. Trapp (ed.), *Cybernetics and Systems '94*, World Scientific, Singapore, pp. 507–514.
- Goldschmidt, G: 1995, Visual displays for design: imagery, analogy and databases of visual images, in A Koutamanis, H Timmermans and I Vermeulen (eds), *Visual Databases in Architecture*, Avebury, Aldershot, pp. 53–74.
- Holyoak KJ: 1991, Symbolic connectionism: toward third generation theories of expertise, in KA Ericsson and J Smith (eds), *Toward Third Generation Theories of Expertise: Prospects and Limits*, Cambridge University Press, Cambridge, pp. 301–335.
- Holyoak KJ. and Thagard, P: 1989, Analogical mapping by constraint satisfaction, *Cognitive Science* 13: 295–355.
- Kolodner, JL: 1993, *Case-Based Reasoning*, Morgan Kaufman, San Mateo, CA.
- Lawson, B: 1994, *Design in Mind*, Butterworth Architecture, London.
- Neisser, U: 1964, Visual search, *Scientific American* 210: 94–102.
- Pollard, P and Evans JSBT: 1987, On the relationship between content and Context effects in reasoning, *American Journal of Psychology* 100: 41–60.
- Polya, G: 1973, *How to Solve It. A New Aspect of Mathematical Method*, Princeton University Press, Princeton.
- Rips, LJ: 1983, Cognitive process in propositional reasoning, *Psychological Review* 90: 38–71.

- Rowe, PG: 1982, A priori knowledge and heuristic reasoning in architectural design, *Journal of Architectural Education* 36(1): 18–23.
- Tulving, E and Thomson, DM: 1973, Encoding specificity and retrieval process in episodic memory, *Psychological Review* 80: 353–373.
- Vosniadou, S and Ortony, A.: 1989, Similarity and analogical reasoning; a synthesis, in S Vosniadou and A Ortony (eds), *Similarity and Analogical Reasoning*, Cambridge University Press, Cambridge, pp. 1–7.

## **EVOLUTIONARY APPROACHES IN DESIGN**

---

*An evolutionary approach to the inverse problem in  
rule-based design representations*  
Stephan Rudolph and Rolf Alber

*Evolving three-dimensional architecture form: An application  
to low-energy design*  
Luisa Caldas

*Strategic shape design*  
Toshiharu Taura, Takayuki Shiose and Ryohei Ishida

*An evolutionary framework for enhancing design*  
Kwai Hung Chan, John Hamilton Frazer and Ming-Xi Tang

## TOWARDS COMPUTATIONAL TOOLS FOR SUPPORTING THE REFLECTIVE TEAM

ANDREW W HILL, ANDY DONG AND ALICE M AGOGINO  
*University of California at Berkeley*  
*USA*

**Abstract.** The content of engineering design documentation, beyond capturing the details of the design, communicates the shared knowledge integration of the design team. In this research, we present a method to analyze design documentation for levels of shared understanding and team cohesiveness in design teams by applying the computational tools of latent semantic analysis and natural language processing. We study the design documentation from students in a multidisciplinary, graduate-level product design and development course. The results show promise in measuring cohesiveness and shared understanding in design teams by analyzing their documentation and correlating metrics of effective communication to the likelihood of successful outcomes of the design team.

### 1. Introduction

#### 1.1 MOTIVATION

A dialectic exists between two prevailing paradigms of design: Simon's (Simon 1996) rational problem solving process and Schön's (Schön 1983) reflective practice. Whereas Simon's viewpoint is prescriptive and positivistic, Schön's is constructivist and phenomenological (Dorst 1997). In solving a complex design problem, the rational problem solving practitioner might propose the decomposition of the design problem into reasonably independent subproblems for which a solution could be built from known scientific and engineering principles (Pahl and Bietz 1996), whereas the reflective practitioner might suggest re-framing the problem in a way that opens up the possibility of moving in a new design direction.

While, in the context of this paper, it is not necessary to ascribe to a particular theory of design, we cite these theories to highlight the roles of rational problem solving and reflection in design. Although reflective

practice may better describe design teams in practice (Valkenburg 2000), in collaborative, team-based design environments, successful design outcomes require both rational problem solving and the knowledge integration (shared understanding) of the design team. High-performance teams excel in developing shared understanding (Cooper and Kleinschmidt 1995; Griffin 1995) through cooperative exchange of information and mutual agreements (Citera et al. 1995). Social interaction in the design process is a significant determinant of the success of collaborative design (Bucciarelli 1994) and is likely as equal a determinant as the expertise of the team.

Cross-functional teams exist in nearly all firms engaged in design, such as product development and architecture (Henke et al. 1993). Ideally, at each stage of the design process, a team would be able to draw upon and reflect upon all of their collective knowledge to determine the best course of action. But the very *raison d'être* of cross-functional design teams – bringing the required skills to solve design problems “upstream” in the design process – is also the primary challenge of design teams. Incompatible viewpoints among design team members and failure to negotiate different design perspectives and specialties may result in ineffective collaboration. In practice, numerous social barriers exist, such as ineffective leadership, lack of shared objectives and cultural heterogeneity. Without a strong product champion or manager to detect transgressions and facilitate their resolution, disagreements could lead to a break down of the team. Helping teams to form a shared understanding is an important management role.

However, with the increasing trend to virtual and distributed teams, the manager and all team members may not be available for in-person monitoring and team self-evaluation. While numerous computer-supported collaborative work tools exist to foster collaboration and enhance rational problem solving, few tools exist to help teams reflect on their process. Part of the reason for the lack of these tools is that formal methods for cognitive modeling of the psychosocial behavior of design teams have not been extensively examined. Advancements in understanding how design teams acquire and maintain their collective identity would improve our ability to build tools that assist teams to reflect on their process.

For this reason, we have been studying computational methods to better understand the attitudes and behavior of teams in real-time in order to build tools that will have an evaluative impact on dealing with the nuances of the interactions as they occur and provide mechanisms for constructive in-process improvements of team performance. We are developing reflective, evaluative tools for individuals seeking to improve effective communication in their teams so that the communication may lead to increased shared understanding and cohesiveness.

In this paper, we studied the documentation of a group of nine collaborative design teams, collectively totaling 37 men and 7 women, in the context of a multidisciplinary, graduate course in product design and development. The student teams attend a course that emphasizes customer-driven product development and blends the study of design theories and methods with execution of a product development project<sup>1</sup>. Students from computer science, engineering, business and information management and systems join forces on product development teams of four to five members coached by faculty and professional designers from industry. Because the students self-select both the product and the make-up of the team, each student brings his or her own disciplinary perspective to the team effort, and must learn to synthesize that perspective with those of the other students in the group to develop a sound, marketable product. Part of the learning in this course is for the students to assess patterns of cooperation and team dynamics and to reflect on both the behavioral and organizational challenges the teams faced. Large amounts of customer feedback and user input are encouraged through the process and generally teams do a quality job of this.

Within each team, the students' worked collaboratively on their design product and project deliverables, which consisted primarily of design documentation and reflection on the team's execution of the process at each stage of the product development process. We used these documents to measure the shared understanding and cohesiveness of the design teams. The teams utilized the course management system from Blackboard, Inc. to share design documents with each other and the faculty. Design documents were captured and analyzed using the computational linguistic tools of latent semantic analysis (LSA) (Landauer et al. 1998) and natural language processing. We developed and tested a method to ascertain the level of shared understanding and cohesion of these teams based on their design documentation.

## 1.2 THEORETICAL FRAMEWORK AND HYPOTHESES

To a large extent, the theoretical underpinnings for our research in teams rest on social network theory and social interaction theory (Adams 1967; Cook and Whitmeyer 1992; Wasserman and Galaskiewicz 1994). Questions surrounding social cohesion (Moody and White 2000) – understanding the foundations of group formation and interaction – have engaged sociologists extensively. Relationships within these social networks have as an integral element “intimate communication.” The purpose of communication in a team is to establish a set of coherent ideas and “get them across” to design

---

<sup>1</sup> <http://best.me.berkeley.edu/~aagogino/me290p/me290p.html>

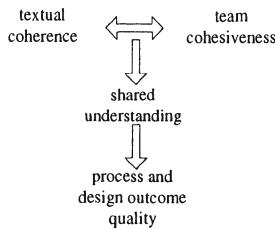
team members and stakeholders. As opposed to a merely passive purpose, namely the passive transmission and reception of information, the communication serves an active role of generating new meaning. Communication in a social setting is often characterized as the creation of shared understanding through interaction among people.

Groups of people who communicate often and over long periods of time form “semantic communities” (Robinson and Bannon 1991) with their own conventions of meaning. Similarly, Bødker and Pedersen (1991) define the term “workplace cultures” to describe a group of workers that share a common “code of conduct.” Research in information use in design (Baird et al. 2000; Lloyd 2000) show that words and phrases used by designers in the design process often captured personal experience and contributed to a wider narrative at the team, project or corporation level.

Because design knowledge is generated, codified, and reflected on in an ongoing process between the various stakeholders, teams must be able to synthesize this shared knowledge into a shared understanding in order to assure successful outcomes. Our approach to identifying the correlation between design documents and shared understanding is to detect the underlying patterns of word choices and meaning that express the shared understanding of a group of communicators. The premise is that textual coherence is both an indicator and measure of team cohesiveness when that text is a product of intra-team communication.

The ability of a team to work cohesively together towards a common goal plays a large role in the overall success of the team. There are many factors that affect the cohesiveness of a team such as commitment to the task, group pride and interpersonal relationships. Nonetheless, there is a significant correlation between cohesiveness and performance (Neck and Manz 1994). Observational studies and surveys of teams have shown that many of the problems that occur in a team arise from team members (or the entire team itself) becoming lost or diverging (McDonough et al. 2000). Thus, to measure the level of shared understanding, we hypothesize that *the cohesiveness of communication in design teams is a measure of the level of shared understanding and commitment of the team.*

Given this assertion, we postulate that teams with a high level of shared understanding will exhibit both design process quality and design outcome quality. Figure 1 illustrates the relationship between document coherence, team cohesiveness, shared understanding, and process and design quality outcome.



*Figure 1.* Correlation between document coherence and shared understanding

A distinction is made between process quality, which relates to how a team performs as a team, and outcome, which relates to the quality of the designed artifact the team produced. To measure the level of shared understanding, we developed a technique for investigating textual coherence using latent semantic analysis. We cross-validated our technique for measuring shared understanding with expert reviews of the design teams, looking for correlations between our measurements of high shared understanding (as exhibited by high textual coherence) and the experts' assessment of high quality team process and design outcome.

We propose two methods to measure the cohesiveness of team communication. In the first method, we analyze the variance of the documents, the presumption being that shared understanding and variance have an inverse correlation. The second method analyzes the coherence of the documentation. The presumption is that coherence and shared understanding have a direct correlation and that teams with a strong shared understanding will distinguish themselves from other teams. Analytically, the variance measures the correlation among a team's documents whereas coherence measures the distinctiveness between one team's entire set of documentation and that of another team.

## 2. Methodology

### 2.1 GENERAL METHODOLOGY

The general approach to this study involves the capture of design documents such as mission statements, customer surveys, concept descriptions, concept selection rationale, prototype description and test plans, and design evaluation, generated within a design team working on a collaborative design project. Applying latent semantic analysis to the corpus of collected text requires computing singular vectors to model each document. Correlation analysis of the singular vectors reveals attributes of the teams'

communication patterns. The results of the correlation analysis will be compared to faculty and professional design judges' reviews of the design teams' process and outcomes.

## 2.2 TEXT PROCESSING

The teams' design documents as described in Section 0 were stored in a database. A word-by-document matrix, an example of which is shown in Table 1, was generated from this corpus by extracting key words from the natural language text. This word-by-document matrix  $F$  is comprised of  $n$  words  $w_1, w_2, \dots, w_n$  in  $m$  documents  $d_1, d_2, \dots, d_m$ , where the weights indicate the total frequency of occurrence of term  $w_p$  in document  $d_q$ .

TABLE 1. Word by Document Matrix

|       | $d_1$ | $d_2$ | ... | $d_m$ |
|-------|-------|-------|-----|-------|
| $w_1$ | 0     | 1     |     | 2     |
| $w_2$ | 1     | 0     |     | 0     |
| ...   |       |       |     |       |
| $w_n$ | 0     | 1     |     | 1     |

We then convert the frequency counts to a log-entropy weighted word document matrix  $X$  using Equation 1.

$$X_{pq} = \frac{\log(F_{pq} + 1)}{-\sum_{1-m}((\sum_{1-m} F_{pq}) * \log(\sum_{1-m} F_{pq}))} \quad (1)$$

where  $p=1\dots n$  and  $q=1\dots m$ .

## 2.3 LATENT SEMANTIC ANALYSIS

Latent semantic analysis (LSA) (Landauer et al. 1998) is a text analysis method that extracts the meaning of documents and of words by applying a statistical analysis to a large corpus of text to detect semantic similarities. The mathematical foundation for LSA lies in singular value decomposition (SVD), a method for reducing the dimensions of a matrix. The baseline theory for LSA is that by looking at the entire range of words chosen in a wide variety of texts, patterns will emerge in terms of word choice as well as word and document meaning. LSA is unique in its method of analyzing text; there is no consideration of word order or syntax. LSA was chosen as an analysis tool because of its demonstrated successes in identifying contextual meanings of documents, identifying the voice of a given document's author, and for analyzing the cognitive processes underlying

communication (Landauer 1999). Because identifying consensus and shared understanding in a corpus of communication requires more than simply correlating documents through key word similarity, LSA is an important computational tool to explore theoretically and empirically the links between meaningfulness of words and their social and intentional existence. LSA tells us that the higher-order associations contained in the usage of words generate social phenomena, such as “self”, “personal identity” and “social group perception” (Landauer 1999).

It is important to note that we utilized latent semantic analysis to measure textual coherence – not textual cohesiveness. Cohesiveness in text (lexical cohesion) is based on the pragmatics of grammar, or “the ties that bind a text together” (Halliday and Hasan 1976) whereas textual coherence relates to the thematic consistency. That is, a coherent text presents (a) unified concept(s) “about the speaker’s or author’s purpose” (Fillmore 1974). One recent study has demonstrated a structural link between lexical cohesion and textual coherence (Harabagiu 1999).

As an example of the contrast between textual coherence and cohesiveness, consider the two excerpts from design documents by Team 5. The first (A) comes from the original project proposal by a team member, whereas the second (B) is the product description from the second draft of the product mission statement.

A: *“There are very many different types of products that come under the description of roof racks and most car owners would admit to owning at least one system.”*

B: *“To provide a stylish, secure, reconfigurable storage and organizational platform for passenger cars, SUV’s, wagons, and minivans, allowing active, recreational users fast and easy access to stuff.”*

In this example, the A is written in passive prose, whereas B is written as a directive. The sentences also utilized different grammatical constructs. Even though there is weak evidence of textual cohesiveness, there appears to exist a consistency of theme, storing and organizing items in cars. In fact, if all the documents authored by a team were collapsed into one, the grammar and structure of the various authors would likely not be textually cohesive. We used LSA to measure textual coherence, which we regard as a measure of shared understanding when that text arises from intra-team communication and design documentation.

Also, LSA cannot detect prosody in textual communication, that is, tone and rhythm. Prosody is an important device for accomplishing cohesion in spoken interaction that in writing is exhibited through the use of certain grammatical constructs and punctuation (Gumperz et al. 1984). Nor could LSA pick up on argumentation in text. If the text prefaced with terms such

as “disagree” or “counter point,” yet dealt with a single concept or theme, our method based on LSA would measure an equivalent textual coherence to a document that prefaced text with terms such as “complete agreement” and “on the same page.” Yet, one would hardly characterize a team authoring the former documents as cohesive. Other techniques would need to be applied to detect argumentation (Raccah 1995).

Singular value decomposition is applied to the log-entropy weighted word-by-document matrix  $X$  shown in Equation 1 resulting in three component matrices using the relationship  $X = SVD^T$  where  $S$  is the  $n \times m$  matrix of left *singular vectors*. The matrix dimension will typically be in the thousands to tens of thousands. However, the useful feature of LSA, in particular to SVD, is that one is able to discount all but about the first 300 singular values in each singular vector without losing much descriptive information. One aspect in applying LSA is determining the sufficient number of dimensions to retain to adequately capture context while reducing the number of retained dimensions to the smallest number possible in order to minimize computational complexity. The exact number of singular vectors to retain is subject to change as part of the algorithm tuning. These singular vectors  $s_1, s_2, \dots, s_k$  are the basis for the remaining computations. Table 2 describes which singular vectors are retained for analysis in each Case A through J.

#### 2.4 METHOD ONE: VARIANCE OF DOCUMENTS

To quantify shared understanding in design teams, we build upon research documented by Hill (Hill et al. 2001). Hill successfully applied latent semantic analysis upon a corpus of design documents to identify topical and voice similarities in design documents. These similarities, when combined, allow for the identification of a shared understanding. In order to identify teams that share a vision, we utilize latent semantic analysis and calculate the *s-dimensional* radius of a set of design documents generated by a given design team, where  $s$  is the number of singular values being retained for the given analysis. Because shared understanding – comprised of topical and voice similarity – is a self-defined signature of a team, this method identifies the level of a team’s shared understanding by comparing the magnitudes and deviations of the *s-dimensional* radius of the documents in the LSA space. Conceptually, the centroid of the documents represents the shared understanding; dilution of that shared understanding would manifest through the variance of the radii and distance to outlying documents.

TABLE 2. Experimental case descriptions

| Case Descriptions | Singular Values Included in Analysis |
|-------------------|--------------------------------------|
| Case A            | Singular Values 1-10                 |
| Case B            | Singular Values 2-11                 |
| Case C            | Singular Values 1-30                 |
| Case D            | Singular Values 2-31                 |
| Case E            | Singular Values 1-100                |
| Case F            | Singular Values 2-101                |
| Case G            | Singular Values 1-200                |
| Case H            | Singular Values 2-201                |
| Case I            | Singular Values 1-300                |
| Case J            | Singular Values 2-301                |

Based on this, our algorithm proceeds as follows:

- 1) Define  $t$  matrices  $m_i$ ,  $i=1\dots,t$ , denoting the total number of unique teams represented in the original word document matrix. Place all documents  $n_i$  generated by team  $i$  in the matrix  $m_i$ . Thus, the matrix  $m_i$  has  $v_i$  columns, the number of singular values, and  $n_i$  rows, the number of documents.
- 2) Calculate the centroid  $c_i$  of each team's document set by finding the average value of each column of  $m_i$ .

$$c_i = \frac{\sum_{k=1}^{v_i} s_{k,i}}{n_i} \quad (2)$$

where  $s_{k,i}$  is the singular vector  $k$  representing a document produced by team  $i$ .

- 3) Calculate maximum (max), mean ( $\mu$ ) and standard deviation ( $\sigma$ ) of the  $s$ -dimensional radius for each team.

$$\max_i = \max_{n_i} \left[ \sqrt{\sum_{k=1}^v (s_{k,i} - c_i)^2} \right] \quad (3)$$

$$\mu_i = \frac{\sqrt{\sum_{k=1}^{v_i} (s_{k,i} - c_i)^2}}{n_i} \quad (4)$$

$$\sigma_i = \frac{\sqrt{\sum_{n_i} ((s_{k,i} - c_i) - \mu_i)^2}}{n_i} \quad (5)$$

We expect that teams that exhibit “good” shared understanding will have a smaller standard deviation and a lower maximum relative to the value of the mean of teams that exhibit “poor” shared understanding.

## 2.5 METHOD TWO: TEXTUAL COHERENCE

To quantify textual coherence, we build upon the work of Dhillon and Mohda (1999) who developed a technique for clustering very large sets of documents using LSA. Dhillon and Mohda determined that the coherence of a set of documents is simply a measure of the  $L^2$  norm of the document vectors in the set. Analytically, the norm determines whether all singular vectors for a team’s set of documents are identical. In other words, it is as if all the authors were writing expressing the same sentiment but in slightly different styles. We hypothesize that the coherence of a set of design documents is indicative of the cohesion of the design team. The idea is that cohesiveness of the team will be expressed in the generated documents as coherence. To do this analysis, the set of singular vectors generated from the documents of each team are grouped and the coherence is measured. The following method was employed.

- 1) Define  $t$  matrices  $m_i$ ,  $i=1\dots t$ , denoting the total number of unique teams represented in the original word document matrix. Place all documents  $n_i$  generated by team  $i$  in the matrix  $m_i$ . As with the variance of documents, the matrix  $m_i$  has  $v_i$  columns and  $n_i$  rows.
- 2) Calculate the cohesiveness  $\chi$  of each group, which according to Dhillon and Modha, is equivalent to computing the  $L^2$  norm of the sum of the singular vectors representing the document set for each team.

$$\chi = \left\| \sum_{n_i} s_i \right\| \quad (6)$$

## 2.6 QUALITATIVE COMPARISONS

To validate our quantitative metrics, qualitative measures of the process and outcome quality for each team were taken into consideration. Each team is assigned a Rank from 1 to 9, 1 being the best and 9 the worst. The Rank is based on the cumulative assessments of ten professional product designers on the quality of the team's final product (in terms of the final product satisfying the design objectives as stated by the mission statement) and of the team's process quality (i.e., how well the team executed the product development process and functioned as a team). There were seven criteria: mission statement, customer and user needs, concept generation, concept selection, prototype feedback, financial analysis and final prototype. For each criterion, the judges assigned a score from 1 (worst) to 5 (best) based on recommended rubrics provided by the faculty. For example, in assessing a team's mission statement, the judge would examine whether the mission statement communicated the intent of the students' project, the clarity of the mission statement, and the description of the target market and business goals of the product. The judges based their assessments on 10-minute formal presentation by the team, a notebook recording the process undertaken, the final prototype, and informal discussions between the judges and each team during a three-hour "trade show." We then correlated the quantitative measures with the qualitative assessments.

To assure validity of the quantitative measure, we followed a protocol that maintained the anonymity of the teams and of the team members and removed potential bias in the data that might result from the teams' knowledge of the purpose of the study. First, we removed all the names of the teams and the names of the authors of the documents from the analyses. Second, the results of these analyses did not figure into the grading of the student teams. The instructors and judges had no prior knowledge of the analysis results. These practices ensured that the teams did not "massage" the collected data in an attempt to improve their grading.

## 3. Experiment and Results

### 3.1 DATA SET

Each design team submitted 18 design documents as part of their product development deliverables. A total of 135 documents were submitted. Each group turned in between 13 and 18 documents because some teams combined deliverables into one document. The documents types ranged from mission statements to financial analysis of the project to customer and user feedback summaries. The activities of the groups are summarized in

Table 3. Table 3 shows how many design documents were generated, as well as how many team members contributed a significant number of documents. While the contributing member statistic is not a direct reflection of how the work was distributed among the team – in fact, it is only a measure of who turned the document in – it does provide insight into how the team distributed tasks.

TABLE 3. Summary of group activity

| Team                     | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  |
|--------------------------|----|----|----|----|----|----|----|----|----|
| No. Documents Submitted  | 14 | 15 | 17 | 13 | 15 | 18 | 16 | 14 | 13 |
| No. Contributing Members | 2  | 2  | 5  | 3  | 5  | 4  | 5  | 4  | 3  |

### 3.2 QUALITATIVE MEASURES

The results of the judging are shown in Table 4. All judges' ratings were included to establish the ranking. The teams are ranked in order from highest 1, to lowest, 9. In Table 4 and as in all subsequent tables, the top five performing teams are indicated in **bold**.

The results of our quantitative analyses for shared understanding and cohesiveness were compared to the judges' assessments by identifying how each team performed relative to the median value of the metric. For example, in method one, we calculated the standard deviation of the s-dimensional radius for each team.

TABLE 4. Team ranking

| Team | 1        | 2 | 3        | 4 | 5        | 6 | 7        | 8        | 9 |
|------|----------|---|----------|---|----------|---|----------|----------|---|
| Rank | <b>5</b> | 7 | <b>3</b> | 9 | <b>1</b> | 6 | <b>2</b> | <b>4</b> | 8 |

We then calculated the median value. Those teams that performed better than or equal to the median are indicated with a '+' in the table of results, while those that performed worse than the median are indicated with a '-'. If we had perfect correlation between the judges assessment and the measurements, then the top 5 rated teams would show a plus and the bottom four would show a negative. Based on the correlation, we hope to draw some conclusions about the correlation between shared understanding and cohesiveness with process and design quality outcomes. More importantly,

correlation between the quantitative measures and the judges' assessments would offer an initial basis for validating our methodology.

### 3.3 RESULTS OF METHOD ONE

In order to analyze the data, we generated tables summarizing each experimental case. The results of the document variation analyses (method one) are summarized in Tables 5, 6 and 7.

TABLE 5. Mean Euclidian distance in LSA space

| Mean<br>Distance<br>Median | Case A | Case B | Case C | Case D | Case E | Case F | Case G | Case H | Case I | Case J | Average |
|----------------------------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|---------|
| <b>Group 1</b>             | +      | +      | +      | +      | +      | +      | +      | -      | -      | -      | +       |
| Group 2                    | +      | -      | +      | +      | +      | +      | -      | -      | -      | -      | -       |
| <b>Group 3</b>             | -      | -      | -      | -      | +      | +      | -      | -      | -      | -      | -       |
| Group 4                    | -      | -      | -      | -      | -      | -      | -      | -      | -      | -      | -       |
| <b>Group 5</b>             | -      | -      | -      | +      | +      | +      | -      | -      | -      | -      | -       |
| Group 6                    | +      | +      | +      | +      | -      | -      | +      | +      | +      | +      | +       |
| <b>Group 7</b>             | -      | -      | -      | -      | -      | -      | +      | +      | +      | +      | +       |
| <b>Group 8</b>             | -      | +      | -      | -      | -      | -      | +      | +      | +      | +      | -       |
| Group 9                    | +      | +      | +      | -      | -      | -      | +      | +      | +      | +      | +       |

TABLE 6. Maximum Euclidian distance in LSA space

| Max<br>Distance<br>Median | Case A | Case B | Case C | Case D | Case E | Case F | Case G | Case H | Case I | Case J | Average |
|---------------------------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|---------|
| <b>Group 1</b>            | +      | +      | +      | +      | +      | +      | -      | -      | -      | -      | +       |
| Group 2                   | +      | +      | +      | +      | +      | +      | -      | -      | -      | -      | +       |
| <b>Group 3</b>            | -      | -      | -      | -      | +      | +      | +      | +      | -      | -      | +       |
| Group 4                   | +      | -      | +      | +      | -      | -      | -      | -      | -      | -      | -       |
| <b>Group 5</b>            | +      | +      | +      | +      | +      | +      | -      | -      | -      | -      | +       |
| Group 6                   | -      | -      | -      | -      | -      | -      | -      | -      | +      | -      | -       |
| <b>Group 7</b>            | -      | -      | -      | -      | -      | -      | +      | +      | +      | +      | -       |
| <b>Group 8</b>            | -      | -      | -      | -      | +      | +      | +      | +      | +      | +      | -       |
| Group 9                   | -      | +      | -      | -      | -      | -      | +      | +      | +      | +      | -       |

TABLE 7. Standard deviation of Euclidian distance in LSA space

| St. Dev<br>Distance<br>Median | Case A | Case B | Case C | Case D | Case E | Case F | Case G | Case H | Case I | Case J | Average |
|-------------------------------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|---------|
| <b>Group 1</b>                | +      | +      | +      | +      | -      | -      | -      | -      | -      | -      | +       |
| Group 2                       | +      | +      | +      | +      | +      | +      | -      | -      | -      | -      | +       |
| <b>Group 3</b>                | -      | -      | -      | -      | +      | +      | +      | +      | -      | -      | +       |
| Group 4                       | +      | +      | +      | +      | -      | -      | -      | -      | -      | -      | -       |
| <b>Group 5</b>                | -      | -      | -      | +      | +      | +      | +      | +      | +      | +      | +       |
| Group 6                       | -      | -      | -      | -      | -      | -      | -      | -      | +      | +      | -       |
| <b>Group 7</b>                | -      | -      | -      | -      | +      | +      | +      | +      | +      | +      | -       |
| <b>Group 8</b>                | +      | -      | +      | -      | +      | +      | +      | +      | +      | +      | +       |
| Group 9                       | -      | +      | -      | -      | -      | -      | +      | +      | +      | +      | -       |

As can be seen in these three tables, the measure of increasing standard deviation correlates well with decreasing shared understanding. Cases E and F had the highest accuracy. They correctly identified Teams 3, 5, 7, 8 as having better than average shared understanding and Teams 4, 6 and 9 as having lower than average shared understanding. It mis-categorized Teams 1 and 2. Thus, on average, it had an accuracy of 78%.

Figure 2 illustrates the difference between levels of shared understanding. Team 3 created a superior product relative to Team 9 and had a superior process. Graphically, this manifests as greater dispersion in the documents (shown as 2-D LSA vectors) generated by Team 9 compared to Team 3. Since Cases E and F had the highest accuracy, we conclude that retaining the first 100 dimensions of the singular vectors (as shown in Table 2) is sufficient to capture the context of the documents. We will test this assertion again with method two.

The prior analysis tries to match the quantitative metrics from our studies to the qualitative scores given by judges. Examining the comments that judges made relative to high performing teams and low performing teams can have further insight into the analysis. The following are comments given to Team 3, a high performing team:

- “Direct, Specific and Clear”
- “Well thought out process to determine customer needs and generate concepts.”

“Good rigorous methods to organize all the features and combinations.”

All told, the comments about Team 3, and about the high performing teams in general, indicated a positive impression of the team’s vision and how the vision was executed. When contrasted with comments about low

performing teams, as follows about Team 9, a clear difference can be seen in how this team's vision was created and executed. Comments about Team 9 are as follow:

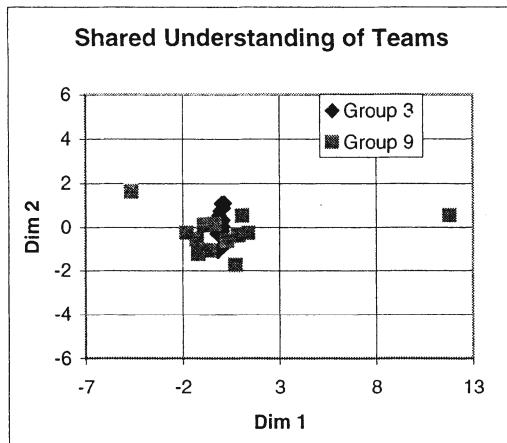


TABLE 8. Cohesiveness

| Coherence<br>Median | Case A | Case B | Case C | Case D | Case E | Case F | Case G | Case H | Case I | Case J | Average |
|---------------------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|---------|
| <b>Group 1</b>      | -      | -      | +      | +      | +      | +      | +      | +      | +      | +      | -       |
| <b>Group 2</b>      | -      | -      | +      | +      | +      | +      | +      | +      | +      | +      | -       |
| <b>Group 3</b>      | -      | -      | -      | +      | +      | +      | +      | +      | +      | +      | -       |
| <b>Group 4</b>      | -      | -      | -      | -      | -      | -      | -      | -      | +      | +      | +       |
| <b>Group 5</b>      | +      | +      | -      | -      | -      | +      | +      | -      | -      | +      | +       |
| <b>Group 6</b>      | +      | +      | +      | +      | -      | -      | -      | -      | -      | -      | +       |
| <b>Group 7</b>      | +      | +      | +      | +      | +      | +      | -      | -      | +      | +      | +       |
| <b>Group 8</b>      | +      | +      | +      | +      | +      | +      | +      | +      | -      | -      | +       |
| <b>Group 9</b>      | +      | +      | -      | -      | -      | -      | -      | -      | -      | -      | -       |

### 3.5 RESULTS SUMMARY

Table 9 summarizes the experimental cases that were run, showing the accuracy of each metric in sorting the teams according to the judges' assessments. Equation 7 defines accuracy.

$$\text{Accuracy} = \frac{TP + TN}{\sum g_i} \quad (7)$$

TP represents the True Positives found in the study, that is high quality teams as deemed by the judges is also identified by the metrics and TN represents the True Negatives, that is, low quality. The denominator represents the total number of teams being studied, in this study nine. As can be seen in both Table 9 and Figure 3, the standard deviation and coherence metrics turn out to be the best method for measuring the shared understanding of design teams.

TABLE 9. Accuracy of metrics

| Metric Summary | Case A | Case B | Case C | Case D | Case E | Case F | Case G | Case H | Case I | Case J |
|----------------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|
| Coherence      | 56%    | 56%    | 56%    | 56%    | 78%    | 78%    | 78%    | 78%    | 56%    | 56%    |
| Mean Dist.     | 22%    | 44%    | 22%    | 44%    | 67%    | 67%    | 56%    | 56%    | 44%    | 44%    |
| Max Dist.      | 33%    | 33%    | 44%    | 44%    | 67%    | 78%    | 67%    | 67%    | 44%    | 56%    |
| St. Dev Dist.  | 44%    | 22%    | 44%    | 44%    | 78%    | 78%    | 78%    | 78%    | 56%    | 56%    |

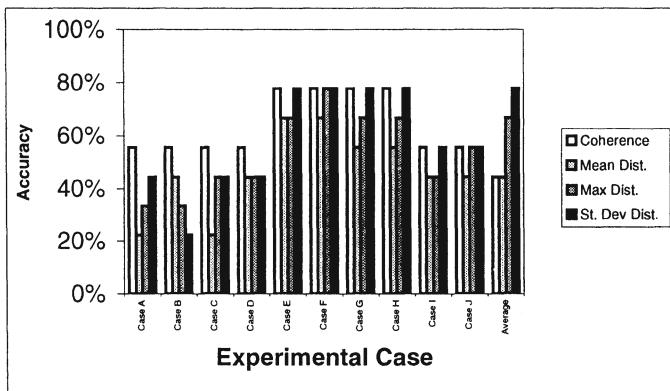


Figure 3. Summary of evaluation of metrics

As stated in Section 0, our premise is that teams with a high level of shared understanding will exhibit both design process quality and design outcome quality. Using our methodology to mine design documents for levels of shared understanding based on variance in communication and textual coherence, we were able to distinguish between teams with high and low shared understanding. We cross-correlated our quantitative measurements to judges' assessments of the teams. We found up to 80% agreement between our quantitative measures and the judges' assessments. Because we have shown that our method can find teams with high levels of shared understanding, and that these same teams had high quality outcomes based on impartial judges' assessments, then, based on the premise described in Figure 1, we have made progress towards demonstrating the validity of our methodology for measuring levels of shared understanding in design teams.

#### 4. Discussion and Implications

This paper addresses the question of identifying the level of shared understanding in design teams using the computational linguistic tool of latent semantic analysis. By computing the standard deviation (variance) of the s-dimensional radius of the singular vectors representing the design team's documents and the textual coherence of the documents, we were able to sort the design teams in order of level of shared understanding. We correlated our metrics of shared understanding to judges' assessments of the process quality and design quality of the design teams. We also found a correlation between textual coherence and team cohesiveness. Finally, we

found some quantitative support to suggest that a high level of shared understanding and cohesiveness among the design team leads to a better process and design quality outcome. As such, we conclude that, with about 80% agreement to expert human assessments in our study, our methodology can distinguish between levels of shared understanding in design teams based on the team's design documentation and that these levels correlate with increasing process and outcome quality.

The validation of our methodology provides exciting possibilities in future implementations. First, the experimental results from applying the metrics of variance and cohesion to the document show that retaining the 100 most significant singular vectors is sufficient to capture the context of the documents while minimizing the computational complexity. Also, the 100 most significant singular vectors provide the most accurate results. Future studies of design team communication using our methodology can use this guideline for retaining the singular values. Second, we were able to differentiate high and low performing teams with good correlation to expert assessments.

As a tool for reflective teams, they can use our methodology to reflect on their progress through the design process and identify potential problems in their performance in a predictive manner. Because most design teams use a document management or product data management system to help manage the design project, the technique can be run over the design documentation as each engineer contributes new documents. Each team can benchmark itself in real-time and use information on team cohesion to address deficiencies when appropriate. However, not all design documentation would be appropriate for this type of analysis. For example, documents from a finite element analysis of a component or structured data from a Taguchi quality test would not capture any meaningful dialog between the designers that would indicate their level of shared understanding. Minutes from design team meetings, studio reviews of the design, formal stage-gate reviews, product mission statements and other similar documentation which capture the thoughts and contain a "dialogicality of communication" between designers would be suitable candidates for this type of analysis.

This research offers significant advances in studying the communication of design teams to ascertain the level of shared understanding and cohesiveness. As a possible tool for reflection, design teams might apply these metrics to measure how well they compare respect to other design teams. In other words, they can use it as an indicator of the shared understanding and cohesiveness of their own design teams, taking steps as necessary to improve these psychosocial metrics as desired. For example, product development teams could use it for personal reflection and to

measure the level of commitment of their teammates to adopt appropriate strategies of persuasion. We believe that this could present a useful addition to computer-support collaborative work tools that support collaborative design teams.

While this research offers insight into shared understanding based on the final design documentation of the design teams, we believe that this area has much more potential for investigation with respect to communication in real-time, such as via e-mail and chat software. Further, by studying communication as it happens and through various stages of design, we can study the change in shared understanding to detect if there are patterns of shared understanding that lead to better design outcomes. For example, are design teams that reach shared understanding too quickly and never diverge less likely to produce novel designs? Finally, what is the implication of the mis-categorized teams? We postulate that while a high level of shared understanding and cohesiveness increases the likelihood of a high quality process and outcome, they are not guarantees.

The changing nature of teams from small, closely situated teams to larger, widely dispersed teams whose communication is nearly exclusively via information technology tools such as e-mail, places people in an information space that is not only too large for any single person to know completely, but that is also growing and changing at an increasing rate. Bringing tools to assist teams in gauging their communication may ultimately help them to improve team performance. We hope this line of research will have potential application to a range of enterprises that depend on effective teams for success. Finally, the method of latent semantic analysis shows great promise as a tool for modeling and evaluating the cognitive and psychosocial behavior of design teams.

### Acknowledgements

The authors wish to acknowledge the support of the faculty and the students of the new product development course at UC Berkeley in providing a test-bed for this research.

### References

- Adams, BN: 1967, Interaction theory and the social network, *Sociometry* 30(1): 64-78.
- Baird, F, Moore, CJ, and Jagodzinski, AP: 2000, An ethnographic study of engineering design teams at Rolls-Royce Aerospace, *Design Studies* 21(4): 333-355.
- Bødker, K and Pedersen, JS: 1991, Workplace cultures: looking at artifacts, symbols and practices, in J Greenbaum and M Kyng (eds), *Design at Work: Cooperative Design of Computer Systems*, Lawrence Erlbaum, Hillsdale, New Jersey, pp. 121-136.
- Bucciarelli, LL: 1994, *Designing Engineers*, MIT Press, Cambridge, MA.

- Citera, M, McNeese, MD, Brown, CE, Selvaraj, JA, Zaff, BS, and Whitaker, RD: 1995, Fitting information systems to collaborating design teams, *Journal of the American Society for Information Science* **46**(7): 551-559.
- Cook KS, and Whitmeyer, J: 1992, Two approaches to social structure: exchange theory and network analysis, *Annual Review of Sociology* **18**: 109-127.
- Cooper, RG, and Kleinschmidt, EJ: 1995, Benchmarking the firm's critical success factors in new product development, *Journal of Product Innovation Management* **12**(5): 374-391.
- Dhillon, IS, and Modha, DS: 1999, *Concept Decompositions for Large Sparse Text Data Using Clustering*, IBM Almaden Research Center, Research Report RJ 10147 (95022).
- Dorst, K: 1997, *Describing Design – A Comparison of Paradigms*, Doctoral Thesis, Delft Technical University, The Netherlands.
- Griffin, A: 1995, PDMA research on new product development practices: updating trends and benchmarking best practices, *The Journal of Product Innovation Management* **14**(6): 429-458.
- Fillmore, CJ: 1974, The future of semantics, *Berkeley Studies in Syntax and Semantics*, Department of Linguistics and Institute of Human Learning, University of California, Berkeley, CA, pp. 1-5.
- Gumperz, JJ, Kaltman, H, and O'Connor, MC: 1984, Cohesion in spoken and written discourse: ethnic style and the transition to literacy, in D. Tannen, (ed.), *Coherence in Spoken and Written Discourse*, ABLEX Publishing Corporation, Norwood, New Jersey, pp. 3-19.
- Halliday, MAK and Hasan, R: 1976, *Cohesion in English*, Longman, London.
- Harabagiu, S: 1999, From lexical cohesion to textual coherence: A data driven perspective, *International Journal of Pattern Recognition and Artificial Intelligence* **13**(2): 247-265.
- Henke, JW, Krachenberg, AR, and Lyons, TF: 1993, Cross-functional teams; good concept, poor implementation, *The Journal of Product Innovation Management* **10**(3): 216-229.
- Hill, A, Song, S, Dong, A and Agogino, AM: 2001, Identifying shared understanding in design using document analysis, *Proceedings of the 2001 ASME Design Engineering Technical Conferences*, DETC2001/DTM-21713.
- Landauer, TK: 1999, Latent semantic analysis: A theory of the psychology of language and mind, *Discourse Processes* **27**(3): 303-310.
- Landauer, TK, Foltz, PW and Laham, D: 1998, Introduction to latent semantic analysis, *Discourse Processes* **25**: 259-284.
- Lloyd, P: 2000, Storytelling and the development of discourse in the engineering design process, *Design Studies* **21**: 357-373.
- McDonough, III, EF, Kahn, KB and Barczak, G: 2000, An investigation of the use of global, virtual, and collocated new product development teams, *The Journal of Product Innovation Management* **18**(2): 110-120.
- Moody, J and White, D: 2000, *Social Cohesion and Embeddedness: A Hierarchical Conception of Social Groups*, Santa Fe Institute, Working Paper No. 00-08-049.
- Neck, CP and Manz, CC.: August 1994, From groupthink to teamthink: toward the creation of constructive thought patterns in self-managing work teams, *Human Relations* **47**(8): 929-952.
- Pahl, G and Beitz, W: 1996, *Engineering Design*, Second Edition, Springer-Verlag, London.
- Raccah, PY: July 1995, Argumentation and natural language - presentation and discussion of four foundational hypotheses, *Journal of Pragmatics* **24**(1-2): 1-15.

- Robinson, M and Bannon, L: 1991, Questioning representations, in L Bannon, M Robinson, and K Schmidt, (eds), *Proceedings of ECSCW'91: 2nd European Conference on Computer-Supported Collaborative Work*, Kluwer, Amsterdam, pp. 219-233.
- Schön, DA: 1983, *The Reflective Practitioner: How Professionals Think in Action*, Basic Books, New York.
- Simon, HA: 1996, *The Sciences of the Artificial*, Third Edition, MIT Press, Cambridge.
- Valkenburg, R: 2000, *The Reflective Practice in Product Design Teams*, Doctoral Thesis, Delft Technical University, The Netherlands.
- Wasserman, S and Galaskiewicz, J (eds): *Advances in Social Network Analysis: Research in the Social and Behavioral Sciences*, Sage Publications, Thousand Oaks, CA.

## AN EVOLUTIONARY APPROACH TO THE INVERSE PROBLEM IN RULE-BASED DESIGN REPRESENTATIONS

STEPHAN RUDOLPH AND ROLF ALBER  
*Universität Stuttgart*  
*Germany*

**Abstract.** In this paper design grammars as a rule-based representation formalism for design generation are discussed and used for the construction of transmission towers. In the context of such a rule-based representation technique several theoretical questions such as the completeness and capacity of a representation are still unanswered. Especially the question whether a given design object can be generated within a certain design grammar is still open and is known as the inverse problem. In the framework of the transmission tower design grammar an evolutionary approach for the solution of the inverse problem, i.e. the identification of a grammar that generates a predetermined pylon, is shown. The solution of the inverse problem is based on an evolutionary search algorithm whereby the fitness function is applied on the geometry of the design object and is therefore independent of the specific implementation of the design grammar. The design grammar search algorithm uses a combination of design space exploration techniques together with methods from pattern recognition and is shown in two simulation examples.

### 1. Introduction

Engineering design grammars are an alternative to other commonly used design representations. Rule-based concepts which stem originally from the area of artificial intelligence have already demonstrated their capabilities in several technical areas. Known applications can be found p.ex. in architecture for the design of shape patterns (Stiny 1977), in product-shape, function and cost estimates of coffee makers (Agarwal and Cagan 1997, 1998) as well as in mechanical design applications (Heiserman 2000). Studies of generic XML-based representations of knowledge enhanced engineering design grammars for a DAE (Differential Algebraic Equations) modelling approach have also been undertaken (Noser and Rudolph 2000).

However, in most cases the grammar representation has been designed for one special application making it hardly possible to use the same grammar compiler for any different domain. To overcome this disadvantage, a generic grammar compiler based on a graph-grammar formalism has been designed in order to offer a common core, applicable to different specialised engineering objects. The related process chain is illustrated in Figure 1.

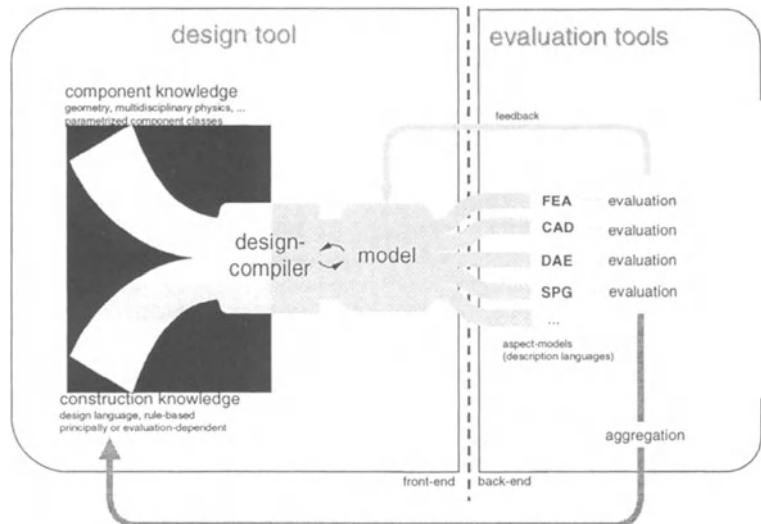


Figure 1. Process chain of a grammar compiler

The core of the grammar compiler in Figure 1 consists of an interpreter which is responsible for the construction of the desired physical model. The information needed to carry out this task is stored in two parts. The first part contains the construction knowledge for which graph grammars serve as a general syntax to express how different parts can be assembled and thererby defines the topology of the design. The second part incorporates the information needed to translate between the abstract symbolic information level that is applied in the grammar rules and a specific modeling description applicable for further post-processing (i.e. in CAD- and FEM-programs).

The rule-based design representation for the grammar compiler in Figure 1 is inteded to work as a front-end for common design analysis tools in order to allow the incorporation of a manifold of different applications in the design process. This is carried out by returning the analysis results back into the grammar compiler and thereby beeing completely accessible for the rule-based part of the design process.

The first part of this paper shows the working principle behind rule-based design grammars for the example of biological growth modeling and makes proposals for a systematic transfer of this approach into technical domains.

The second part connects the proposed representation formalism with a common evolutionary search technique and together with means from the area of pattern recognition proposes an approach for the inverse problem.

## **2. Design Grammars as a Language for Formal Design Representation**

The grammar-based design concept proposed here consists of two phases. The first part takes place in a very formal and abstract domain and works simply with symbolic placeholders far away from the real-world design entities. This part plays the role of evolving a well formed topological structure by arranging a set of symbols through the subsequent application of production rules. The second step inaugurates this topology to a “real-world” design by interpretation of the symbols as material entities which properly connected form the design object.

The principle of this approach is illustrated by a string grammar called Lindenmayer-Systems (or L-systems for short) which was developed for the description of biological plant and tree growth. L-systems turned out to be a compact and powerful means for the representation of hundreds of different complex natural objects (Lindenmayer and Prusinkiewicz 1996).

### **2.1 BASIC CONCEPT**

The main idea behind this concept is to consider the creation of natural structures as the result of a “developmental program” which could be understood as a recursive process (Lindenmayer 1968). In nature there are no blue-prints nor technical sketches, instead rule-based systems (encoded in chromosomes) define how cells develop (by transcription and translation of genes) in order to finally form a complete and functioning organism.

L-systems represent a special case of the grammar formalisms which were developed and classified within the scope of formal language theory (Chomsky 1957), in order to describe the structure of a string as a systematic application of substitution rules.

#### *2.1.1 Rule-based structuring*

In L-systems as well as in Chomsky grammars the abstract symbols are represented by characters from an alphabet which manifest a topological structure through their arrangement patterns in strings.

The basic concept for structuring this set of symbols (here the characters from the alphabet) relies on the principle of substitution. In the case of strings the grammars are therefore also called rewriting systems. In general the substitution technique generates complex strings by recursive replacement of parts from a simple initial object, the so-called axiom. This replacement is controlled by a set of substitution or production rules which together with the axiom form a grammar.

A substitution rule consists of an initiator on the left side and a generator on the right side as illustrated in Figure 2. If the initiator is identified anywhere in the object at the current stage of recursion, the corresponding rule is applied by replacing this part of the object by the part defined in the generator. As L-systems are character-based, the initiator as well as the generator consist of characters or strings out of a predefined set, the so-called *alphabet*. More rudimentary operations like addition or subtraction can also be expressed within a substitution rule.

The example illustrated in Figure 3 shows how a complex structurized string is unfolded from a single character (i.e. the axiom) within only three iterations by simple application of two substitution rules. It shall be noted that at this point none of the utilized characters have special functions concerning the application of rules.

Initiator → Generator

*Figure 2.* Syntax of a production rule

|             |          |
|-------------|----------|
| B           | (Axiom)  |
| B → F[-B]+B | (Rule 1) |
| F → FF      | (Rule 2) |

recursion    expanded string

- 0:    B
- 1:    F[-B]+B
- 2:    FF[-F[-B]+B]+B+F[-B]+B
- 3:    FFFF[-FF[-F[-B]+B]+F[-B]+B]+FF[F[B]+B]+F[-B]+B

*Figure 3.* Expansion of a string by recursive application of rewriting rules

### 2.1.2 From strings to plants

Since classical L-systems were introduced to represent biological growth processes, a physical interpretation of the topological structure encoded in the abstract character set is necessary to obtain botanic objects. For this purpose, the concept of "turtle-graphics" (Abelson and diSessa 1984) was adopted and modified to translate the single characters in the expanded string as material components and construction operations. Table 1 shows

such a possible mapping between the characters of the alphabet and the material components.

TABLE 1.Character assignment of material components and growth operations

| Character | Interpretation                                                        |
|-----------|-----------------------------------------------------------------------|
| F         | add stalk segment                                                     |
| -         | rotate growth direction by a discrete angle $\delta$ counterclockwise |
| +         | rotate growth direction by a discrete angle $\delta$ clockwise        |
| [         | introduce stalk segment with branching                                |
| ]         | jumps back to the last branching point                                |

In the following examples the character-interpretation given in Table 1 was applied. Characters which are not contained in the table are not interpreted and may simply serve for the initialization of a substitution rule.

Figure 4 shows several L-system grammars together with their corresponding two dimensional geometric interpretation. In the shown Figures the notation  $n$  is the necessary recursion depth during the expansion phase and  $\delta$  denotes the angle applied for the interpretation in Table 1.

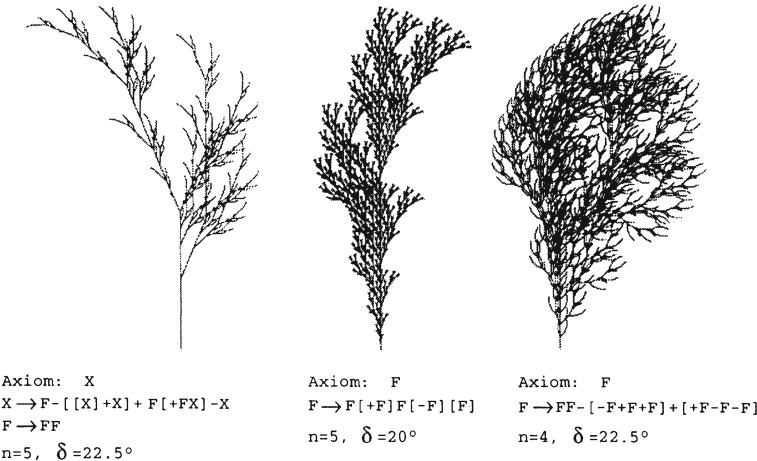


Figure 4. Different L-system grammars with their corresponding material interpretation (Lindenmayer and Prusinkiewicz, 1996)

The remarkable compactness of the design representation which can be seen by comparing the complexity exhibited by the generated object with the amount of rules necessary for the description is one of the most impressive features of classical L-systems.

### 3. Grammars for Technical Objects

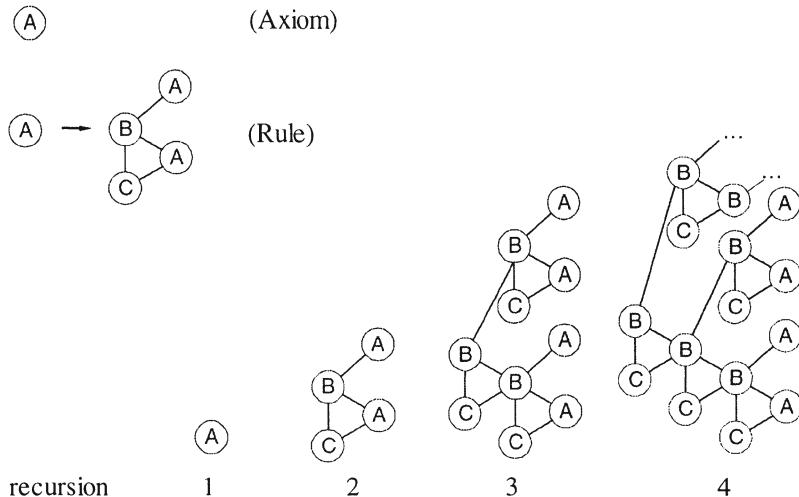
#### 3.1 PROPOSED MODIFICATIONS FOR TECHNICAL GRAMMARS

For an application in technical domains the design grammar formalism had to undergo the following modifications:

- In order to facilitate the isomorphism between the symbol arrangement on the abstract layer and the topology shown in the desired real world objects, it is attempted to work with an abstract structural representation that is still accessible with a rule-based approach but needs less complicated interpretations later. This can be achieved by formatting the dependencies on the abstract symbol layer in undirected graphs instead of strings
- The corresponding formalism for a rule-based construction of such a graph is known as graph grammar and commonly used in information processing and compiler optimization (Ehrig and Kreowski 1990). The analogy to string grammars with their substitution rules presented in the last section is straightforward. The initial state denoted by the axiom which was a string in the former case is now represented as an initial graph.

The production rules still resemble the initiator-generator form of the rewriting rules in Figure 2, with the simple difference that now the initiator as well as the generator are represented by graphs. The substitution procedure is again applied by finding the initiator-graph as subgraph in the current object and replacing it by the graph defined in the generator. A more detailed examination will show that for complex rules the substitution procedure needs an additional description which is called “embedding-relation”. This however lies outside the scope of this paper and can be found i.e. in Göttler (1988).

- Figure 5 presents an example for the syntax of a graph grammar and its application and can be seen as a direct analogy to the string grammar formalism illustrated in Figure 3.
- In the classical approach the structuring grammar application and the following interpretation were described as two clearly separated subsequent steps. However, this distinct partition is questionable for technical applications and definitely holds not true for real world biological systems.



*Figure 5.* Evolution of a graph by recursive application of a graph grammar

- As the development of plants is not uniquely determined only by the information encoded in the genes but also highly depends on environmental influences, it is clear that the current approach neglects an important developmental feature. To resemble this circumstances, it is proposed to break up with the paradigm of a separated 2-step grammatical procedure. This is done by introduction of a so called “feedback loop”, see Figure 1.
- The idea is to have the abstract layer, that is the structured symbol set, as well as the evolving product in memory at the same time. This can be achieved by performing the interpretation step immediately after each rule application. Furthermore, the evolving object can undergo a direct evaluation after each step providing the system with additional in-time information about certain aspects of the “real-world” design object at the current stage.
- The simultaneous existence of abstract design topology as well as real world materialized components gives furthermore the possibility to provide the production rules with additional functionality. It is therefore proposed to extend the conditional part of the rule syntax (denoted by the initiator) with additional constraints concerning the materialized model and the results derived so far from an evaluation. This means that a rule

is only applied when the initiator is identified in the abstract layer *and* the additional constraints are not violated in the materialized object.

In the following, the construction of an engineering design grammar will be shown as an example for the proposed graph-grammar design approach.

#### 4. A Design Grammar for Truss-Like Structures

As an example for the grammar design proposed in the last paragraph an engineering design grammar for truss-like structures as they appear for example in pylons, in civil engineering domains or in space-structures has been developed.

As transmission towers show at least a certain amount of symmetry, self-similarity and repetition, it is expected that this technical object serves as a good example for how a compact design representation can be achieved by means of a rule-based approach. Furthermore grammars for truss structures have also been addressed in different approaches (Shea and Cagan 1998)

##### 4.1 BASIC CONCEPT

Following the approach proposed in the last paragraph, basic building blocks for the most common truss-like structures have been identified in order to be replaced by abstract symbols for an easy-to-use rule-based implementation. Figure 6 shows such a primitive where the parameterization effects the aspects height, length ratios, twist angles as well as the rod topology inside the polyeder. In this way a high variety of primitives can be inherited from the same building block and therefore a high amount of self similarity and repetition is achieved as will be illustrated in later examples.

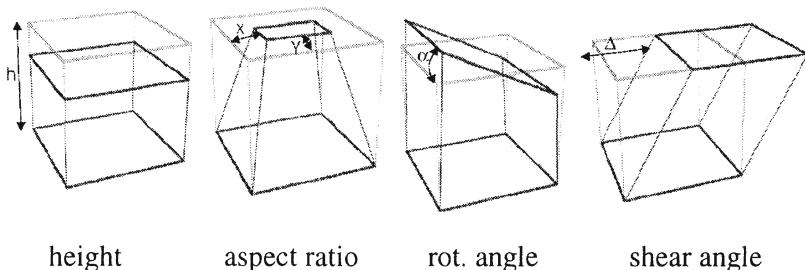


Figure 6. Parameterization effects

By defining such polyeders most of the common truss-structures can be represented by a graph that describes how these components are instantiated with specific parameters and connected to each other. The combination

between different primitives is done through the side faces of the polyeder which serve as interfaces. For geometrical consistency, it is necessary that the geometry of a subsequently added primitive is adopted such that the corresponding interfaces match each other. Therefore the geometric parameters are always defined relative to the corresponding parameters of the parent primitive.

Figure 7 shows in an example how a truss-structure can be assembled from three connected geometrical primitives. The shaded areas indicate the interfaces where the connection of components occurs.

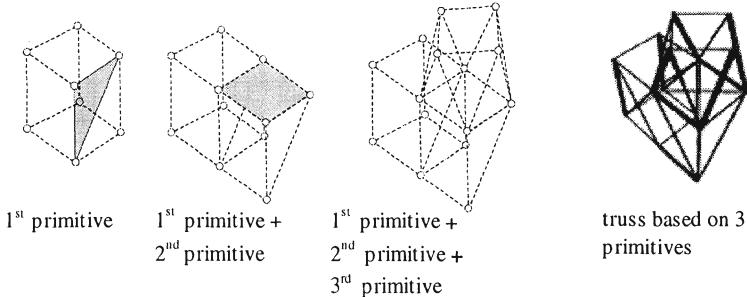


Figure 7. Truss based on three connected geometrical primitives

#### 4.2 IMPLEMENTATION

The graph grammar implementation which has been chosen for the truss grammar operates on an abstract node set, whereby each node has a label assigned for a unique identification.

After each rule application an immediate interpretation step follows which corresponds to the assignment of a geometric primitive to the nodes. Thereby the graph-relations define the connection between the single primitives. Furthermore an evaluation step is performed after each rule application. For our purpose an evaluation of basic geometric properties is sufficient. The geometric information (like position, size, angles) derived in this step can be interrogated by additional constraints in the rule.

#### 4.3 EXAMPLE

In the following an example for the truss grammar is presented. Figure 8 shows a set of 5 rules which describe the building principles behind the desired truss. As a new feature wildcards have been introduced to the rule syntax which are denoted by the symbol “\*”. In the rule-set introduced here, two additional constraints occur, interrogating the height above ground (Z) and the size (R) of the primitives’ connection interface. The interpretation of the derived symbol graph is done with the assignment

table given in Figure 9. The 6 shown primitives were all derived from the same underlying polyeder with different geometric and typological parameters. The shaded faces indicate possible connection interfaces.

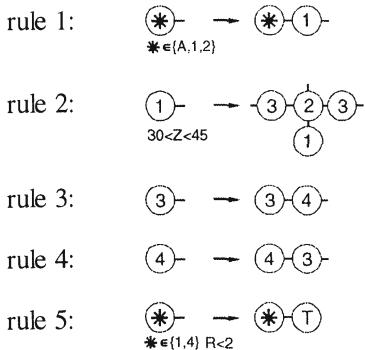


Figure 8. Rule set for the topology of a transmission tower

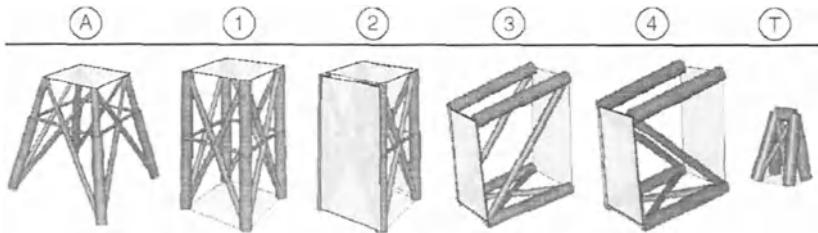


Figure 9. Interpretation table for assignment of material components to symbols

Figure 10 shows snapshots of the stepwise construction of a transmission tower following the rules of our grammar. On the left side the actual graph which is built from the grammar rules is shown, whereas the right side shows the corresponding object derived through interpretation.

Comparing the complexity of the derived object with the information needed for its construction shows a high packing density for this design grammar similar to that of the classical L-systems from which they were initially inspired.

## 5. Exploration of Design Space

The goal which was aspired with the grammar approach was to capture large portions of the design space within one single formalism and especially to increase the design freedom, such that the automatic generation of new

designs becomes possible. By manual formulation of grammar rules it was possible to find a truss-language description for several “real-world” objects. Examples are illustrated in Figure 11 (Alber 2001).

step 0

(A)

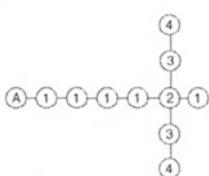


step 4

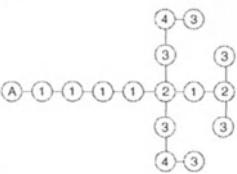
(A-1-1-1-1)



step 6



step 8



step 19

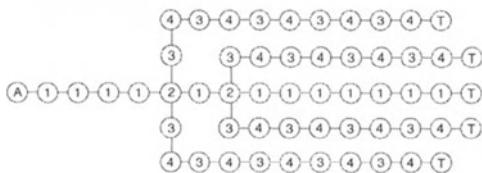
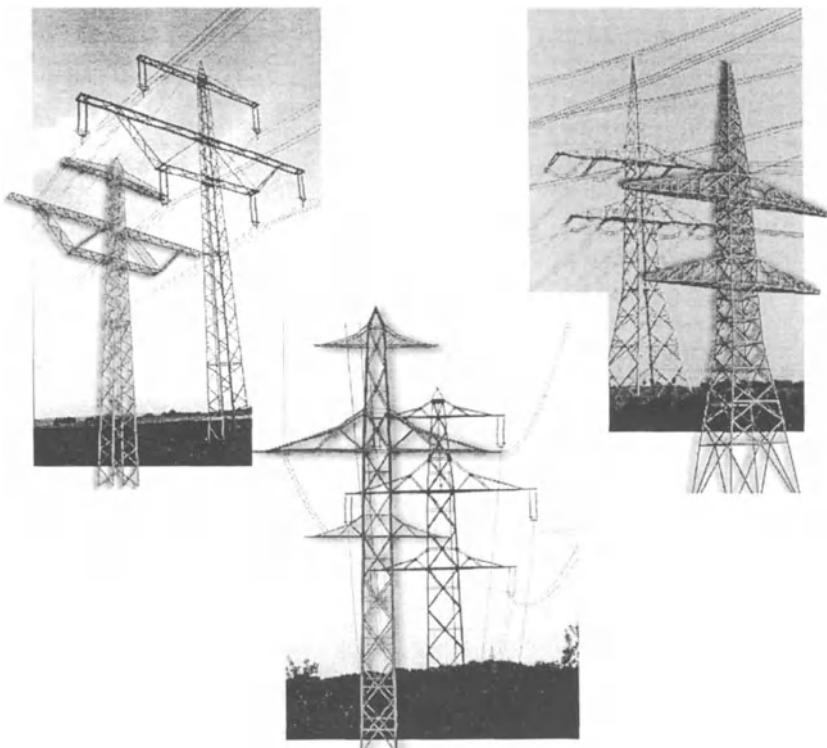


Figure 10. Rule based evolution of a transmission tower



*Figure 11.* "Real-world" trusses translated into the truss-grammar language

Each of the shown transmission towers was generated by a certain design grammar within the truss grammar formalism and thus corresponds to a point in the captured part of design space. In the following a method from the domain of evolutionary algorithms shall be adopted to work with a rule-based design representation in order to provide a means for the systematic and automated exploration of design space.

### 5.1 EVOLUTIONARY DESIGN SPACE EXPLORATION

Evolutionary algorithms perform a direct stochastic search by imitating the principle of biological evolution. Instead of a focused optimization on one single object, the collective development of a whole population is undertaken - a principle that has successfully been applied in nature for millions of years.

In order to allow a variation of the information which is presented in the genotype, operations are defined that perform stochastic modifications in an analogy to nature's ability to mutate and recombine genes in organic cells. In that way the evolutionary process can breed in each generation a certain amount of new individuals. By means of a mixed deterministic and stochastic selection strategy it is expected that after a certain number of evolution- cycles only those genotypes survive which represent an optimal solution for a predefined objective.

## 5.2 GENETIC GRAMMAR PROGRAMMING

In the scope of evolutionary algorithms Koza (1992) introduced the genetic programming in order to simulate an evolutionary development for computer programs. Similar to common computer languages, a rule-based grammar can be regarded as a programming language which is interpreted by a grammar compiler. Therefore a genetic grammar programming environment has been developed to simulate an evolutionary cycle for the truss-grammar.

In order to allow a genotype evolution, operators have been defined that are able to perform randomly controlled modifications in a rule-based representation. To allow a complete evolutionary search, these operators should be able to modify all features on which the phenotype of a grammar is dependent.

For our truss-grammar these features are:

- the maximum number of possible iterations
- the axiom
- the rules with initiator-part, generator-part and constraints
- the order of the rules

Besides the ability to perform modifications in all of the mentioned features, it is important to guarantee that the syntactical correctness of the grammar is never violated. This objective restricts single parameter values to a predefined range and is especially difficult to achieve if there exist dependencies between different parameters which represent additional constraints for modifications. For this reason a template for each type of rule has been created using object oriented software techniques, which exactly define in which way an item of a rule may be modified by evolution.

For the genetic grammar programing the following operators have been implemented:

- *rule mutation*: performs changes in the initiator and generator graphs of a rule like addition or deletion of certain nodes or connections.

Furthermore the addition or modification of constraints in the initiator are possible.

- *permutation*: changes the order in the set of rules
- *deletion*: removes one rule
- *addition*: a new rule is randomly generated and added to the grammar
- *mutation of iteration depth*: alters the number of iterations
- *mating / crossover*: exchanges rules with another randomly selected grammar.

To provide a set of starting points for the evolutionary search a grammar generator is needed which randomly creates a set of rules and combines them in a syntactically correct grammar.

### 5.3 EVOLUTION STRATEGY

In the scope of evolutionary algorithms several strategies have been introduced in order to perform a successful search. By adequately choosing population size, reproduction type and the described kind of selection the evolution cycle simulated with the truss-grammar was implemented in a common  $(\mu, \lambda)$  -strategy (Rechenberg 1973). The corresponding flowchart is shown in Figure 12.

In a first step, a population consisting of  $\mu$  individuals is randomly generated. Every member of this initial population represents a parent. In every generation each of the parent individuals breeds  $\lambda$  descendants who after an evaluation get assigned an individual fitness corresponding to how good their present features match the desired objective. This is done by means of a carefully chosen fitness function. The parent and the descendants form a family with  $(1+\lambda)$  members from which one single survivor is determined in a selection process. The selected individual represents the new parent for the next generation in the evolutionary cycle.

In the following it is described how the proposed genetic grammar programming together with a well designed fitness function can be used as an approach for the inverse problem.

## 6. An Evolutionary Approach to the Inverse Problem

A design grammar and it's corresponding compiled object represent a genotype-phenotype pair in analogy to the biological terminology. The set consisting of all possible grammars defines a genotype-space which is closely linked with a corresponding phenotype set as shown in Figure 13. While it is straight forward from a computational viewpoint to generate an

almost arbitrary amount of objects from a known grammar type, it is still an open scientific question whether a grammar can be generated for a single known object or for a set of given objects.

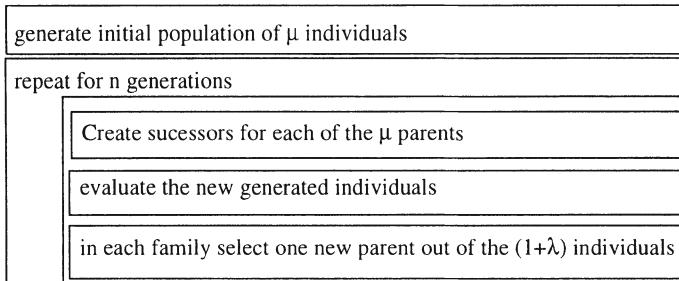


Figure 12. Flowchart of an evolutionary algorithm with a  $(\mu, \lambda)$ -strategy

This question which is known as the *inverse problem* is illustrated in Figure 14. The parsing of design objects into grammar representations has been undertaken for simpler design representations but shown to be a complicated and computationally highly intensive task (Mackenzie 1989). The still missing solution to this question for more complex rule-based concepts represents a general drawback for current grammar concepts.

It is suggested that an evolutionary approach for design grammars as described in section 2 in combination with an appropriate fitness function can be used to create a search algorithm for the solution of the inverse problem. This has been investigated for the example of the truss-grammar and has been confirmed in the framework of this study by numerical simulations.

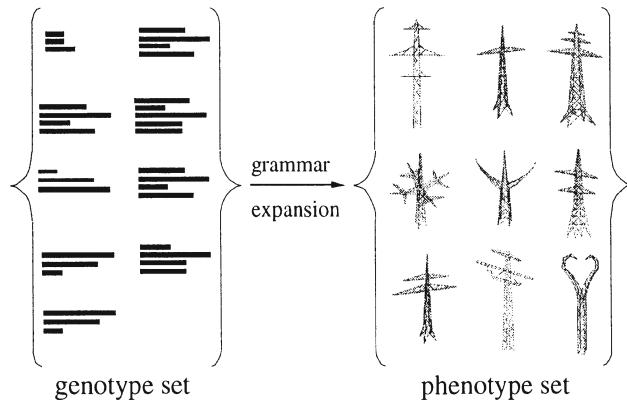


Figure 13. Genotype-space and corresponding phenotype-space

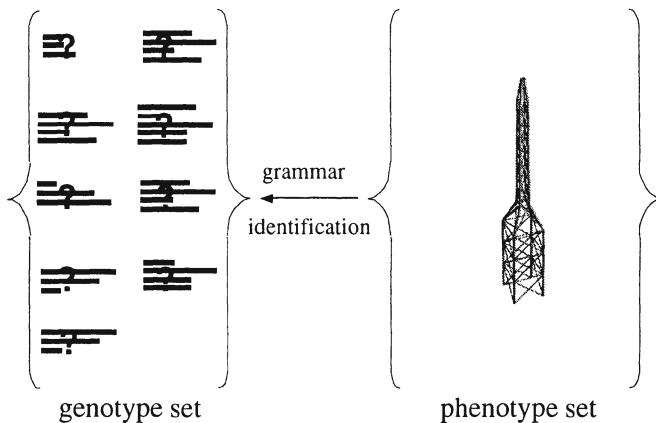


Figure 14. The inverse problem

## 6.1 FORMULATION OF AN APPROPRIATE FITNESS FUNCTION

The formulation of the fitness function mainly determines the goal which is aspired by an evolutionary algorithm and therefore has to be carefully designed. For a solution of the inverse problem, this goal is to find a rule-based description which defines an Object that is as close as possible to a predefined phenotype.

For the fitness function a quantitative scalar value is needed in order to judge how good an individual matches this objective. In our case the geometrical shape is relevant for evaluating the similarity of two phenotypes. This must however not always be the case. In this work known methods from image-processing and pattern recognition are extended for three dimensional objects and used in the design of our fitness function.

### 6.1.1 Evaluation of geometrical similarity by means of higher statistical moments

A first step for the detection of similarity lies in the abstraction of the examined object. From the field of image processing stems the approach to extract specific scalar features from the color information which is distributed over a set of image pixels (points). For the objects generated with the truss grammar this distributed information is represented by the density collocation  $p(x,y,z)$ . For a point  $(x,y,z)$  in space this function returns the corresponding density value if it belongs to a material part of the object otherwise 0.

An integral transformation applied to the density collocation is used for the extraction of scalar feature values in equation 1. The distributed information is thereby weighted with the local point-coordinates, which makes this transformation equivalent to the calculation of higher statistical moments.

$$m_{pqr} = \iiint x^p \cdot y^q \cdot z^r \cdot \rho(x, y, z) dx dy dz \quad (1)$$

Here  $p, q$  and  $r$  denote the moment order in x-, y- and z- direction. The retrieved features represent integral information about symmetry and density distribution of the examined object. A further step of abstraction is to gain independence of the examined object's overall dimensions. This is done by transforming the moments  $m_{pqr}$  to a dimensionless group  $\mu_{pqr}$  (Emrich 1998), whereby the relationship has proven to be a reasonable approach. In equation 2  $h_{max}$ ,  $w_{max}$  and  $d_{max}$  denote the maximum spatial dimensions in the X-, Y- and Z-direction.

$$\mu_{pqr} = \frac{m_{pqr}}{h_{max}^p \cdot w_{max}^q \cdot d_{max}^r \cdot m_{000}} \quad (2)$$

By calculation of a set of higher moments  $m_{pqr}$  res.  $\mu_{pqr}$  of order  $p, q, r = 0 \dots (p, q, r)_{max}$  the examined object is assigned to a point in feature space.

$$\vec{\mu}_i = [\mu_{100}, \mu_{010}, \mu_{001}, \mu_{110}, \mu_{101}, \mu_{011}, \mu_{200}, \dots \\ \dots \mu_{p_{max}, q_{max}, r_{max}}]_i \quad (3)$$

The higher the maximum order  $(p, q, r)_{max}$  is chosen, the more precise is the representation of the original geometric information by the calculated features. This can be proven by the computation of the backtransformation of equation 1 (Melan and Rudolph 2000).

$$d = \left( \sum_{p=0}^k \sum_{q=0}^k \sum_{r=0}^k (\mu_{Ref, pqr} - \mu_{i, pqr})^2 \right)^{\frac{1}{2}} \quad (4)$$

As a result of the abstraction it is expected that similar objects are projected on points in close vicinity in feature space. Thus the distance of these points can serve as a numerical measure for the similarity of the corresponding objects. Within the scope of this investigation the euclidean distance  $d$  between a reference point  $\mu_{Ref, pqr}$  and the examined object will be used as fitness function:

In equation 4  $k$  denotes the maximum order for all three dimensions. The reference point, in our case corresponds to the aspired target object for which the inverse problem shall be solved.

## 7. Simulations

For a first assessment of sensible parameters for the evolutionary cycle, simulation runs were performed where the goal was not yet the exact reproduction of a predefined truss. Instead the objective was to reach the three endpoints illustrated in Figure 15 with the truss's tips.

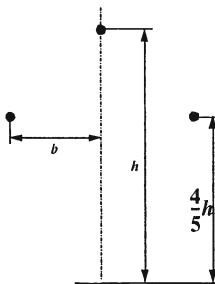


Figure 15. Objective for the position of the truss's tips

In this way the computationally intensive transformation in equation 1 can be replaced by the straightforward calculation of the sum

$$m_{pqr} = \sum_{i=1}^n x_i^p \cdot y_i^q \cdot z_i^r$$

where  $n$  now denotes the number of tips present in the examined object. Furthermore this objective still remains us a certain design freedom for the shape of the truss which in several simulations resulted in different new truss structures, all fulfilling the prescribed condition.

Figures 16 and 17 show the evolutionary development of two grammar descriptions with a ( $\mu=20$ ,  $\lambda=5$ )-strategy over 200 resp. 300 generations. The maximum order for the higher moments was set to  $k=10$ .

By comparing the results with the objective it is clearly visible that a directed evolution has been achieved with the fitness-function proposed here.

Motivated by the results obtained, extended simulation runs have been made, now with the goal of reproducing a complete predefined phenotype as illustrated in Figure 18.

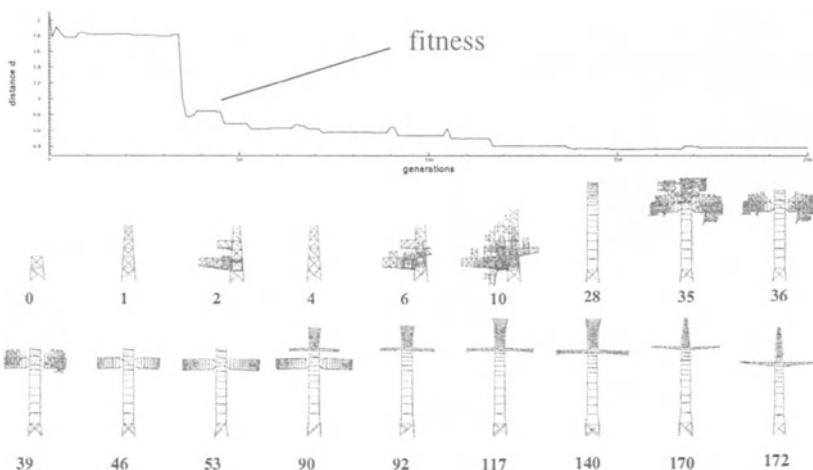


Figure 16. Evolutionary development of a grammatically represented

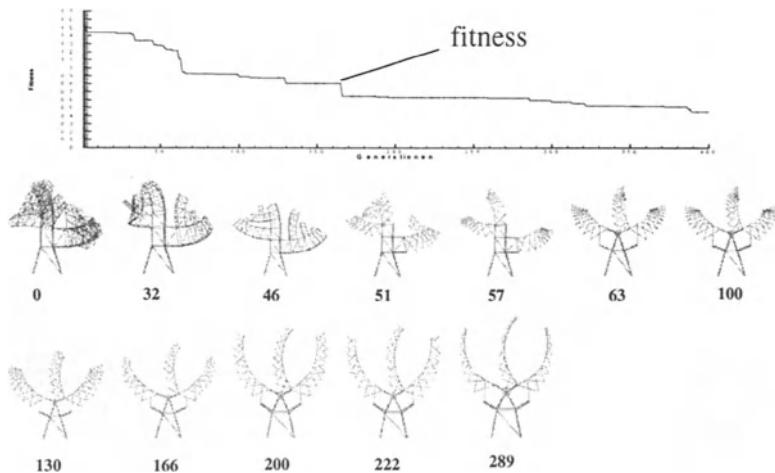


Figure 17. Evolutionary development of a grammatically represented

In order to again reduce computation time the maximum order for the higher moments was reduced to  $k=6$ . shown in Figure 19.

Again a clearly directed evolution towards the solution was observed. The grammar obtained after 148 generations was able to represent the pylon he proposed evolutionary approach to the inverse problem is not limited to the truss grammar presented here and could also be applied to other

grammars. Thereby the implementation of the stochastic grammar modification and generation is generally possible but needs additional effort for validating that only syntactically correct grammars can be evolved.

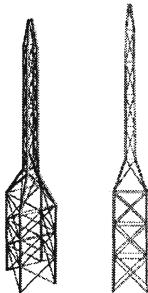


Figure 18. Objective for the inverse

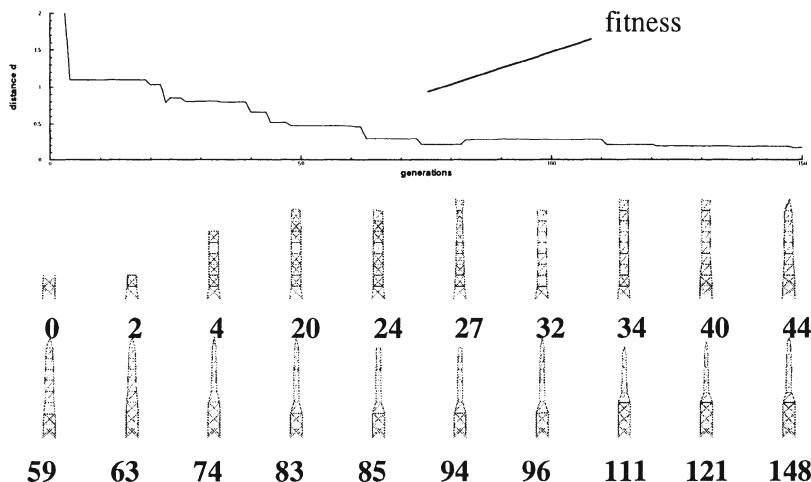


Figure 19. Evolutionary development of a grammatically represented truss

## 8. Summary

The possibility to describe complex design topologies with grammars was shown for a well known example from biology. Based on the promising results exhibited by this technique a procedure for design of a graph-grammar based representation language for technical structures was proposed and shown for the example of a truss grammar.

The engineering design grammar developed for truss-structures showed that a compact design representation for certain technical objects can be developed using rule-based techniques in a relatively straightforward way.

The examination of the design space resulting from this approach showed that a lot of different designs could be captured by one consistent formalism.

For a systematic exploration of design space an evolutionary approach was adopted as genetic grammar programming technique and implemented for the mentioned example.

In combination with evaluation methods derived from the field of pattern recognition and similarity theory, a novel conceptual approach for the solution of the inverse problem was presented. The feasibility of the approach was confirmed in numerous simulations of which two have been described in some detail.

## References

- Abelson, H. and diSessa, A: 1984, *Turtle Geometry*, The MIT Press, Cambridge, MA.
- Agarwal, M, and Cagan, J: 1997, Shape grammars and their languages - a methodology for product design and product representation, *ASME Design Engineering Technical Conferences 1997*, ASME, DETC97/DTM-3867.
- Alber, R: 2001, *Synthese und Evolution einer technischen Entwurfsgrammatik nach dem Vorbild biologischer Wachstums- und Entwicklungsprinzipien*, Diplomarbeit, Universität Stuttgart.
- Cagan, J and Agarwal, M: 1998, A blend of different tastes: the language of coffee-makers, *Environment and Planning B: Planning and Design* **25**: 205- 22.
- Chomsky, N: 1957, *Syntactic Structures*, Mouton, The Hague.
- Ehrig, H and Kreowski, H-J: 1990, *Graph Grammars and Their Application to Computer Science*, Springer-Verlag, Berlin.
- Göttler, H: 1988, *Graphgrammatiken in der Softwaretechnik*, Springer-Verlag, Berlin.
- Heisserman, J: 2000, A design representation to support automated design generation, in JS Gero (ed.), *Artificial Intelligence in Design'00*, Kluwer, Dordrecht, pp. 545-566
- Kaandorp, J: 1994, *Fractal Modelling, Growth and Form in Biology*, Springer Verlag, Berlin.
- Koza, J: 1999, *Genetic Programming*, MIT Press, Cambridge, MA.
- Lindenmayer, A and Prusinkiewicz, P: 1996, *The Algorithmic Beauty of Plants*, Springer-Verlag, Berlin.
- Lindenmayer, A: 1968, Mathematical models for cellular interaction in development, Part 1 and 2, *Journal of Theoretical Biology* **18**: 280-315
- Mackenzie, CA: 1989, Inferring relational design grammars, *Environment and Planning B* **16**(3): 253-287.
- Melan, A and Rudolph, S: 2000, An analytical approach to classification by object reconstruction from features, *Proceedings SPIE Aerosense 2000 Conference On Signal Processing, Sensor Fusion and Target Recognition IX*, SPIE, Washington, pp. 102-110.

- Rudolph, S and Noser, H: 2000, On engineering design generation with XML- based knowledge enhanced grammars, *Proceedings IFIP WG5.2 Workshop on Knowledge Intensive CAD (KIC-4)*, Kluwer, Dordrecht, pp. 227-237
- Shea, K and Cagan, J: 1998, Generating structural essays from languages of discrete structures, in JS Gero and F Sudweeks (eds), *Artificial Intelligence in Design'98*, Kluwer, Dordrecht, pp. 365-384.
- Stiny, G: 1977, Ice-ray: a note on the generation of Chinese lattice designs, *Environment and Planning B* **4**: 89-98.
- Rechenberg, I: 1973, *Evolutionsstrategie*, Fromman-Holzboog, Stuttgart.

## **EVOLVING THREE-DIMENSIONAL ARCHITECTURE FORM**

*An application to low-energy design*

LUISA CALDAS  
*Instituto Superior Técnico  
Portugal*

**Abstract.** This paper presents the application of an evolution-based generative design system to the creation of three-dimensional architectural form, guided by goals of daylighting use and energy conservation in architecture. The generative system is based on a genetic algorithm and a building simulation software, DOE2. From an initial schematic design and a set of rules and constraints that encode the architect's intentions, the system searches the solution space for design configurations that respond to the initial targets. The first part of the paper deals with methodological questions of encoding schematic layouts and accounting for the emergence of new design features. The second part presents results using energy use intensity as the objective function, and researches the use of penalty functions in this context.

### **1. Introduction**

A method is proposed to incorporate evolutionary mechanisms into the generation of three-dimensional architectural form. Adaptation is directed towards environmental behavior, looking for shapes that best harvest daylighting and reduce thermal exchanges with the external environment.

The paper starts by addressing a number of representation issues. In order to enable a building description to be manipulated as raw data by a generative design system, it is necessary to define a framework to encode architectural form through the use of symbolic languages, design variables and data structures (Mitchell 1977). These abstract representations include not only geometric and topological information, but also other architecture-specific information layers, derived from general characteristics common to most buildings: an architecture piece stands in a physical site, has a certain orientation, is subject to a given climate, is built with specific construction materials, contains mechanical and electrical equipment, and is occupied by people. To perform the conceptual transition from pure form to actual

building designs, these factors must necessarily be included in a building description.

Another relevant representation issue is the need to conceptually define schematic design layouts that govern the overall evolution of solutions. Within a generative system (GS) like the one presented here, the architect must describe a design solution in terms of abstracts relationships between different elements, variables and their constraints, compositional rules, etc., instead of completely specifying a design in a deterministic way. The GS then uses these initial rules as the basis for generation of new solutions, and exact configurations for each element will only emerge at the end of the evolution process.

As for the incorporation of evaluation methods, the paradigm of goal-oriented design (Monks 1998) is adopted. The architect sets performance targets for the design to achieve, and the GS makes use of a search mechanism to look for solutions that approach the desired objectives, by manipulating the 3D geometry of the building, its space layout and its façade design. Design targets used in this paper were: maximizing the use of daylighting; reducing the subsequent use of artificial lighting; and reducing energy consumption for heating and cooling the building throughout the year. All these goals are translated into the annual energy consumption of the building, which is passed to the genetic algorithm as the objective function value.

In this GS, the search mechanism used is a Genetic Algorithm, and evaluation is performed by a thermal and lighting simulation software, DOE2, that assesses the performance of solutions by doing dynamic, hourly building simulations, and assigns fitness values to each individual. The concepts and functioning of this Generative System have been described elsewhere (Caldas and Norford 2000; Caldas and Rocha 2001), even though applied to façade design and not yet to 3D shape manipulation.

In this paper, a series of different experiments are described. The first one departs from a fixed space layout, and only surface tilts and façade openings can be manipulated. In the subsequent experiments, building geometry and space layout are variables themselves, and different methods are used to incorporate area assignments into the calculation of fitness values for different solutions.

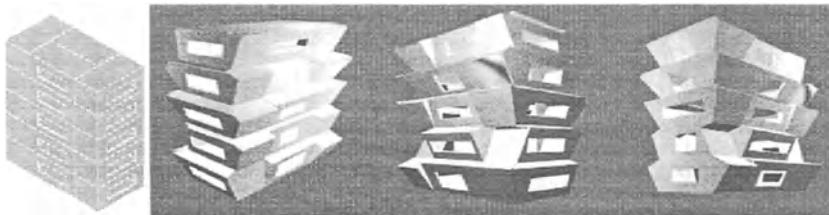
## 2. Varying External Wall Tilts

In the first experiment, the overall shape of the building was fixed, but the GS was allowed to play with the tilt of exterior wall surfaces. The initial, schematic building shape from which the GS departed can be seen to the left of Figure 1, while results are shown to the right of the same Figure.

The building has five identical floors, with the narrower sides facing south/north, and the longer sides east/west. Each floor has four rooms, with identical dimensions. There is just one window per room, in the widest wall.

Parameters varying in these experiments were: wall tilts, which could vary from 60° to 140° (a vertical wall having a 90° tilt, a smaller value representing a tilt inwards, and a larger value a tilt outwards), and window dimensions (width and height). The location chosen for the experiments was Oporto, Portugal, representing a mild climate.

Results are shown in Figure 1. The generated building shapes are formally complex and interesting, considering the simple initial shape from where they departed. In general, the walls tended to tilt away from the sun, especially for east and west facades. For these orientations, window size generally increases with wall tilt, as the less exposed windows reduce heat gain into the building while admitting useful natural light. North and south-facing walls tend to be less tilted. North windows are quite large, as they suffer no direct sun penetration. Towards the south, windows in walls that remained vertical are smaller than those carved in inclined walls. Towards the top floors, where more tilted surfaces appear, there are quite large openings.



*Figure 1.* Initial building shape (left) and results for Oporto (from left to right, northwest, northeast and southeast views of the same building)

These results suggested that from a simple initial layout, and through the application of simple rules, or parametric deformations, it was possible to generate shapes with a high degree of complexity, that would be hard to predict from the outset and that were trying to use the simple mechanisms at their disposal to enhance the building's adaptation to its environment.

### 3. Parametric Shape Manipulation

Finding simplified initial layouts that could allow for the emergence of a rich variety of formal solutions when manipulated by the Generative System was one question researched in this paper. Determining adjacencies was perceived as a major problem, so a simple experimental layout was chosen where all adjacencies between spaces were predetermined.

The basic layout from where the new solutions could emerge was simple: in plan, it was a square divided in four smaller similar squares. In the 1<sup>st</sup> floor, this corresponded to four rooms (R1, R2, R3 and R4, in Figure 2), which could vary in their length and width, but were constrained to have the same height. In the 2<sup>nd</sup> floor, there were four other rooms (R5, R6, R7, and R8), but instead of only varying in length and width, they could vary in height too. Roof tilts could vary from a flat roof to a maximum of 45°. The azimuth of the tilt could also vary from 0° to 90°, as shown in Figure 2 by the arrows. Whenever there was a tilted roof, a roof monitor with length equal to the corresponding wall was also generated.

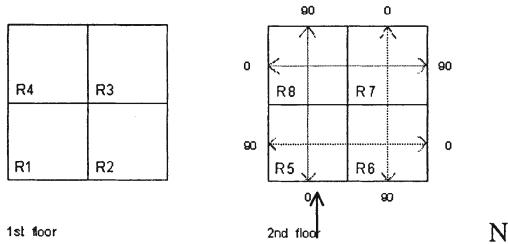
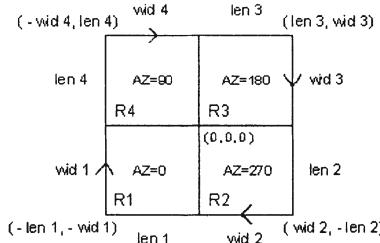


Figure 2. Basic layout for 1<sup>st</sup> and 2<sup>nd</sup> floor. Arrows show roof tilt direction.

This simple problem has nonetheless 44 independent variables and generates about 350 dependent variables that must all be correctly coded for the program to run without mistakes. By looking at the specificities of passing building geometrical information into a simulation program that calculates heat transfer, it will be possible to understand the reason why so many dependent variables are generated.

The Building Description Language (BDL) used by DOE2 has three distinct coordinate systems, namely the building, room and wall systems. To locate a new space in the overall building layout, it is first necessary to correctly determine its coordinates in the building coordinate system. The building coordinate system origin (0,0,0) was defined as the intersection point of the two axes defining the four squares. From that reference point, it is possible to correctly locate each of the spaces, even though their dimensions are variables generated by the genetic algorithm. By using the insertion point of each room as its lower left corner, and considering the rooms' azimuth (North = 0, East = 90, South = 180, West = 270), it is possible to encode the insertion point of each space as shown in Figure 3. A similar procedure was used to locate the 2<sup>nd</sup> floor spaces, with the difference that the Z coordinate was not 0 as in the 1<sup>st</sup> floor, but equal to the height of the 1<sup>st</sup> floor (2.8 m). Since the height of 2<sup>nd</sup> floor rooms was allowed to vary,

the Z coordinate for the roofs was variable too. Because roof azimuths could also vary, insertion point coordinates would be different if the roof azimuth was set to 0° or 90°. This was solved using if-then rules.



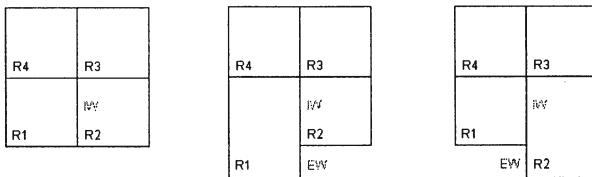
*Figure 3.* 1<sup>st</sup> floor with room azimuths and insertion points coordinates. Arrows indicate direction of the room's azimuth.

Until now, a description for handling a variable building geometry has been presented. However, this represents one of the simplest parts of the problem. When building geometry is fed into a program like DOE2, other layers of information like topological become crucial. Adjacencies between spaces are important information for a program that calculates heat transfer across surfaces. In the example used here, this problem was simplified by choosing a layout where space adjacencies are fixed. For example, Room 1 is always adjacent to Room 2 towards the east, to Room 4 towards the north, and has external walls facing south and west. The floor is always adjacent to the ground, and the ceiling to Room 5. However, adjacencies to the external environment are not entirely described using this layout, once the spaces start to be dynamically parameterized by the GA.

Figure 4 illustrates the simplest example of altering just one room dimension. While in the first case there is only an internal wall between R1 and R2, in the second case a previously nonexistent exterior wall has appeared in R1. In the third case, a new exterior wall also appears, but this time belonging to R2. Within BDL, the interior wall must be declared for both R1 and R2, but the new exterior wall should only be declared for the room it belongs to.

While in geometrical terms interior and exterior walls may be the same vertical planes, in construction terms those entities are usually built with different materials, and that is coded into DOE2. When considering heat transfer, they are even further different. An interior wall is just a boundary between two indoor environments that are usually not very different, and thus heat transfer across the wall is small. In contrast, an exterior wall is a

boundary between indoor and outdoor environments that can be very different, and thus the wall can have significant heat transfer across it. Furthermore, for the exterior wall the azimuth it faces is important, due to its relation to sun position. Another characteristic of exterior walls is that they may have windows, which are determinant factors in the performance of the building.



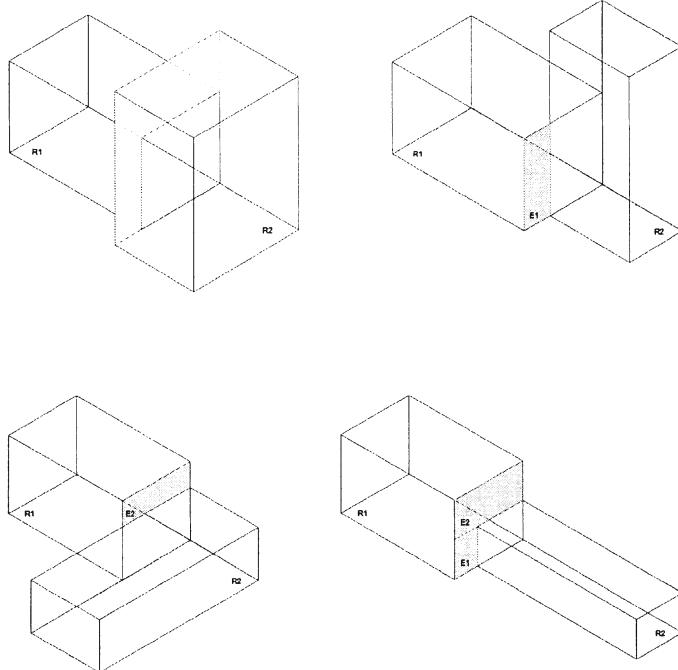
*Figure 4.* Example of external walls appearing due to parametric variation of room dimensions

In the 2<sup>nd</sup> floor, the generation of new external walls becomes more complex, since the rooms are also allowed to vary in height. For each possible adjacency situation, four different possibilities have to be predicted, which are depicted in Figure 5. In the first case, no new external wall is created for R1. In the second case, a vertical external wall appears (E1). In the third case, a horizontal one appears (E2). In the forth case, an L shaped external wall appears, which has to be decomposed in two pieces (E1 and E2), as shown in the Figure, due to BDL format specificities.

These four possibilities, that in geometric terms are quite simple, pose nevertheless a data representation problem for the generative system. Because the GS does not have a CAD interface, from which it could read information on building geometry and generate the respective BDL file, the system needs to work with a fixed-format BDL file. This file must include from the beginning all the possible geometric occurrences within the problem setup.

In this example, the most complex case would be the L shaped wall, with occurrence of both E1 and E2 components. Given this, all rooms must predict the possible existence of E1 and E2 entities. If a given entity does not occur in an individual, that entity parameters and coordinates must be driven to 0. If that entity exists, its parameter values (like width and height, for example, and other possible information about construction materials, wall solar absorptivity, internal light reflection, etc.) must be calculated from the independent variable values, and its insertion point coordinates determined for correct positioning in relation to the space, using the room coordinate system (not the building coordinate system as before). For

simplicity, the code prevents the appearance of windows in these external walls, whose existence is uncertain, as this would add another layer of complexity to the problem. However, in a more complex problem formulation it would also be possible to predict the appearance of openings in the newly generated facades.



*Figure 5. New external walls in R1, according to R2 dimensions*

A similar procedure to that used for 2<sup>nd</sup> floor external walls was applied to 1<sup>st</sup> floor roofs, and exterior floors slabs of 2<sup>nd</sup> floor rooms. In all of them, the most complex case is the existence of an L shaped element, and thus all rooms must incorporate the possibility of the occurrence of this case. If any elements do not exist in a given solution, the program must drive their values to 0. Forgetting to include any of these surfaces can introduce significant errors into the search procedure, as they represent important heat transfer areas. Finally, interior walls between adjacent spaces must also be calculated.

As for window size and positioning, a number of issues emerge when dealing with variable building shapes. When the building shape is fixed, it is possible to easily determine the upper bounds for window size, as those are

limited by the dimensions of the wall in which the window is located. However, if the wall size is not known in advance, since it is a variable itself, it is not possible to determine those bounds in advance. This represents a drawback in terms of standard genetic algorithm functioning. In common GA implantations, the constraints for each variable are determined in advance, prior to running the program. To overcome the fact that constraints are not known in advance, they would have to change dynamically during program execution. Since this dynamic constraints algorithm has not yet been implemented, it was necessary to find a simplified solution for the experiments presented here. This solution was to make the window width equal to wall width minus external walls thickness, thus becoming a dependent variable. In terms of height, 1<sup>st</sup> floor windows posed no problems, as wall height was fixed and constraints could be determined in advance. For the 2<sup>nd</sup> floor, the maximum window height was set equal to the minimum wall height, to ensure windows would always fit into the respective wall, no matter what their height might be.

These simplified rules have the disadvantage of allowing little variation in façade design. Windows always stretch from wall to wall, and vary in height only. This led to a certain standardization of generated facade solutions, which is nevertheless counteracted by the great variety of shapes produced by the GS. To introduce more diversity into the experiments, and also as an useful environmental analysis strategy, window height could be driven to 0, meaning that if the GS found that excluding a window was beneficial in terms of overall building performance, it was allowed to do so.

The location of daylighting reference points has to be calculated by the program for each new solution generated. The rules for placing the sensors were: one sensor in the center of the space, and the other 2 meters away from the innermost walls, that is, the walls that have no windows. This tries to ensure that natural light is used throughout the space, and that it penetrates into the deeper areas of the rooms. The Z coordinate of the sensors is 0.75m, approximately desktop height.

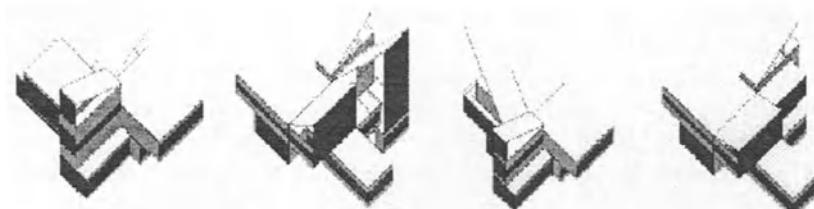
Other parameters, like room area and volume, have to be calculated after the independent variable values are generated. In calculating the volume, it should not be forgotten that when a tilted roof exists, the corresponding volume under it must be added to the initial parallelepiped. Not including this additional volume can introduce significant inaccuracies in calculations, as air volume is an important factor for HVAC systems, mainly air-based ones.

A remark should be added about self-shading calculations. DOE2 will not calculate building self-shading unless appropriate surfaces are explicitly declared as shading surfaces. Since from the outset it is unknown what

surfaces will be shading others, all exterior planes (wall, exterior floors, roofs) should be declared as shading surfaces so that they are considered as such. This can be particularly important in cases where a 2<sup>nd</sup> floor space projects over a 1<sup>st</sup> floor one, and acts as an overhang for it, influencing both solar gain and daylighting levels, or for rooms with different heights, tilted roofs, and recessed external walls in relation to adjacent ones.

#### 4. Experiments

Some initial random configurations generated by the GS are shown in Figure 6.



*Figure 6. Some initial random configurations generated by the GS*

A major concern in these experiments was that the main strategy the GS would use to reduce building energy consumption would be reducing building area. No matter how efficient and adapted to the outdoor environment a large building is, it will always use more energy than a very small building. So, the predicted outcome from experiments using energy consumption as objective function would be a population of minimum-dimensions buildings, with some variation in façade design.

It was thus evident from the outset that building area would have to be included in the fitness function. This was implemented in two different ways. One implied the use of penalty functions related to area requirements violations, and the other used Energy Use Intensity (EUI) as the objective function, which translates energy use per unit area.

##### 4.1 PENALTY FUNCTIONS

The use of penalty functions was first experimented. Each of the building floors was required to have a certain area, related to functional and programmatic requirements. The GS could assign different areas to each of the four spaces in that floor, according to the best strategy for good environmental performance, but the total area of the floor would have to equal a certain number of square meters. Penalties were calculated

according to the amount by which that area requirement had been violated. The absolute value of the difference between the required area and the actual one was used, so penalties would be applied both for too small and too large spaces. A similar method was used to 2<sup>nd</sup> floor area requirements. Penalties were added to the original fitness value of the solution (annual energy consumption), and degraded that fitness according to the extent of the violation.

This method intended to ensure that if the GS tried to reduce floor area to a minimum so that energy consumption would be low, a high penalty function would significantly degrade that solution's performance and make the GS move away from it. The penalty values could vary by a large extent, since they are based on floor area calculations, and floor area is allowed to vary significantly in this problem set. Each room dimension can vary between 3 m and 15 m, so room area can range from 9m<sup>2</sup> to 225 m<sup>2</sup>. If the four rooms are aggregated into the total area for each floor, each floor plan can range from 36 m<sup>2</sup> to 900 m<sup>2</sup>, a very significant variation. For this reason, the penalty values for solutions that seriously violated the area requirements could be quite high. The penalties were based on a target area of approximately 470 m<sup>2</sup> per floor, which is about half of the allowed range. Following this method, the penalty for the 1<sup>st</sup> floor area violation was calculated as:

$$\text{penalty1} = (\text{abs}(470 - \text{floorarea1}) / 470) * 70$$

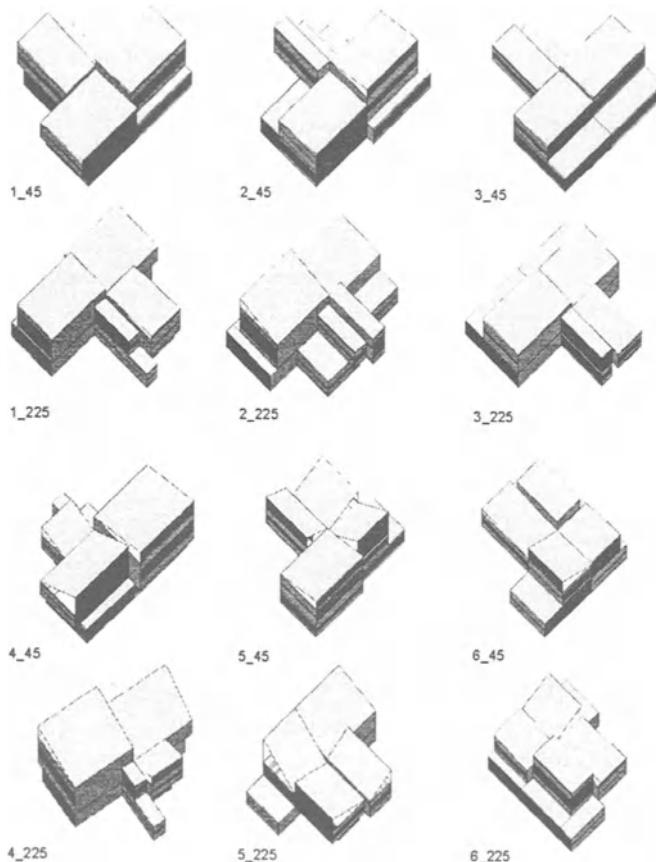
The factor of 70 was adopted after parametric experiments using other factors, as it was found that too small penalty factors would still make area reduction the best strategy for energy savings. The penalty function method was developed because there is no formal way in genetic algorithms to constraint the outcome from a combination of variables. For example, it is possible to place upper and lower bounds in a room's length and width, but not in a room's area, since it is the result of a multiplication of two independent variables.

Figures 7 and 8 show results for experiments using penalty functions, for Oporto's climate. A standard GA implementation was used, with a population size of 30 individuals, from which nine are represented. Figure 7 shows six good results, and Figure 8 shows three poor-performance ones.

The GS was run for 200 generations. From Figure 7 and Table 1, some conclusions can be drawn.

The values in the table show that the best solutions use different strategies to achieve low fitness values. While some stay closer to the required areas, increasing energy consumption but decreasing penalty values (like solution 1), others reduce their energy consumption levels by cutting

on floor area, while still achieving a good final fitness value (like solution 2).



*Figure 7.* Best solutions for Oporto using penalty functions. 1 is the best solution. For each building there is a SW ( $45^\circ$ ) and a NE view ( $225^\circ$ ). Drawings not to scale.

Solution 1 is the best. Even though it complies with area requirements, it manages to do so without greatly increasing energy consumption values, what reveals a good degree of adaptiveness to the environment. It creates elongated south-facing spaces, with large south openings, but with small windows towards the west. In general, east/west facades are smaller than south/north ones. Towards the northern side on the building, it avoids the northeast orientation, which is always unfavorable due to reduced lighting

levels for most of the day, placing in that corner very small spaces that could be used as service areas. To the northwest, it used larger volumes (even if smaller than those facing south), and used the 2<sup>nd</sup> floor volume to shade the 1<sup>st</sup> floor west-facing opening, to control solar gain, such as it had done with the southwest room. In the overall, this seems a balanced and reasonable solution, and demonstrates the GS is able to create appropriate geometries for a given problem.

TABLE 1. Best solutions for Oporto using penalty functions (solution numbers correspond to those in Figure 7)

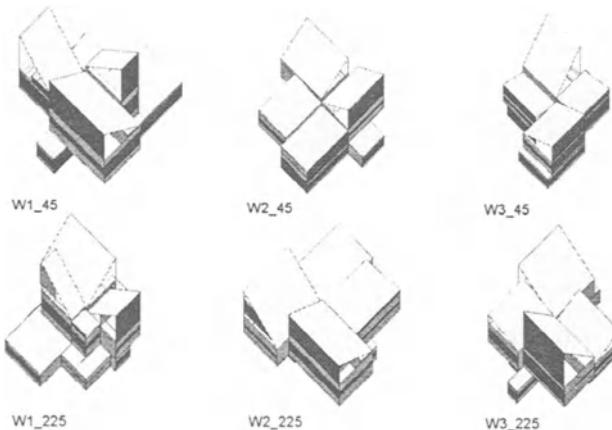
| <b>Solution #</b> | <b>Fitness without<br/>penalty (MWh)</b> | <b>Penalty for area<br/>violation</b> | <b>Final fitness<br/>value</b> |
|-------------------|------------------------------------------|---------------------------------------|--------------------------------|
| 1                 | 96                                       | 1                                     | 97                             |
| 2                 | 82                                       | 26                                    | 108                            |
| 3                 | 98                                       | 11                                    | 109                            |
| 4                 | 92                                       | 19                                    | 111                            |
| 5                 | 105                                      | 9                                     | 114                            |
| 6                 | 85                                       | 30                                    | 115                            |

Solution 2 is not very dissimilar to 1 towards the south side, but towards the north it substantially reduced room areas. This generated an energy reduction, but also a quite large area penalty, which lead to a solution over 10% worse than 1. Solution 3 is not too dissimilar to 1, but has higher energy consumption even though its area has been reduced (as shown by the penalty function value), probably because of the exposed south-facing 1<sup>st</sup> floor roofs, since the 2<sup>nd</sup> floor is often recessed into a terrace configuration. It also has quite larger west-facing openings. Solution 4 is interesting in that it plays with the tilts of the roofs to generate south and north lighting sources, privileges south-facing spaces, and uses projected volumes to shade 1<sup>st</sup> floor west facing windows. However, it reduces too much the north side volumes, and is penalized in terms of areas. Solutions 5 and 6 start to be more hybrid. 5 stays close to the area requirements, but has high energy consumption levels, probably due to the large glazing areas towards east and west, namely under the steep roof facing those orientations. Solution 6 has a very high area penalty, and probably too big east-facing windows.

Solution 1 seems to be a quite adapted solution to Oporto's mild climate. One could argue that some of the other solutions may be more appealing in aesthetics terms. This raises the issue of the interactivity between the architect's intentions and the GS. After being presented with these initial solutions, it is then up to the architect to decide what paths of exploration he is willing to take to achieve solutions closer to his intentions. He can decide to change some constraints, run a MicroGA (Krishnakumar 1989) to explore

the neighborhood of good solutions like 1, or manually perform same changes and do a DOE2 simulation of the modified design to assess their impact.

Looking at poor performance solutions is also an useful exercise, to consider which design features have a negative impact on performance. In Figure 8 and Table 2, the worst solutions in the final population are presented.



*Figure 8.* Worst solutions in the last generation, using penalty functions. W1 is the worst solution. For each building there is a SW (45°) and a NE view (225°).

TABLE 2. Worst solutions using penalty functions (solution numbers correspond to those in Figure 8)

| Solution # | Fitness without<br>penalty (MWh) | Penalty for area<br>violation | Final fitness<br>value |
|------------|----------------------------------|-------------------------------|------------------------|
| W1         | 109                              | 44                            | 153                    |
| W2         | 113                              | 29                            | 142                    |
| W3         | 119                              | 22                            | 141                    |

In general, these solutions not only have large penalty values for area violations, but they also show high energy consumption levels. This is mainly due to large west and east facing glazing areas from roof monitors generated by the steep roofs, and deep overhangs shading south and north openings, thus blocking daylight and useful solar gains. It should be added that the external floors were modeled with no insulation (they are just

concrete slabs with an interior finish) so in general the appearance of large projected volumes is not encouraged.

The use of penalty functions introduced another level of complexity into the interpretation of results. The two factors (energy use and floor area violation) are somehow difficult to isolate for result analysis. For this reason, it was decided to do another set of experiments where floor area would also be included in the final evaluation, but using a different method. Energy Use Intensity (EUI) was used, measuring energy consumption per unit floor area. This seemed a more effective method to evaluate the relative environmental performance of different solutions, independently of the overall floor area.

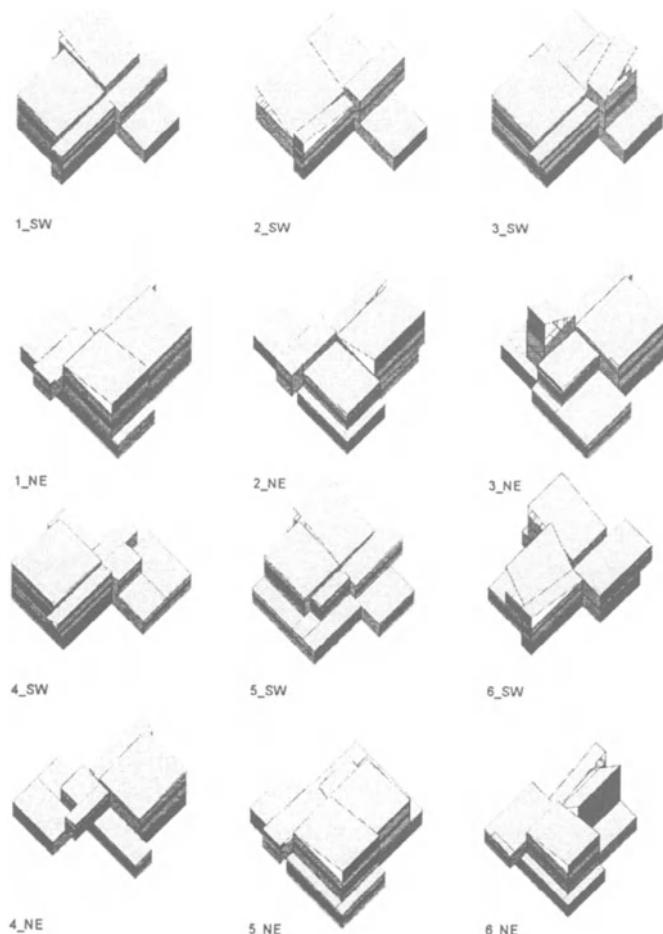
#### 4.2 ENERGY USE INTENSITY (EUI) FOR OPORTO EXPERIMENTS

Figure 9 shows results for Oporto climate using EUI as the final objective function. The best individual is solution 1, which is somehow different from the one found using penalty functions. Since in this case the GS is not asked to assign a given area to the building, it tends to size and distribute spaces in a different way. Slim, shallow, all-glazed elements are used towards the south, in a configuration that resembles a sunspace. Larger spaces, of more bulky proportions, appear towards the north of the building. Those spaces have lower surface-to-volume ratio, and thus have less heat transfer surfaces to lose energy through. However, they may be more difficult to bring daylight into, since they are deeper. For that reason, the GS generates quite large openings in these spaces, mainly north-facing ones (including roof monitors) but also towards east and west, as those bulky spaces require large openings for appropriate daylighting. In general, the proportions of the building tend to orientate the widest facades towards south and north, and the shorter ones to east and west, as expected. The best performing shapes are in general quite compact, with reduced roof areas in the 1<sup>st</sup> floor, and few overhangs or projecting elements. This overall layout is kept in solutions 2 and 3, with some minor variations, but starts to suffer more significant changes as the fitness of solutions decreases, with results becoming more difficult to interpret for these intermediate solutions.

Table 3 shows the fitness function values for each of the solutions shown in Figure 9, in terms of EUI (kWh per square foot). Among the six best solutions, there is more than 10% variation in Energy Use Intensity.

Finally, we look at the worst performing solutions to try to find patterns of elements leading to poor behavior. Table 4 and Figure 10 show three of the worst individuals, with the worst one on the left (#1). The main characteristics of these solutions are: high surface-to-volume ratios, caused by slim, elongated shapes; exposed external surfaces, like external walls, 1<sup>st</sup>

floor roofs and floors from projected 2<sup>nd</sup> floor elements, which have no insulation; and large glazing areas facing unfavorable orientations like east and west.



*Figure 9.* Best solutions for Oporto using EUI. #1 is the best solution. For each building there is a SW and a NE view. Drawings are not to scale.

Knowing that these configurations lead to lower performance solutions can either make the designer move away from them, or if the type of architectural language the architect is looking for somehow matches these

configurations, they should be counteracted by properly insulated external surfaces, wall and roof solar absorptivity, more sophisticated glazing systems, and appropriate use of shading.

TABLE 3. Fitness values in EUI for the best solutions (solution numbers correspond to those in Figure 9). EUI is in kWh per square foot.

| <b>Solution #</b> | <b>Fitness value (EUI)</b> |
|-------------------|----------------------------|
| 1                 | 9.02                       |
| 2                 | 9.57                       |
| 3                 | 9.69                       |
| 4                 | 9.79                       |
| 5                 | 9.88                       |
| 6                 | 10.04                      |

TABLE 4. Worst solutions in the last generation for Oporto, using EUI.

| <b>Solution #</b> | <b>Fitness value (EUI)</b> |
|-------------------|----------------------------|
| W1                | 13.18                      |
| W2                | 11.85                      |
| W3                | 11.79                      |

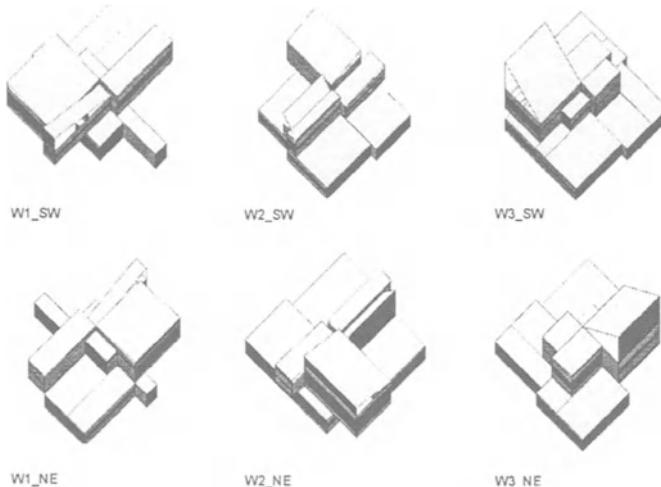
Nonetheless, these solutions will usually imply higher construction costs, so if low-cost construction is an issue, the architect may try to infer from the GS results the type of shapes that lead to better performance by adaptation to the environment.

#### 4.3 ENERGY USE INTENSITY (EUI) FOR CHICAGO EXPERIMENTS

EUI experiments were also performed for Chicago, to illustrate a situation dominated by a very cold climate. Results are shown in Figure 11 and Table 5. The first observation from Table 5 is that Energy Use Intensity is much higher in Chicago than in Oporto, as expected from the extreme winter conditions. Another pertinent observation is that the performance among the six best solutions degrades faster than for Oporto. While for Oporto there was a decrease of 10%, here the decrease is more than 15%, showing that design features may have a larger impact in the overall performance of the building.

The best solution for Chicago is a very compact shape, with minimum exposed roofs and floors. It has three bulky, deep spaces in each floor, plus a sunspace-type room towards the south, slim and all-glazed. In the deep spaces, glazing areas are quite high, for daylighting purposes, and because window surface area is not too significant in relation to the whole volume. Again, as building shape becomes fragmented, and more exposed surfaces

appear, building performance degrades, suggesting that for a cold climate like Chicago, compact shapes perform better. Small wall-to-volume ratios are preferable. Sunspaces can be beneficial too. In deep rooms, small window sizes usually correspond to a decrease in performance due to lack of daylighting, despite increased heat losses.



*Figure 10.* Worst solutions in the last generation for Oporto, using EUI. W1 is the worst solution.

In relation to the worst performing solutions, Figure 12, the most prominent characteristic is the presence of very steep roofs, with large glazing areas. The worst solution (W1) has both east and west facing roof monitors. Solutions that have south-facing roof monitors also perform poorly, but not as badly. The overall geometry of the buildings is fragmented, spaces are small, roofs exposed, and there are several overhangs, which represent large sources of heat losses due to the fact that they have no insulation, and also tend to deprive spaces underneath from useful daylighting, in a climate where shading is not a priority.

In general, the tendency that has been expressed between formal characteristics of good-performance solutions and poor-performance ones is that the former tend to have simpler, more rationalized geometries, while the latter show a variety of formal accidents and more eclectic shapes, which seem to contribute to degrade a solution's environmental performance, at least within the constraints of this problem formulation.

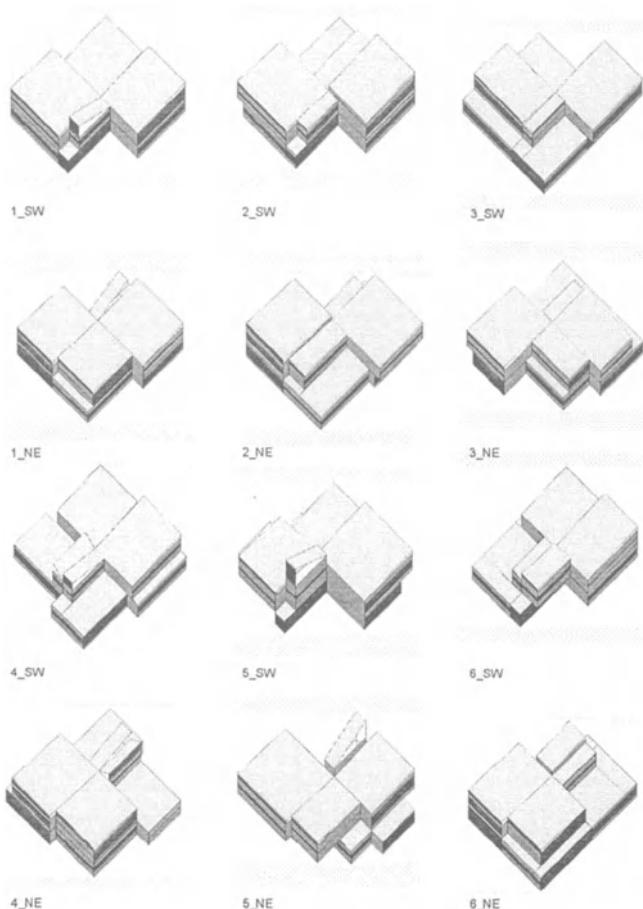
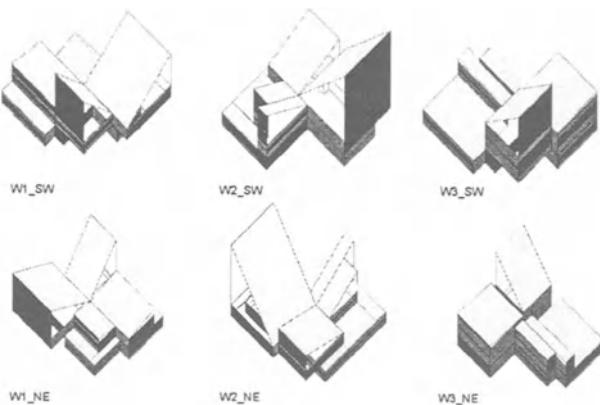


Figure 11. Best solutions for Chicago using EUI. #1 is the best solution

TABLE 5. Fitness values in EUI for the best solutions for Chicago

| Solution # | Final fitness value (EUI) |
|------------|---------------------------|
| 1          | 26.27                     |
| 2          | 27.60                     |
| 3          | 28.79                     |
| 4          | 28.88                     |
| 5          | 30.18                     |
| 6          | 30.27                     |



*Figure 12.* Worst solutions in the last generation for Chicago, using EUI. W1 is the worst solution

## 5. Conclusions

Departing from identical initial schematic layouts, the Generative System used in this paper was able to create a variety of architectural shapes that respond to the different climates where they were located, both in terms of daylighting use and control of heat losses and solar gains, leading to low-energy design solutions that differed among geographical locations.

The use of penalty functions for area requirement violations proved to be a somewhat unstable method, in that solutions generated adopted different strategies to achieve low annual energy consumption levels. Despite the high penalty factors used, some solutions still used reduced areas to maintain small fitness function values. Energy Intensity Use provides a more reliable measure to assess the effectiveness of different building configurations, combined with specific façade solutions, to achieve low energy designs. Those solutions are probably more generic, since they can be scaled up or down to achieve the desired area requirements, without significantly altering their behavior. Further work is needed to assess to what extent the scale factor would alter the solution behavior.

Future work will also address the issue of incorporating dynamic constraints into the system, so that an extra degree of flexibility is added to design elements like windows, roof monitors and other light sources.

This GS is not to be regarded as an optimization tool, but instead as a generative mechanism whose goals are not only to reduce energy consumption in buildings, but also to suggest alternative building

configurations and work as an augmented design aid. The particular shapes generated in these experiments are a result of the initial layout, rules and constraints applied. Different initial conditions would lead to the emergence of other design solutions, suggesting this Generative System can be a powerful tool for architects to quickly study alternative low-energy designs and understand which architectural features are more decisive towards achieving the desired performance.

### Acknowledgements

Luisa Caldas developed this work as part of her Ph.D. dissertation in Architecture at Massachusetts Institute of Technology, with a grant from PRAXIS XXI program, Fundação para a Ciência e a Tecnologia, Portugal.

### References

- Cagan, J and Mitchell, WJ: 1993, Optimally directed shape generation by shape annealing, *Environment and Planning B* **20**: 5-12
- Caldas, L and Norford, L: 2000, Energy design optimization using a genetic algorithm, *Automation in Construction* **11**(2): 173-184
- Caldas, L and Rocha, J: 2001, A generative design system applied to siza's school of architecture at oporto, in JS Gero, S Chase and M Rosenman (eds), *CAADRIA'01*, Key Centre of Design Computing and Cognition, Univeersity of Sydney, Sydney, pp. 253-264
- Damski, J and Gero, J: 1997, An evolutionary approach to generating constraint-based space layout topologies, in Junge, R (ed.), *CAAD Futures1997*, Kluwer, Dordrecht, pp. 855-874
- Krishnakumar, K: 1989, Micro-genetic algorithms for stationary and non-stationary function optimization, in G Rodriguez (ed.), *Intelligent Control and Adaptive Systems*, SPIE, Bellingham, Philadelphia, pp. 289-296.
- Mitchell, W: 1977, *Computer-Aided Architectural Design*, Petrocelli/Charter, New York.
- Monks, M, Oh, B and Dorsey, J: 1998, Audiointimization: Goal based acoustic design, *MIT Technical Report MIT-LCS-TM-588*
- Radford, A and Gero, JS: 1978, A dynamic programming approach to the optimum lighting problem, *Engineering Optimization* **3**(2): 71-82
- Winkelman, F: 1993, Daylighting calculation in DOE-2, *Lawrence Berkeley National Laboratory Report*, LBL-11353, May 1983

## STRATEGIC SHAPE DESIGN

TOSHIHARU TAURA, TAKAYUKI SHIOSE AND RYOHEI  
ISHIDA

*Kobe University  
Japan*

**Abstract.** Although many attempts have been made to enable creative and effective design activities using a computer, many design activities depend on the designer's personal ability and are learned through interpersonal communication. These limitations are thought to be due to the fact that the knowledge we use does not cover the entire design process, comprising the problem-forming process and the problem-solving process, in a systematic manner. In this study, a method of dealing with the design information, in which the problem-forming process is considered, and dealt with simultaneously the problem-solving process, is proposed. This method is composed of three steps: (1) determination of the evaluation function from the shape, (2) synthesis (composition) of the evaluation function, and (3) generation of the shape candidates from the evaluation function. Interaction with a computer system implementing this method is expected to assist the shape design at an early stage. In this study, this method is developed and realized in a computer system.

### 1. Introduction

Many attempts have been made to enable creative and effective design activities using a computer. Although these attempts have been somewhat successful, we must admit that these successes are highly limited and many design activities are still dependent on the designer's personal ability. The primary reason for these limited successes is that we cannot outline the whole process of design systematically. That is, our attempts to computerize the design process have focused only on one side, the problem-solving processes, while a design actually involves two processes: a problem-forming process and a problem-solving process. The former process has been studied practically with a cognitive concern for 'creativity' (Gero 1996; Finke et al. 1992; Tversky and Kahneman 1981; Weisberg 1992) and the latter has been studied with a computational concern for 'optimization' or

'efficiency' (Chapman and Jakiela 1994; Paul and Beitz 1997; Takeda et al. 1990; Tutiya 1993).

Here, a problem-forming process can be regarded as a process that defines a certain design space by determining an evaluation function, constraints and some parameters. In conventional studies on the design process, the determination of these factors had to be done by humans. Once such a design state space had been clearly defined, the remainder of the design process could be regarded as a step-by-step problem-solving process (e.g., Paul and Beitz 1997, Takeda et al. 1990) and was usually formalized by using some optimization techniques. However, it is difficult for the designer to define this initial design state space in practical design processes except under laboratory conditions. Cognitive studies on the 'creative' design process (Gero 1996; Finke et al. 1992) have pointed out the importance of dynamically changing such a design space; however, there are few studies that have manipulated the problem-forming process by computational methods.

The purpose of this study is to render the problem-forming process operable with computers. In concrete terms, we focus on the dynamical changes of an evaluation function as a representative of a design state space and implement the function in our proposed interactive design support system in the domain of shape design. We call this design process 'Strategic Shape Design' in which it is necessary to manage two processes, the problem-forming process and the problem-solving process.

In Strategic Shape Design, the design state space plays a more important role than simply being the necessary condition for designing a shape, namely, it has a role as a certain alternative representation for the design solution. The reason is that the design state space corresponds to the trend of what the design solution should be. Therefore, searching for a design solution can be replaced with searching for a design state space, which is more abstract than the design solution itself. This replacement is expected to enable us to examine the design process at a more conceptual level.

In the next section, our proposed method is outlined. The third section presents to the system architecture of our proposed system. The fourth section shows some experimental results. Finally, our conclusions are presented.

## 2. Outline of New Method

The industrial design process includes several phases, such as investigation of customer needs, conceptualization, and preliminary refinement... etc (Ulrich and Eppinger 1995). In particular, the early phases, conceptualization and preliminary refinement, are known to be important

processes for deciding the quality of the product, but are difficult processes to represent using computational terminology. These phases, conceptualization and preliminary refinement, correspond to the above-mentioned problem-forming process. Here, important issues are how we should represent ‘conceptualization’ and how we should manipulate the concept to achieve ‘preliminary refinement’.

A form feature model (Ludy, Dixon and Simmons 1986) is one means of representing the ‘concept’ by expressing form features directly. Although we have made some attempts which are similar to a form feature model, most of them unintentionally fixed the features in the design process. If the representation of the ‘concept’ is fixed, the ‘preliminary refinement’ phase will also converge instantaneously. In contrast, outstanding designers are considered to avoid such sudden convergence by changing the representation of the ‘concept’ dynamically. Suwa et al. (1997, 1996) pointed out that professional designers excel in discovering more new viewpoints from the same drawings than others do. Also, professional designers themselves discover clues to new viewpoints in the process of sketching by hand. Suwa reported that sketching by hand is sufficiently ambiguous for designers to find new ideas.

Sometimes the ‘concept’ can be expressed as a subset of form features expressed by the exaggeration of some parts of features. Therefore, in order to represent the ‘conceptualization’ phase, more abstract and fundamental expressions are needed. In this study, we focus on an evaluation function which is a necessary component for establishing the design state space, and implement it as a function expression by using genetic programming (GP, Koza 1992) methods. Expressing an evaluation function by GP enables us to manipulate it (e.g., to change, to synthesize and so forth,...etc), and such manipulation of an evaluation function corresponds to the ‘preliminary refinement’ phase.

However, even if an evaluation function becomes manipulatable through GP expression, the designer cannot grasp such GP expressions directly. The designer cannot help constantly judging whether or not the evaluation function given by the system is valid through considering the practical shapes resulting from the problem-solving process using the tentative evaluation function. For such interaction, our proposed system must be equipped with an interface which allows the designer to select favourite shapes from among a number of choices. Summarizing the above discussions, we assume the following hypothesis.

**Hypothesis:** A method of dealing with shape information, in which the evaluation function is determined from the shape, then synthesized and

applied to the shape through interface with the designer, is effective in the early stage of shape design

In this study, we develop a method of realizing and verifying the above hypothesis, Figure 1. In order to operate the evaluation function using a computer, the method involves the following steps. Steps (1) and (2) are part of the 'problem-forming process' and step (3) is part of the 'problem-solving process'.

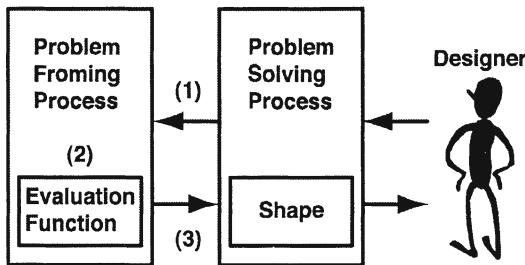


Figure 1. Concept of strategic shape design

### 1. Obtaining the evaluation function

Usually, a designer cannot express his/her own evaluation function explicitly. Therefore, in this method the evaluation function is obtained in the following two ways.

#### (a) Obtaining the evaluation functions from shape samples.

The evaluation function is acquired autonomously from some shape samples by the computer. This process appears to be the opposite of the conventional design process in which the shape candidates or shape solutions are generated using the evaluation function.

#### (b) Obtaining the evaluation functions from the designer's preferences.

The acquired evaluation function is not guaranteed to be equal to the designer's implicit evaluation function. Therefore, it must be verified whether or not the obtained evaluation function will indeed yield shapes which the designer will choose.

### 2. Synthesizing (composing) the evaluation functions

In this phase, new evaluation functions are synthesized (composed), using GP. This operation has two roles. The first one is to adapt the evaluation function to the designer's concept. The second one is to change the concept or to combine a number of concepts intentionally in order to present the designer with a new viewpoint.

### 3. Developing shape candidates from the acquired evaluation functions

In this phase, the shape candidates are generated, using the genetic algorithm method (GA), from the new synthesized (composed) evaluation functions and shown to the designer.

### 3. System Architecture

Here, the above-mentioned three phases of strategic design are described in detail.

#### 3.1 AUTONOMOUS EVALUATION-FUNCTION-ACQUIRING PROCESS

In our experiments, an evaluation function is represented by a tree structure (an example of an evaluation function is shown in Figure 2). This tree structure usually has two types of nodes: a terminal node and a nonterminal node. The former one is represented by one of the variables which describe the target design shape. The latter one is represented by operations (e.g., the four basic operations of arithmetic, some conditional clauses, and so forth). One conditional clause enables us to express qualitative relationships among variables that describe the shape and, as a result, provides a powerful representation of what the user wants to express.

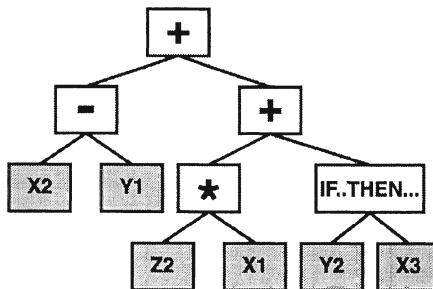


Figure 2. An example of evaluation function

At the beginning of the design process, the system must specify primitive evaluation functions (PEF) from some sample shapes (the number of sample shapes is N0). The autonomous evaluation function acquisition system (AES) holds the determined number (N1) of functions as individuals in GP and acquires the evaluation function by evolving the individuals. Then, the individuals are sent to the shape forming system (SFS) (P1 in Figure 3), the details of which are explained in 3-3, and the shapes which fit the PEFs are generated using GA. These shapes are sent to the evaluation system of the AES and compared with the original sample shapes (P2 in Figure 3). The individuals are evaluated in order to decrease the difference between the

acquired shapes and the original shapes. The individuals are also evaluated through interaction with the user (P3 in Figure 3) via interactive evolutionary computation algorithms.

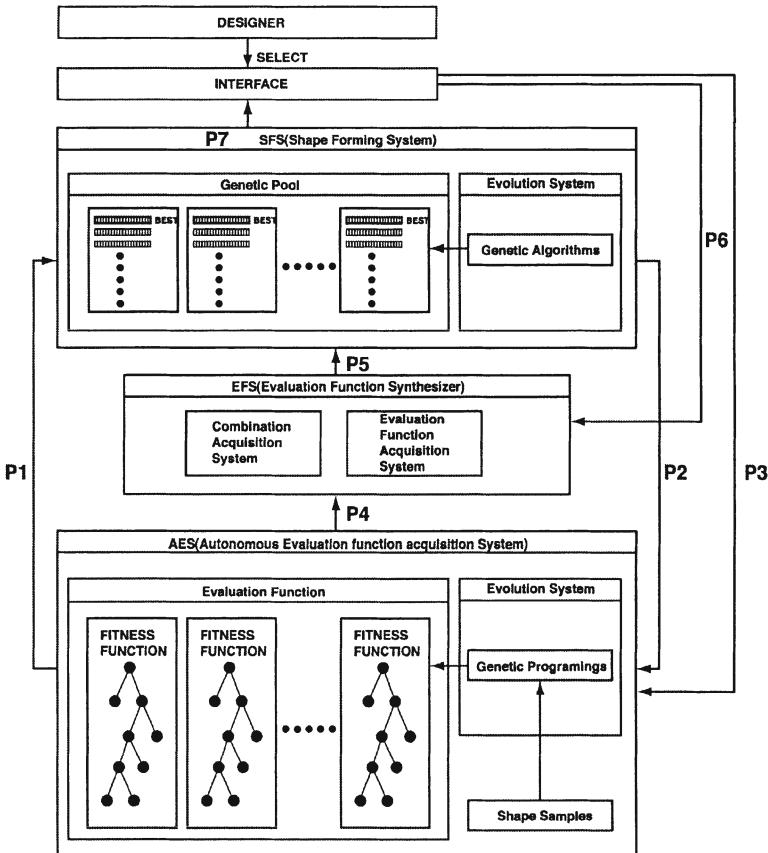


Figure 3. System architecture

Here, the user is defined to be part of the loop in order to estimate the number of evaluation functions per  $T$  times of P2. The reason for this is to reduce the user's work load in selecting the evaluation function.

### 3.2 EVALUATION-FUNCTION-SYNTHESIZING PROCESS

Next, the evaluation-function-synthesizing system (EFS) acquires the determined number ( $N_0$ ) of functions as the PEFs from the above-mentioned AES. EFS holds the determined number ( $N_1$ ) of functions as individuals,

and the validity of the new evaluation functions is estimated by the following method.

1. First, EFS generates the new evaluation functions ( $E_j$ ) by combining or rearranging the PEFs (P4 in Figure 3). In particular, the balance of combined PEFs is represented by

$$E_j = \sum_i W_{ij} * PEF_i + W_{Mj} * E_{Mj}, \quad (i=1, \dots, N_0, j=1, \dots, N_1),$$

where  $\sum_i W_{ij} + W_{Mj} = 1.0$ . Here,  $W_{ij}$  and  $W_{Mj}$  represent coefficient values for defining the balance of combined PEFs.  $E_{Mj}$  stands for the appendant evaluation functions that widen and accentuate the range of the new evaluation function ( $E_j$ )'s expressions. Figure 4 shows the conceptual image of such combined evaluation functions.

2. Next, the individuals of such new evaluation functions are sent to SFS, as in the case of the above-mentioned P1 (P5 in Figure 3).
3. Then, the shapes are validated through interaction with the user (P6 in Figure 3). In particular, the new evaluation functions are updated by a kind of interactive GP. Concretely, the balance of combinations is changed by updating the coefficient values of equation (1) and the mutation operator is also applied to the appendant evaluation function  $E_M$ , Figure 4.

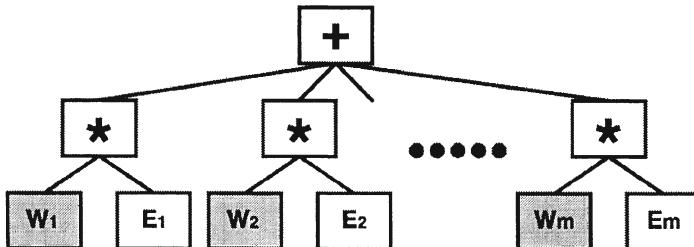


Figure 4. Conceptual image of synchronization process

### 3.3 SHAPE-FORMING PROCESS

The evaluation functions newly obtained in EFS are sent to the shape-forming system (SFS) (P7 in Figure 3). SFS holds the same number of genetic pools for each evaluation function. It is a gene in the genetic algorithm method, the phenotype of which is the representation of shape, and generates the best-fitting shape for each evaluation function by evolving the individuals.

#### 4. Experiments

In this study, we asked subjects to solve a given design problem using our proposed method. We believe that the effectiveness of this method can be verified if the balance of synthesized evaluation functions converges to a certain value. Some subjects were asked to “design a car shaped like a dolphin.” First, the subject had to imagine the shapes of a car and of a dolphin respectively, and then design the assigned object by combining the two different images. Figure 5 shows sample shapes of a car and a dolphin.

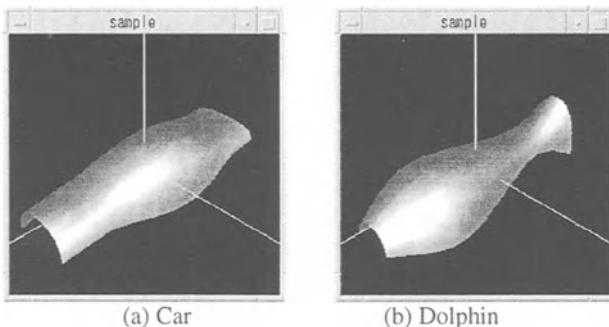


Figure 5. Sample shapes of a car and a dolphin

##### 4.1 SET OF EXPERIMENTS

Each image has a default format and it is described by 19 variables (e.g., the width of the bottom, the curvature of the side, etc.). Then, the evaluation function is described by these 19 variables as terminal nodes and by these following operations as nonterminal nodes, Table 1.

TABLE 1. Nonterminal nodes in evaluation functions

|               |                                                   |
|---------------|---------------------------------------------------|
| +             | Addition                                          |
| -             | Subtraction                                       |
| *             | Multiplication                                    |
| /             | Division                                          |
| RETURN<br>100 | IF(argument1>argument2)THEN(return<br>100)ELSE(0) |
| RETURN 50     | IF(argument1>argument2)THEN(return 50)ELSE(0)     |

Additionally, the parameters for GP (applied in AES and EFS) and the parameters for the genetic algorithm (applied in SFS) are given as follows, Tables 2 and 3, respectively.

TABLE 2. Parameters in GP

|                          |     |
|--------------------------|-----|
| Population size (N1)     | 100 |
| Number of generation     | 100 |
| User Interaction Rate: T | 10  |
| Crossover probability    | 0.8 |
| Mutation probability     | 0.5 |
| Inversion probability    | 0.5 |

TABLE 3. Parameters in GA

|                       |     |
|-----------------------|-----|
| Population size       | 50  |
| Number of generation  | 50  |
| Crossover probability | 0.5 |
| Mutation probability  | 0.2 |

SFS displayed 12 shapes to the user and the user's selection was fed back to the AES as a reinforcement signal (P3 in Figure 3).

#### 4.2 FIRST PHASE (AUTONOMOUS EVALUATION-FUNCTION-ACQUIRING PROCESS)

At the beginning of Phase 1, AES was required to generate two primitive evaluation functions ( $E_c$  and  $E_d$ ) corresponding to the two sample shapes of a car and a dolphin, by means of the above-mentioned three phases (P1, P2, P3). Here, Figure 6 shows an example of the acquired PEFs ( $E_d$ ) and Figure 7 shows example shapes generated from the PEF.

Figure 6 shows an example of the acquired evaluation function for the shape of a dolphin ( $E_d$ ). The right part of Figure 6 represents part of the tree structure for the underlined term in the left part of Figure 6. Here, we can see that this evaluation function estimates whether or not the shape has the narrow parts of the tail and fins (because the terminal node ( $Y_8/Y_9$ ) corresponds to such narrow parts).

#### 4.3 SECOND PHASE (EVALUATION-FUNCTION-SYNTHESIZING PROCESS)

EFS must combine with the acquired PEFs and update the new acquired evaluation functions. Figure 8 shows the acquired shapes after 10

generations from each new evaluation function. Additionally, Figure 9 shows the change in the balance of combined PEFs.

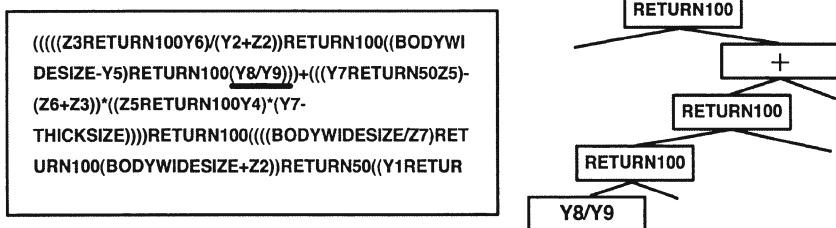


Figure 6. Example of the Acquired Evaluation Function (Ed)

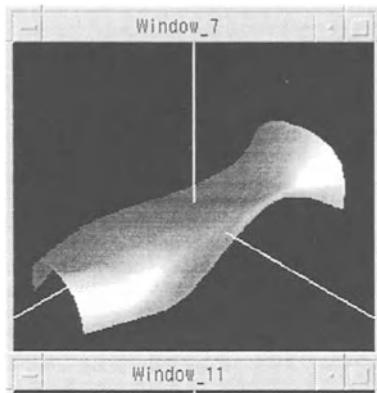


Figure 7. Example shape generated from the New Evaluation Function (Ed)

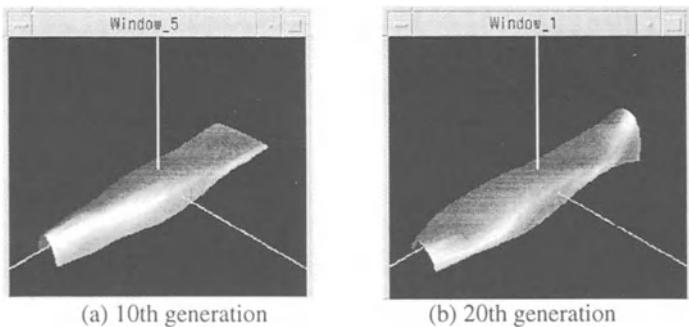


Figure 8. Example shape generated from the New Evaluation Function

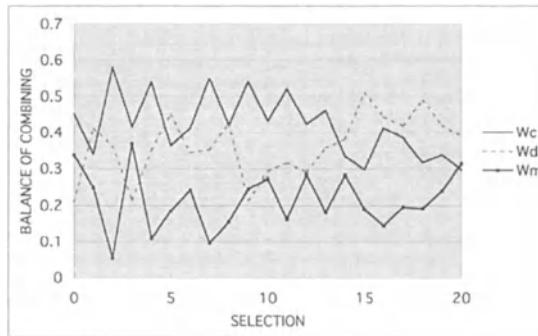


Figure 9. Changes in the combining balance

Figure 9 shows that at the beginning of interaction, the balance among synthesized evaluation functions fluctuates. After several interactions between the subject and the system, it was found that this balance converged to a certain value.

#### 4.4 DISCUSSION

In this experiment, our proposed method could be used to combine the primitive evaluation functions acquired from the sample shapes and provide the subject with new candidates generated from the synthesized evaluation functions. Our proposed method may help the subjects to design new shapes since the balance among synthesized evaluation functions converged to a certain value.

#### 5. Conclusions and Future Work

In this study, we demonstrated that the entire design process is composed of two processes: the problem-forming process and the problem-solving process. We suggested a new method, called “strategic shape design”, for realizing both processes, and implemented it by using interactive GP and GA. Experimental results verified that our proposed strategic shape design enables us to combine different images without losing the key characteristics that define each shape.

In future work, we must verify the effectiveness of synthesizing evaluation functions through comparison with other methods of supporting the early stage of the design process.

## References

- Chapman, CD and Jakela, MJ: 1994, Genetic algorithm-based structural topology design with compliance and manufacturability considerations, *Journal of Mechanical Design* **116**: 89-98.
- Finke, RA, Ward TB and Smith SM: 1992, *Creative Cognition*, The MIT Press, Cambridge, MA.
- Gero, JS: 1996,:Creativity,emergence and evolution in design, *Knowledge-Based Systems* **9**: 435-448.
- Koza, J: 1992, *Genetic Programming, On the Programming of Computers by means of Natural Selection*, MIT Press, Cambridge, MA.
- Ludy, SC, Dixon, JR and Simmons, MK: 1986, Designing with features: creating and using a feature database for evaluation of manufacturability of castings, *ASME Computers in Engineering Conference*, ASME, New York, pp. 285-293.
- Paul, G and Beitz W: 1997, *Engineering Design*, Springer Verlag, Berlin.
- Suwa, M and Tversky, B: 1996, What architects see in their design sketches: implications for design tools, *Human Factors in Computing Systems: CHI'96 Conference Companion*, ACM, New York, pp. 191-192
- Suwa, M and Tversky, B: 1997, What do architects and students perceive in their design sketches?: A protocol analysis, *Design Studies* **18**(4): 385-403
- Takeda, H, Veerkamp, P, Tomiyama, T and Yoshikawa, H: 1990, Modeling design processes, *AI Magazine*, **11**(4): 37-48.
- Taura, T, Yoshimi, T and Ikai, T: 2002, Study of gazing points in design situation - a proposal and practice of an analytical method based on the explanation of design activities, *Design Studies* **23**(2): 165-185.
- Tutiya ,T: 1993, *Is the Science of Mind Possible?*, University of Tokyo Press, Tokyo.
- Tversky, A and Kahneman, D: 1981, The framing of decisions and psychology of choice, *Science*, **211**: 453-458
- Ulrich, KT and Eppinger, SD: 1995, *Industrial Design, Product Design and Development*, McGraw-Hill, New York.
- Weisberg, RW: 1992, *Creative Problem Solving, Creativity-Beyond the Myth of Genius*, WH.Freeman and Company, New York.
- Yoshimi, T and Taura, T, A: 1999, Computational model of a viewpoint-forming process in a hierarchical classifier system, *Genetic Programming 1999: Proceedings of the Fourth Annual Genetic Programming Conference*, Morgan Kaufmann, San Fransisco, pp. 758-766.

## AN EVOLUTIONARY FRAMEWORK FOR ENHANCING DESIGN

*A kernel of computational systems for enhancing design with dynamic structure of hierarchical representations*

KWAI HUNG CHAN, JOHN HAMILTON FRAZER AND MING-XI  
TANG  
*The Hong Kong Polytechnic University  
China*

**Abstract.** A computational framework for enhancing design in an evolutionary approach with a dynamic hierarchical structure is presented in this paper. This framework can be used as an evolutionary kernel for building computer-supported design systems. It provides computational components for generating, adapting and exploring alternative design solutions at multiple levels of abstraction with hierarchically structured design representations. In this paper, preliminary experimental results of using this framework in several design applications are presented.

### 1. Introduction

Evolutionary computation techniques are traditionally used for solving optimisation and searching problems, while their adaptive and explorative abilities for generate potentially large numbers of design solutions are less utilised in design support systems. These techniques have attracted much attention in recent years, and evolutionary design has emerged as an area of interests for design research.

When solving engineering problems with computer-supported systems, potential solutions are formulated in appropriate representations in advance. However, design problems do not have absolute concrete representations, especially in the early stages of the design process. Design involves dynamically changing activities throughout the whole process, and the potential large numbers of design solutions are often explored by designers at various different abstractive levels of representation. Conventional evolutionary techniques do not provide a flexible structure to handle such

nature of design, which is particularly important in conceptual and creative design.

This paper presents our first attempt of developing a computational framework that supports design process at various abstractions in an evolutionary approach. In the coming section 2, an introduction is firstly given to explain the nature of design and the power of evolutionary computation techniques in design-supporting systems. This section also discusses the issues of evolutionary design and hierarchical representations in design. Section 3 presents the concept of the proposed evolutionary framework that supports evolutionary design in a hierarchical representation structure. Section 4 then introduces the experimental results of applying such an evolutionary frame to several design problems, and a conclusion is given in the final section.

## 2. Evolutionary Design and Design Representations

Two aspects of design are concerned in this paper: 1) the hierarchical abstractions of design representations, and 2) the evolutionary nature of design process. The coming subsections discuss how these two issues are related to developing computer systems for supporting design, and also how they lead to our attempt of developing the proposed framework.

### 2.1 HIERARCHICAL REPRESENTATIONS OF DESIGN

Design solution and design process have been modelled in various ways in different design contexts. One commonly used model is in a hierarchical form. While many hierarchical models used in computational systems exhibit in a way of decomposition-and-composition modelling of design solutions, hierarchical models of multiple representations provide much more flexible and efficient structure for supporting design process, in particular at conceptual stages.

Design has been related to network, layer-network or hierarchy in many studies (Rosenman and Gero 1999; Suh 1990; Tomiyama 1995). There is a consistent preference for a hierarchical approach to modelling design. Many of these models work on one representation domain, in the final representation of design output. Despite of the fact that design solution is the ultimate design product to be implemented, much more emphasises should be placed at various abstractive representations of the design problem and solution, which can often lead to creative and innovative ideas.

The importance of realizing the hierarchical abstraction levels of multiple representations during design process has been discussed in recent studies. In particular, when attention is put on the creativity of design,

“creative leap” (Cross 1997) or “sudden mental insight” (Akin and Akin 1996, 1998), which emphasize the mapping from one design representation to another, becomes an important factor for successful creative design.

Some other research work were concerned with the application of this multiple representation approach in developing computational model for mechanical design (Heissner et al. 2000; Liu et al. 2000; Peak et al. 1998) or architectural design (van Leeuwen and Wagter 1998). However, many adopted a static hierarchical structure, which restricts the flexibility of manipulating multiple representations of design in a dynamical way.

## 2.2 EVOLUTIONARY DESIGN

During the process of design, exploration of possible design alternatives and adaptation of beneficial features of them are required for a successful design. Evolutionary techniques have a great potential in supporting such design exploration and adaptation.

Traditional Evolutionary Computation (EC) techniques are based on mimicking the natural evolutionary process for survival. They conventionally include Evolutionary Algorithms (EA), Evolutionary Strategies (ES), Genetic Algorithms (GA) and Genetic Program (GP). There are many articles and materials introducing the working principles and application of EC (Back 1996; Eiben 1996; Fogel 1995; Michalewicz et al. 1996).

Evolutionary techniques have been applied to solve searching and optimisation problems in various engineering fields. In recent years, some research work have applied evolutionary computation techniques to artistic, graphics and form design (Frazer 1995; Rowbottom 1999; Sims 1991; Todd and Latham 1999; Witbrock and Neil-Reilly 1999). Other studies concentrated on studying different evolutionary methods for exploring design alternatives in various fields (Gero et al. 1997; Graf 1995; Poon and Maher 1996). Many new evolutionary design methods have been developed (Bentley 1999).

## 2.3 EVOLUTIONARY DESIGN IN A HIERARCHICAL WAY

Hierarchical evolutionary techniques for design have been proposed by some researchers (Garza and Maher 2000; Rosenman 1996; Rosenman and Gero 1999). Analogous to the common application of hierarchical concept in modelling objects, these hierarchical evolutionary techniques concentrated on the decomposition modelling of evolutionary entity working on the final solution domain, in order to make the design exploration more efficient.

The proposed framework in this paper is application-independent so that different design tasks can be supported by computational systems using the framework as its dynamic kernel. We target on developing such a computational framework that supports: 1) evolutionary design process involving various different representations, and 2) manipulation of design at various abstractions with a dynamic hierarchical structure of multiple representations.

### 3. An Evolutionary Hierarchical Framework for Enhancing Design

The idea of this framework was firstly raised in Chan et al. (2000). The currently implemented framework (with evolutionary structure) provides dynamic rules (evolutionary mechanisms) that can be used as the kernel of evolutionary computational systems, i.e., the systems that act evolutionarily.

The following subsections present the basic structure of such a framework, the important modules in this framework, and a more formal definition of the framework in mathematical terms. This section also discusses the important role of designers and users played in the computational systems with this framework kernel.

#### 3.1 BASIC STRUCTURE OF THE FRAMEWORK

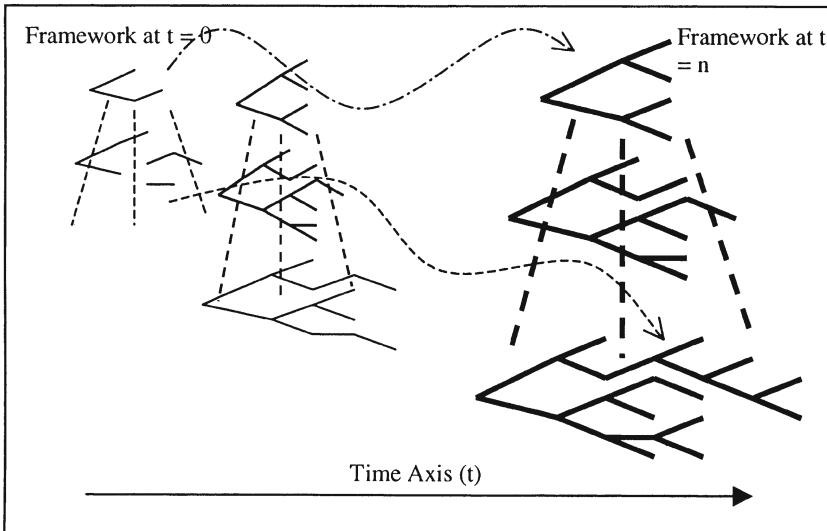
Our framework is structured as a set of evolutionary elements connected in a hierarchical way. Each of these evolutionary elements has its attributes representing design objects, or parts, at specific levels of abstraction. It is also attached with a set of evolutionary mechanisms. The set of evolutionary elements in one level represent the design solution(s) at a specific abstract representation.

##### 3.1.1 Dynamic Hierarchical Structure at Multiple Representations

Figure 1 shows the dynamic structure of the representations during the evolutionary process, and how it is developed from a simple form to a complex one in a design application. The static representation at a time frame represents a set of design solutions, situated at different layers of representation abstraction. Each layer in the static representation represents the set of connected evolutionary elements (the vertices in the figure) that construct the design solution(s) at that representation domain.

Be noted that more than one solution may situate in one representation level, and the tree structure of each layer in the figure can be realized as one or more design representations structured in that representation level. As the design evolves dynamically, the structure of the whole representation as well as that of the connect elements in each layer change.

When the framework is applied to solving design tasks, the evolutionary elements in an upper layer represent problem, solution or sub-solutions in a much more abstractive conceptual level. Those evolutionary elements in lower layers represent in much more concrete formats closer to the final design output domain. For example, the upper layer may represent textual specifications while the lower one may represent 3D models.



*Figure 1.* Dynamics of the proposed evolutionary framework in the temporal axis.

### *3.1.2 Fractal or self-symmetric structure of the framework*

While there are different evolutionary mechanisms attached to various evolutionary elements, each evolutionary element situated in any one of the representation layers of the hierarchy has the same structure. The evolutionary element may be realized with the concept of the fractal structure, similar to the one proposed by Hintersteiner (1999).

Analogous to the fractal relationship of many natural systems in nature, each evolutionary element has its specific evolutionary mechanism(s), which influences the evolution of the element itself. This evolutionary mechanism controls the evolution of the element and may also influence other elements in its connected layers. This hierarchical relationship can be extended further to even deeper layers when needed.

With the structure of this evolutionary framework, there are two main problems to be tackled. The first one is the right evolutionary mechanisms

that link to each element. The second is the appropriate representation and interpretation of design solution at various abstraction levels.

### 3.2 EVOLUTIONARY MECHANISMS

Evolutionary mechanisms are essential parts to make our computational framework changing dynamically, and evolutionarily. They have to offer two basic functions in the framework: 1) to evolve the evolutionary elements to which they are attached, and 2) to influence other evolutionary elements at adjacent layers in the hierarchical structure when needed. Therefore, changing design objects at one abstraction level may lead to the corresponding modification of those at the adjacent abstraction levels.

In this study, evolutionary mechanisms are not restricted in a narrowed-Darwinism sense. They are not only limited in the category of conventional Evolutionary Computation techniques. Any module can be an evolutionary mechanism in our framework as long as it offers a dynamic mechanism to evolve design objects, based upon specific dynamic environment towards a specific tendency or preference. Designers, users, expert systems, Genetic Algorithms are potential evolutionary mechanisms. In fact, these mechanisms may be described as entities, or agents, that run autonomously, and preferably collaboratively, under a dynamic environment.

### 3.3 ABSTRACTION AND INTERPRETATION OF DESIGN TASKS

In the framework, all data are meaningless without correct representation and interpretation of them. It is important to have the right representation domains to be manipulated at. Thus with specific applications, specific representation or interpretation functions are needed to map the framework data to some meaningful and interpretable forms to the outside world, to designers, and users as shown in Figure 2.

The proposed computational framework is application-independent. How this framework is applied to specific design tasks relies on the correct interpretation of the data in our framework to human-understandable information. Thus there is an interpretation middle-layer that links our framework data to the external outside world – human interaction domain.

### 3.4 FORMAL DEFINITION OF THE FRAMEWORK

In the previous subsections we have introduced the framework in a descriptive way, and a general concept of the framework has been presented. While it is not meant to give a rigorous mathematical derivation and proof of the framework, this subsection attempts to present the framework in a more formal way in order to provide a better overview of the whole framework in a concise and systematic way.

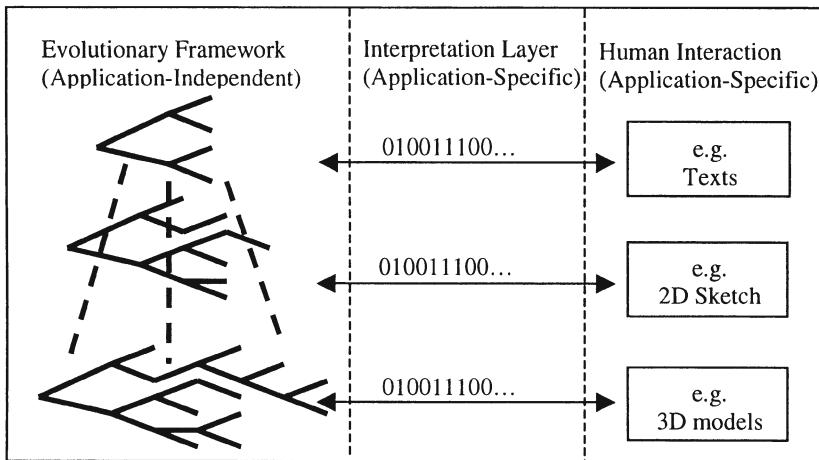


Figure 2. A computational system using the framework with human interaction.

### 3.4.1 Representation of design objects

We start from the primitive,  $p$ , the most fundamental “matter” in our framework. The primitive belongs to a numerical set, such as real number or integer. It can be better related to the primitive data type in software systems, when it is used as a kernel of computer systems. (Be noted that the “double” data type is used for the primitive in our initial implementation of the experimental work to be discussed in section 4.)

We then define a design object in our framework,  $d$ , that exists in the domain of the set of all potential design objects,  $D$ . It can be: 1) a simple primitive,  $p$ , 2) a finite sequence of primitives  $\langle p_1, p_2, p_3, \dots \rangle$  or 3) a set of other design objects,  $d \subseteq D$ .

Be noted that a design object  $d$  in our framework alone is application-independent. It is only meaningful under specific representation domain that can be meaningfully interpreted by a group of people in certain context, such as designers or users.

In a computational system using the framework with a specific representation domain,  $r$ , there must exist at least one function,  $f_r$ , that maps  $d \in D$  to a domain for specific application, which is interpretable to human users or designers. For example: a design object in our framework can be mapped to a geometrical model,  $g \in r$ , with the representation function,  $f_r$ , and thus,  $f_r(d) = g$ .

In most cases we will often presume there exists a final representation domain (the least abstract representation), that the design at this

representation level can lead to a direct implementation or execution to obtain a real solution of the problem to be solved. For example, we may presume the final design output will be a 3D model in 3D geometric representation space in designing mechanical parts.

### *3.4.2 Evolutionary elements and evolutionary mechanisms*

The basic element of the framework, the evolutionary element  $e$ , is a triple,  $(d, em, ee)$ , which consists of a design object  $d$ , a set of evolutionary mechanisms  $em$ , and a set of “linked” evolutionary elements  $ee$ .

A dynamic mechanism,  $m$ , can be realized as a function that maps a design object,  $d$ , to another design object,  $d'$ , and thus  $m(d) = d'$ . An evolutionary mechanism,  $em$ , is an extension of such a dynamic mechanism that *evolves* an evolutionary element,  $e$ . The evolutionary mechanism not only changes the design object of  $e$ , but also influences its evolutionary mechanisms of  $em$  and linked evolutionary elements  $ee$ . This influence may even further alter the evolutionary mechanisms of the linked evolutionary elements. In general,  $em(e) = e'$  or  $em(d, em, ee) = (d', em', ee')$ .

The evolution of one evolutionary element,  $e_k$ , at layer  $k$ , should then lead to updating of or influencing those elements in the connected layers  $k-1$  and  $k+1$  in our bi-directional hierarchical structure of framework, although in practical implementations we tend to simplify it to a directional influence in a top-down manner. Be noted that  $em$  is not necessarily to be one-to-one mapping. One evolutionary element may be evolved to one or more objects by different evolutionary mechanisms

### *3.4.3 The static version of the evolutionary hierarchical framework*

To better understand the proposed evolutionary hierarchical framework, we start with a static version of the framework. The static version of the evolutionary hierarchical framework,  $sF$ , is a sequence of evolutionary element sets,

$$< ee_1, ee_2, ee_3, \dots, ee_n >$$

$ee_k$  is a set of evolutionary elements situated in the  $k$  layer of the framework such that, when the framework is applied to a specific design application, there exists a set of corresponding representation layers ( $r_1, r_2, r_3, \dots, r_n$ ). For each design object,  $d$ , in every evolutionary element of every layer of the static framework, there exists a representation function,  $f_r$ , that maps  $d$  to its interpretable form to human at a specific representation domain,  $r \in R$ , and thus  $f_r(d) \in r$ .

### *3.4.4 The general dynamic version*

Finally our evolutionary framework,  $eF$ , is an infinite sequence of static frameworks,  $\langle sF_1, sF_2, sF_3, \dots \rangle$ , such that to each static framework,  $sF_t$ , at any specific time frame,  $t$ , the invocation of one “evolution action” function,  $f_e$ , to evolutionary mechanisms in the framework produces the next static framework,  $sF_{t+1}$ , in time frame  $t+1$ . Therefore, each evolutionary mechanism maps the attaching evolutionary element(s) to new one(s) and the overall framework evolves,  $f_e(sF_t) = sF_{t+1}$ . This may lead to one of the following consequences:

*Same hierarchical structure*: the framework in next time frame,  $sF_{t+1}$ , is the same as the current one,  $sF_t$ . Therefore, the number of layers in the hierarchy and the number of evolutionary elements in each layer are the same in both static frameworks,

*Modified hierarchical structure*: similar to the same hierarchical structure, but the number of evolutionary elements in some layers may be different between two static frameworks,

*New hierarchical structure*: both the number of layers in the hierarchy and the number of evolutionary elements in some layers may be different between two static frameworks.

The following lists the summary of the formal representation of the framework:

- Evolutionary Hierarchical Framework ( $eF$ ): a sequence of static frameworks

$$eF = \langle sF_1, sF_2, sF_3, \dots \rangle$$

- Evolutionary Action Invocation ( $f_e$ ): a mapping function of static frameworks

$$f_e(sF_t) = sF_{t+1}$$

- Static Framework ( $sF$ ): a sequence of sets of evolutionary elements

$$sF = \langle ee_1, ee_2, ee_3, \dots, ee_n \rangle$$

- Evolutionary Element ( $e$ ): a triple of (design object, a set of evolutionary mechanisms, and a set of evolutionary elements)

$$e = (d, em, ee)$$

- Evolutionary Mechanism ( $em$ ): a mapping (evolutionary) function of evolutionary elements

$$em(d, em, ee) = (d', em', ee')$$

- Design Object ( $d$ ): 1) a simple primitive, 2) a finite sequence of primitives, or 3) a set of other design objects

$$1) p, 2) \langle p_1, p_2, p_3, \dots \rangle, \text{ or } 3) d \subseteq D.$$

- Primitive ( $p$ ): the most primitive in the framework, that belongs to a numerical set.

- Representation ( $r$ ): a specific domain that can be meaningfully interpreted by a group of people, such as designers or users,  
 $r \in R$ , where  $R$  represents the set of all potential meaningful representation domains.
- Representation Function ( $f_r$ ): a function that maps a design object,  $d$ , to an interpreted object,  $o_r$ , in a specific representation domain,  $r$ .

$$f_r(d) = o_r, o_r \in r$$

### 3.5 INTERACTIVE EVOLUTIONARY HIERARCHICAL FRAMEWORK

To implement a fully autonomous model of such a framework is far from realistic. Designers still play a crucial role in interacting with the system to make decisions, evaluate various qualitative factors, and explore design solutions with their own styles. In fact, designers may well be the best engines to work with our proposed framework.

Even with the advanced computational design systems, designers still play an important role in interactive systems in many studies (Campbell et al. 1998; Parmee et al. 2000). Designers and users are the important part in our practical framework. There are four different ways that designers and users can interact with the framework: 1) direct manipulation at every details of design objects in evolutionary elements, a super evolutionary mechanism, 2) manipulation of parameters in computational evolutionary mechanisms, to control the way that these evolutionary mechanisms act on the elements, 3) manipulation of design seeds to be evolved by the preset computational evolutionary mechanisms, and even 4) alternation and manipulation of computational evolutionary mechanisms.

During the design process from its early stages (generally-known as conceptual stages) to later stages (the detailed stages), designers can work on and switch across different levels of representation without any sequential restriction, although the activities in the abstracted representation levels generally tend to dominate in the early conceptual stages while those in less abstracted ones increase when the design process is reaching the final detailed stage.

## 4. Some Preliminary Results

This section briefly introduces some preliminary results obtained in the application of the proposed framework to some experimental work. They include 1) a demonstration of dynamic evolutionary structure of the hierarchical framework, 2) a study of evolutionary behaviour of two evolutionary mechanisms in the hierarchical structure, and 3) an experimental application of wine-glass generation.

#### 4.1 EVOLUTIONARY STRUCTURE OF THE FRAMEWORK

Figure 3 shows a simple example to demonstrate the dynamic nature of the hierarchical structure of the framework. The framework in this work starts with one “root” evolutionary element (at the top of the hierarchy), attached with a self-replicating evolutionary mechanism. When this evolutionary mechanism is activated, it alters the colour attribute of the element as well as reproduces some offspring in the next representation layer in a random manner. At each time frame, the evolutionary mechanism of each evolutionary element in the framework is invoked and the process continues from an early generation (the top image) to later ones (the bottom). The structure of the framework is then dynamically changed as shown, with one single type of simple evolutionary mechanism. In this simple example, all levels of representation are represented and interpreted upon the same representation domain, a 2-D graphical domain.

When this example is related to the formal representation of the framework, each node in the tree diagram will be an evolutionary element. The design object (attribute) of each evolutionary element contains only one primitive, a data type “double”. There is only one evolutionary mechanism (the self-replicating mechanism) attached to the element. The representation function then maps the design objects to their corresponding colours, and presents the framework in a 2D graphical form.

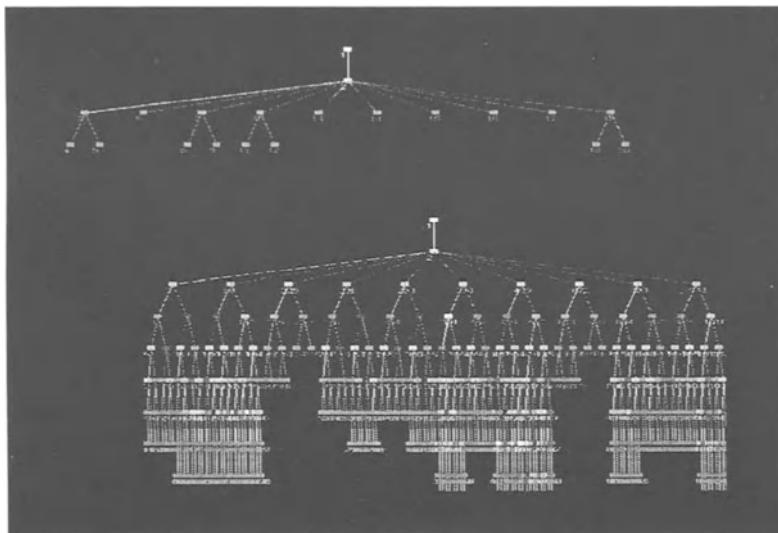


Figure 3. 2D graphical hierarchies generated with a simple self-replication element

At time  $t = 0$ , the static framework  $sF_0$  consists of only one evolutionary element, the  $root = (colour, self\text{-}rep, empty)$ . The evolutionary mechanism ( $self\text{-}rep$ ) evolves the root, and thus changes the root colour and self-replicates some “offspring” in certain random probability. The new offspring will also be attached with the same self-replicating mechanism.

$$self\text{-}rep(root) = root' = (new\text{-}colour, self\text{-}rep, offspring)$$

In the next time frame, the self-replication process proceeds to reproduce some offspring of the root’s offspring. After some generations, a more complex hierarchical structure,  $sF_k$ , is produced (as those shown in the Figure 4). In this case, the only self-replicating evolutionary mechanism,  $self\text{-}rep$ , evolves an element in two ways:

- To elements without offspring, new a random number of offspring.  $self\text{-}rep(colour, self\text{-}rep, empty) = (new\text{-}colour, self\text{-}rep, offspring)$
- To elements with offspring, same offspring are retained.  $self\text{-}rep(colour, self\text{-}rep, offspring) = (new\text{-}colour, self\text{-}rep, offspring)$

#### 4.2 EVOLUTIONARY BEHAVIOR OF THE FRAMEWORK

To study the evolutionary behaviour of evolutionary mechanisms in a hierarchical structure of the framework, an example has been developed. This example shows how inter-evolutionary interaction takes place between two different evolutionary mechanisms. Genetic Algorithms (GA) and Cellular Automata (CA), interact and achieve the goal of seeking preference patterns with our framework.

In engineering sense, Genetic Algorithms can be realized as a stochastic searching approach, which seeks optimal solution(s) from a pool of possible candidates (the population). From the initial population, which is often generated randomly, the individuals of the population evolve through the cyclic process of evaluation, selection, crossover and mutation from one generation to another.

Cellular Automata (CA) is a specific computational model with simple self-organizing mechanism. Much attention is particularly given to its simple self-organizing mechanism applied locally, which can seemingly produce complex global behaviours or patterns. The cells (elements) of CA behave in a self-organizing manner with their neighbours according to a transition rule.

Figure 4 shows some patterns generated with a simple CA system implemented with the computational framework. An ordinary one-dimensional CA can be realized as one specific evolutionary framework, which has only one evolutionary element ( $ee$ ) attached to an evolutionary

mechanism, the CA Transition Rules (*caTR*). Thus, the static framework at time  $k$ ,  $sF_k$ , consists of only one element, *ee*, and

$$\begin{aligned} ee &= (\text{cell-States}, \text{caTR}, \text{empty}) \\ \text{caTR}(ee) &= ee' = (\text{new-cell-States}, \text{caTR}, \text{empty}) \end{aligned}$$

In this case of using the evolutionary framework for producing an ordinary CA, Figure 4, the evolutionary mechanism (CA transition rules) only changes the design objects (the states of the cells) without altering the mechanism and the offspring (empty in this case). Conversely in the GA-governing-CA examples to be discussed next, GA changes the evolutionary mechanisms (the transition rules) of its offspring (CA).

This example of GA-governing-CA can be related to the studies of applying GA to CA for solving the problems of density classification and synchronization (Das et al., 1995). Instead of finding a universal rule that can be applied to every random seed for obtaining a final goal or pattern, our main goal is to find the right transition rules that can be applied to the right seeds to produce the right patterns. The global environment to produce the right pattern is governed by the GA. This GA controls the transition rule of each CA instead of the final pattern, while the transition rule of CA influences the final pattern.

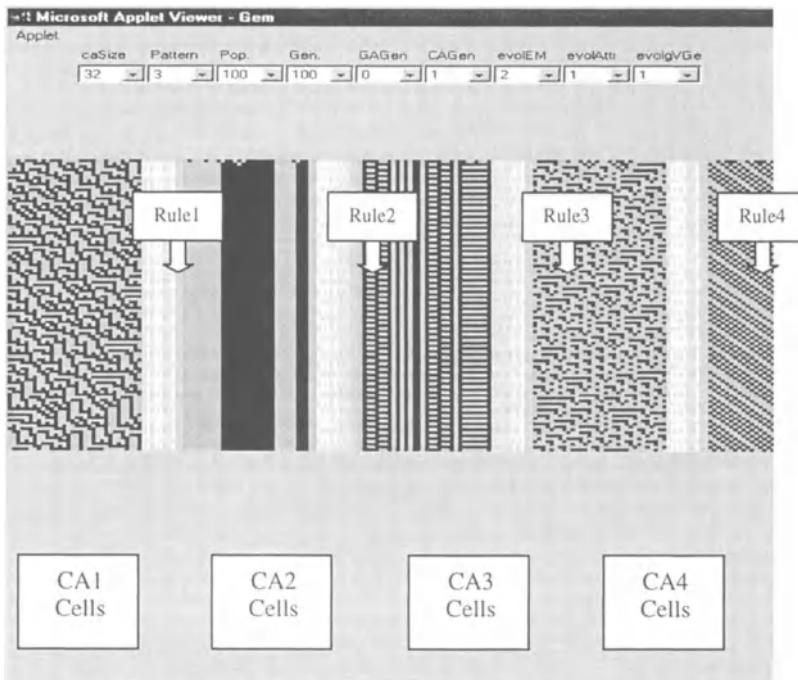
The framework of this example can be realized in a 2-layer hierarchical form, which evolves according to the need. The upper layer can be treated as an external environment, and is controlled by an evolutionary mechanism of Genetic Algorithm (GA), that governs the behaviour of an evolutionary mechanism of 1-dimensional Cellular Automata (CA) cells in the lower layer. Therefore, the 2-layer framework has its root:

$$\text{root} = (\text{GA-obj-fn}, \text{GA}, \text{CAees})$$

The design object of the root is the objective function of GA, set by the user for seeking desired patterns such as vertical lines or checkerboard. The evolutionary mechanism of the root is a GA that mainly evolves the evolutionary mechanisms (CA transition rules) of its offspring (CAs) based on how well the produced CA pattern (cell states) fits to its objective function, the desired pattern. Each CA *Caee*, the child of GA (the root), has the form similar to the ordinary CA.

$$\text{Caee} = (\text{cell-States}, \text{caTR}, \text{empty})$$

With this framework, the evolution process starts with user's setting the GA-objective-function (say, for vertical line or checkerboard patterns). Each generation of the whole framework then works in the following way:



*Figure 4.* An Ordinary 1-dimensional Cellular Automata implemented with the framework.

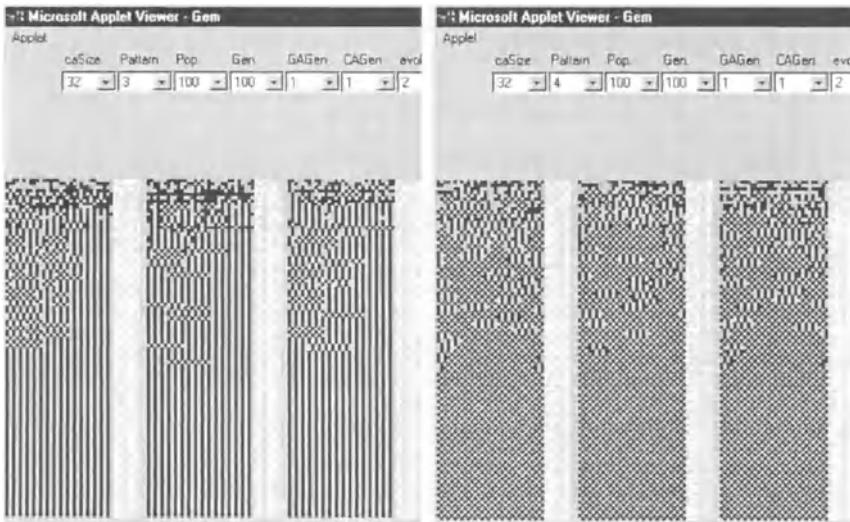
- GA (evolutionary mechanism of root) changes CA.
- CA evolutionary mechanism (transition rules) changes cell states.
- Cell state reflects the fitness value of CA in GA.
- Repeat, until terminated.

Figure 5 shows two results that are generated with different objective functions applied to the GA environment. These objective functions simply match the produced pattern and the desired ones: the patterns of vertical lines (left image) and checkerboard (right image). Under this framework with the evolution of the CA and GA, the right transition rules associated to the right CA seeds produce the right patterns. The results quickly converge to the desired pattern in a few generations.

#### 4.3 A DEMONSTRATIVE APPLICATION: WINE-GLASS GENERATION

To the simple examples as those discussed in the previous subsections, automation may be achieved. However, in practical design tasks with much

higher complexity, even a seemingly simple one such as wine glass generation discussed here, designer's involvement is crucial. The objective of this experimental application in this part is to generate a series of wine glasses with the interaction of designers and users at various representation levels, with the support of the framework.



*Figure 5.* Patterns of vertical lines (left) and checkerboards (right), generated with the GA-governing-CA framework.

The framework is integrated with a commercial CAD tool (MicroStation) and it acts as a system kernel. The layout of the system is shown in Figure 6. The framework associated with this system can be realized as a four-layered hierarchy, the basic feature, manual manipulation of 2D geometric profiles, GA exploration of 2D profiles, and 3D models.

There is only one evolutionary element in each layer, except the layer of GA which has a population of potential elements. In the GA layer a new set of 2D wine-glass profiles are generated in the next generation, which inherit those selected by the user previously. At the moment, only artificial or manual selection is used in this GA process, while more research work are required to study what objective functions are needed and can be formulated for fully-automating this GA process.

The evolutionary framework has a hierarchical form:

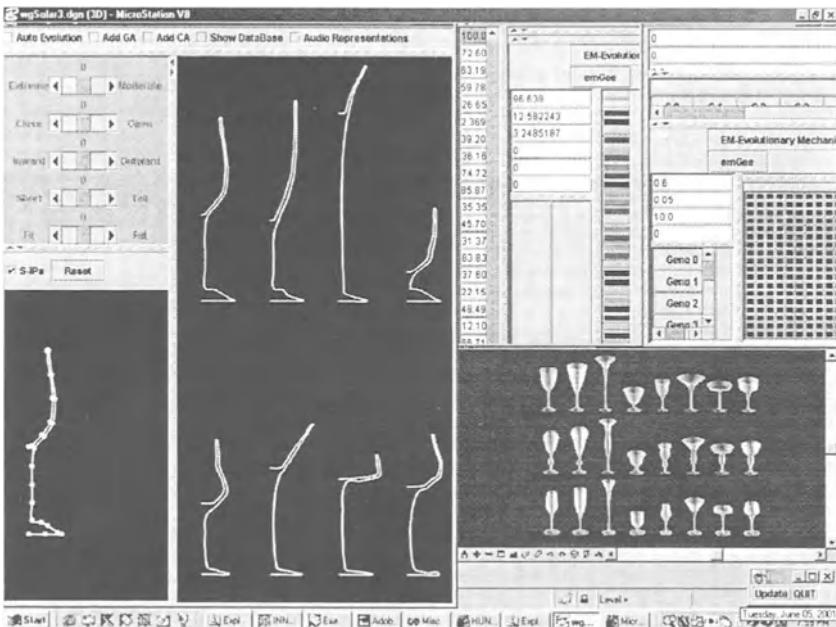
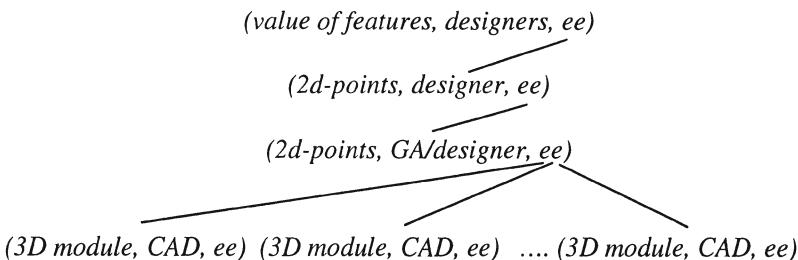


Figure 6. Wine-glass generation system, with the framework as the evolutionary kernel



With this system, designers can manipulate wine-glasses at various abstractions: from the abstract descriptive features of wine-glasses in the top-right window, the 2D profile of the wine-glasses, the add-on evolutionary mechanisms including a Genetic Algorithm, to the final 3D models of the generated series, as shown in Figure 7. A large number of alternative design solutions can then be explored and generated, as shown in Figure 8.

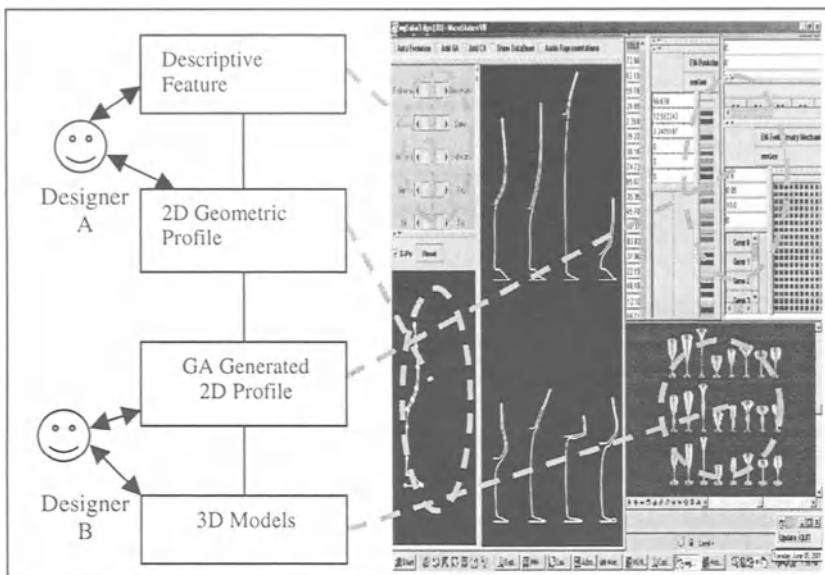


Figure 7. Designer interaction with the system in a hierarchical way

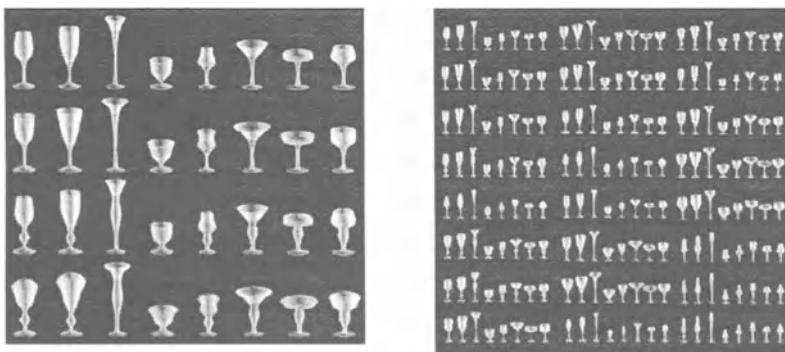


Figure 8. Wine-glasses generated with the system

## 5. Conclusion

In this paper we have presented an evolutionary framework, a kernel of computational systems for supporting design in way of dynamic hierarchy of multiple representations. Some nature of design, the power of evolutionary computation in design-supporting systems, the problem of conventional

techniques for evolutionary design and the importance of hierarchical representations of design are discussed in section 2. Section 3 then introduces the basic concept of the proposed computation framework that supports evolutionary design in a hierarchical representation structure. Some experimental works are also presented (section 4).

The framework is structured with a set of connected evolutionary elements, and each element is attached with some evolutionary mechanisms. Each evolutionary element has its attributes and is allocated at an abstraction level. The element evolves in the model according to its attached evolutionary mechanisms. When this framework is applied to design applications, the set of elements at a layer represents a specific design representation of a solution. With its dynamic structure, the framework offers greater generative capability based on the evolutionary mechanisms of individual evolutionary elements at various representation levels.

The obtained results showed the potential of the framework in providing a much more flexible and dynamic architecture, as well as evolutionary exploration ability. Although the framework provides an infrastructure for evolutionary design with dynamic hierarchical structure of multiple representations, the problems of making an artificially intelligent system with the framework obviously lie with two main issues. The first one is to build suitable abstractions of specific design tasks to be represented and interpreted in such a computational support system for design with the framework. The second is the development of adequate autonomous or semi-autonomous evolutionary mechanisms to specific design tasks.

During our study, one of the most frequently asked questions is “why do we need to design at this abstraction, not that? Who is deciding what abstractions, and what hierarchies of these abstractions are for a design task?” In an ideal case, the framework can evolve automatically in a way in which suitable hierarchies and abstractions of representation can be dynamically generated, throughout the design process of a design domain. However in a more practical sense, if we develop a static framework in a manageable level, it is the tool developer and the user (the designer), who pre-determine the right structure of the hierarchy and the corresponding abstractions of representation.

Even working with such a static framework, there are also problems of developing a hierarchy with appropriate abstractions of representation. There are different abstractions of representation to be used in various design fields, based on different designing patterns of designers. Furthermore, different levels of representation abstractions may be used in different design tasks even with one designer. To make our framework

function for a wider scope of design, more studies are needed in order to investigate suitable abstractions of representation and their related structures.

While this issue is closely related to design psychology and design cognitive modelling which do not fall in our main research scope, to the later issue we have implemented two computational evolutionary mechanisms, Genetic Algorithm and Cellular Automata, to be used in our experimental works. More studies are required to work on what other potential autonomous evolutionary mechanisms can be used. Furthermore, two specific aspects in evolutionary behaviour of complex systems are concerned, the reductionism (a deductive approach) and the emergent behaviour (an inductive approach).

While our current study mainly concentrates on the top-down evolutionary behaviour, from a much abstract layer to a less one, attention is also paid at studying potentials of computational modules with the bottom-up behaviour, that provide a practical contribution to the framework for supporting design. In this aspect, we expect some form of emergent patterns, features, intelligence or knowledge that could be captured in lower levels of the framework (at less abstract representation), which may emerge to some forms of higher abstract representation of such patterns.

Further investigation is also required to study the potential application of our framework in other design areas. Currently the applications of the framework to other design areas are being studied. In particular, its application to sound pattern generation and Chinese font design are being investigated. Another issue being studied is related to the designer interface, which is crucial in supporting design with the framework. Much work has to be done to improve the designer-computer interaction with the framework, and some studies are also being undertaken in this aspect.

### Acknowledgements

This research is supported by the Hong Kong Polytechnic University, under the RGC grant A.05.4, the project accounts B-Q271 and G.S9.00. We gratefully acknowledge the help and advice from the members of the Design Technology Research Centre in the School of Design of the Hong Kong Polytechnic University.

### References

- Akin, O and Akin, C: 1996, Frames of reference in architectural design: analyzing the hyperacclamation (A-h-a!), *Design Studies* 17: 341-361.
- Akin, O and Akin, C: 1998, On the process of creativity in puzzles, inventions and designs, *Automation in Construction* 7: 123-138.

- Back, T: 1996, *Evolutionary Algorithms in Theory and Practice*, Oxford University Press, Oxford.
- Bentley, PJ (ed.): 1999, *Evolutionary Design by Computers*, Morgan Kaufmann, San Francisco, CA.
- Campbell, M, Cagan, J and Kotovsky, K: 1998, A-Design: Theory and implementation of an adaptive agent-based method of conceptual design, in Gero, JS and Sudweeks, F (eds), *Artificial Intelligence in Design '98*, Kluwer, Dordrecht, pp. 579-598.
- Chan, KH, Frazer, J and Tang, MX: 2000, Handling the evolution and hierarchy nature of designing in computer-based design support systems, *Proceedings of the Third International Conference Computer-Aided Industrial Design and Conceptual Design (CAID & CD '2000)*, International Academic Publishers, Hong Kong, pp. 447-454.
- Cross, N: 1997, Descriptive models of creative design: application to an example, *Design Studies* **18**: 427-455.
- Das, R, Crutchfield, JP, Mitchell, M and Hanson, JE: 1995, Evolving globally synchronized cellular automata, in LJ Eshelman (ed.), *Proceedings of the Sixth International Conference on Genetic Algorithms*, Morgan Kaufmann, San Francisco, pp. 336-343.
- Eiben, AE: 1996, Evolutionary exploration of search spaces, in Z Ras and M Michalewicz (eds), *Proceedings of Foundations of Intelligent Systems: 9th International Symposium, ISMIS '96*, Springer, New York, pp. 178-188.
- Fogel, DB: 1995, *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*, IEEE Press, New York.
- Frazer, J: 1995, *An Evolutionary Architecture*, Architecture Association, London.
- Garza, AGDS and Maher, ML: 2000, A process model for evolutionary design case adaptation, in JS Gero (ed.), *Artificial Intelligence in Design '00*, Kluwer, Dordrecht, pp. 393-412.
- Gero, JS, Kazakov, VA and Schnier, T: 1997, Genetic engineering and design problems, in D Dasgupta and Z Michalewicz (eds), *Evolutionary Algorithms in Engineering Applications*, Springer, Berlin, pp. 47-69.
- Graf, J: 1995, Interactive evolutionary algorithms in design, *International Conference on Artificial Neural Networks and Genetic Algorithms ICANNGA '95*, Ecole des Mines d'Al'es, France, pp. 227-230.
- Heisserman, J, Callahan, S and Mattikalli, R: 2000, A design representation to support automated design generation, in JS Gero (ed.), *Artificial Intelligence in Design '00*, Kluwer, Dordrecht, pp. 545-566.
- Hintersteiner, JD: 1999, A fractal representation for systems, in H Kals and FV Houten (eds), *Integration of process knowledge into design support systems: Proceedings of the 1999 CIRP International Design Seminar*, Kluwer, Boston, MA, pp. 427-436.
- Liu, Y, Chrabarti, A and Bligh, T: 2000, A computational framework for concept generation and exploration in mechanical design, in JS Gero (ed.), *Artificial Intelligence in Design '00*, Kluwer, Dordrecht, pp. 499-519.
- Michalewicz, Z, Xiao, J and Trojanowski, K: 1996, Evolutionary computation: one project, many directions, in Z Ras and M Michalewicz (eds), *Proceedings of Foundations of Intelligent Systems: 9th International Symposium, ISMIS '96*, Springer, New York, pp. 189-201.
- Parmee, I, Cvetkovic, D and Bonham, C: 2000, Interactive evolutionary conceptual design systems, in JS Gero (ed.), *Artificial Intelligence in Design'00*, Kluwer, Dordrecht, pp. 249-268.
- Peak, RS, Fulton, RE, Nishigaki, I and Okamoto, N: 1998, Integrating engineering data and analysis using a multi-representation approach, *Engineering with Computers* **14**: 93-114.

- Poon, J and Maher, ML: 1996, Emergent behaviour in co-evolutionary design, *in* JS Gero and F Sudweeks (eds.), *Artificial Intelligence in Design '96*, Kluwer, Dordrecht, pp. 703-722.
- Rosenman, MA: 1996, The generation of form using an evolutionary approach, *in* JS Gero and F Sudweeks (eds), *Artificial Intelligence in Design '96*, Kluwer, Dordrecht, pp. 643-662.
- Rosenman, M and Gero, JS: 1999, Evolving designs by generating useful complex gene structures, *in* PJ Bentley (ed.), *Evolutionary Design by Computers*, Morgan Kaufmann, San Francisco, pp. 345-364.
- Rowbottom, A: 1999, Evolutionary art and form, *in* PJ Bentley (ed.), *Evolutionary Design by Computers*, Morgan Kaufmann, San Francisco, pp. 261-277.
- Sims, K: 1991, Artificial evolution for computer graphics, *Computer Graphics* **25**(4): 319-328.
- Suh, NP: 1990, *The Principles of Design*, Oxford University Press, New York.
- Todd, S and Latham, W: 1999, The mutation and growth of art by computers, *in* PJ Bentley (ed.), *Evolutionary Design by Computers*, Morgan Kaufmann, San Francisco, pp. 221-250.
- Tomiyama, T: 1995, A design process model that unifies general design theory and empirical findings, *Proceedings of the 1995 Design Engineering Technical Conferences*, DE-Vol. 83, ASME, New York, pp. 329-340.
- van Leeuwen, J and Wagter, H: 1998, A features framework for architectural information: Dynamic models for design, *in* JS Gero and F Sudweeks (eds), *Artificial Intelligence in Design '98*, Kluwer, Dordrecht, pp. 461-480.
- Witbrock, M and Neil-Reilly, S: 1999, Evolving genetic art, *in* PJ Bentley, (ed.), *Evolutionary Design by Computers*, Morgan Kaufmann, San Francisco, pp. 251-260.

## **KNOWLEDGE SUPPORT FOR DESIGN**

---

*Knowledge support for customer-based design for mass customization*  
Xuan Fang Zha and Wen F Lu

*Elucidating the design requirement for conventional and  
automated conceptual design*  
Mansur Darlington and Stephen J Culley

*Case-based design facilitated by the design exemplar*  
Joshua J Summers, Zoe Lacroix and Jami J Shah

## KNOWLEDGE SUPPORT FOR CUSTOMER-BASED DESIGN FOR MASS CUSTOMIZATION

XUANFANG ZHA AND WEN F LU  
*Gintic Institute of Manufacturing Technology  
Singapore*

**Abstract.** This paper presents a research effort on customer-based design for mass customization using a knowledge support paradigm. The background and prior research work related to customer-based design for mass customization (CDFMC) is first reviewed. Then, the fundamental issues underlying knowledge support for CDFMC are discussed. A knowledge support framework and its relevant technologies are developed for implementing module based product family design for mass customization, which include knowledge modeling and support for customer requirements' modeling, product architecture modeling, product platform and family generation, and product assessment for customization. The issues and requirements related to the development of knowledge intensive support system for modular product family design for mass customization are addressed. Finally, a case study on knowledge support for power supply family design for customization is provided for illustration.

### 1. Introduction

Mass customization embarks a new paradigm for manufacturing industries (Pin II 1993), focusing on variety and customization through flexibility and quick responsiveness with the goal of developing, producing, marketing, and delivering affordable products that can satisfy as wide a range of customers as possible (Tseng and Jiao 1996; 1998). To adopt the mass customization paradigm, many companies are being faced with the challenge of providing as much variety as possible for the market with as little variety as possible between products in order to maintain economies of scale while satisfying a wide range of customer requirements. Product family is a group of related products that share common variables, features, components, and

subsystems and satisfy a variety of market niches. Family-based product design has been recognized as an efficient and effective means to realize sufficient product variety in support for mass customization. Meanwhile, contemporary design process becomes increasingly knowledge-intensive and collaborative. Knowledge-intensive support becomes critical in design process and is being recognized as a key solution towards future competitive advantages in product development. To improve the product family design, it is imperative to provide knowledge support in the family design process and share design knowledge among distributed designers. The driving force behind this research is to develop a knowledge intensive support methodology and framework based on the modular product design paradigm to efficiently and effectively model and synthesize a family of products which can provide increased product variety necessary for today's market. The motivation and vision presented in this research share a similar theme of design for mass customization with (Tseng and Jiao 1996, 1998) but emphasize the customer-based modular design with knowledge support paradigm at the early design stage.

The aim of this paper is to develop knowledge support methodologies and technologies for customer-based design for mass customization. In this paper, an integrated knowledge support approach to platform-based modular product family design for mass customization is discussed in detail. The implementation and verification of knowledge support system are also addressed. The organization of the paper is as follows. Section 2 reviews the background research related to product family design for mass customization. Sections 3 and 4 discuss the technology base for CDFMC. Section 5 proposes a knowledge support framework and scheme for CDFMC. Section 6 deals with knowledge modeling and knowledge based design support process for CDFMC. Section 7 addresses issues and implementation architecture for developing a knowledge intensive support system for CDFMC. Section 8 provides a case study on knowledge support for power supply family design and customization. Section 9 summarizes the paper and points out the future work.

## 2. Literature Review

In this section, the background research related to product family design for mass customization is briefly reviewed. The approaches and strategies for designing families of products and mass customized goods are prevalent in the literatures. They are extensively spread in either operations research

(Gaithen 1980), or computer science (Nutt 1992), or marketing or management science (Kotler 1989; Meyer et al 1993; Pine II 1993), or engineering design (Fujita et al 1997; Simpson et al 1998, 2001; Ulrich et al 1995).

The existing modeling schemes for product families vary in the literatures but two types of representational models are essential: product family architecture and product family evolution. There are three kinds of approaches widely used for representing architecture and modularity for product family: product-modeling language (Erens et al. 1997), graphic representation (Ishii et al 1995; Agarwal and Cagan 1998), and module or building block (BB) (Tseng and Jiao 1996; Gero 1990; Fujita and Ishii 1997; Rosen 1996). The product modeling language allows product families to be represented in three domains: functional, technological, and physical. It provides an effective means for representing product variety but offers little aid for design synthesis and analysis. In the graph structure, the different types of nodes denote the individual components, subassemblies and fasteners, and the links denote dependencies between the nodes. However, it lacks the ability to model product family constraints. Although grammar approach was conjoint with graph representation to improve its capability of representation, graph grammars are only able to implicitly capture product architecture information and product family information by production rules (Siddique and Rosen 1999; 2001). A model specifically tailored for representation of product family architecture is the building block model, which is derived from the concept of using modules to provide varieties. Building blocks are organized in hierarchical decomposition tree architecture (systems, modules, and attributes) from both functional and technical viewpoints (Kusiak and Huang 1996; Jiao et al 2000). Under the hierarchical representation scheme, product variety can be implemented at different levels within the product architecture. However, module-based product architecture reasoning systems are currently being developed from different viewpoints (Rosen 1996).

Most of the work in strategic management and marketing research seeks to categorize or map the evolution and development of product families (Meyer et al. 1993; Wheelwright et al. 1989; 1992). Sanderson (1991) introduces the notion of a "virtual design" to evolve into product families. Wheelwright and Clark (1992) suggest designing "platform projects" and Rothwell and Gardiner (1990) advocate "robust designs" as a means to generate a series of different products within a single product family. These product family maps are less formal and are intended primarily for strategic

management; they are actually product platforms that can be used to generate product variants to form a product family. None of these approaches have been formalized for design synthesis.

The basic concept of a family of products or multi-product approach is to obtain the biggest set of products through the most standardized set of base components and production processes (McKay et al 1996). One of important aspects in developing product families is to consider the flexibility of assembly and manufacturing process. Stadzisz and Henrioud (1995) describe a methodology for the integrated design of product families and assembly processes through the use of web grammars (Pfaltz and Rosenfeld 1969). The work clusters products based on geometric similarities to obtain product families so as to decrease product variability within a product family and minimize the required flexibility of the associated assembly system. It is more applicable for the later design stages when more quantitative information is available.

Tseng and Jiao (1996; 1998) develop a set of approaches of Design for Mass Customization (DFMC) with an emphasis on how to "set up a rational product family architecture in order to conduct family-based design, rather than design only a single product." The family-based DFMC approach groups similar products into families based on functional requirements, product topology or manufacturing and assembly similarity and provides a series of steps to formulate an optimal product family architecture. Their work is also more applicable in the later stages of design, particularly once the system architecture has been established. Gonzale-Zugasti (2000) proposes a four-step interactive process model for designing a platform-based product family: design requirements and models (e.g. function requirements, and design constraints, etc), platform design, variants design, and platform evaluation, re-negotiation, and iteration.

The most important characteristics that have been stressed in literatures for designing product families are modularity (Chen et al 1994; 1996; Martin and Ishii 1996; Sanderson 1991; Ulrich and Tung 1991), commonality and reusability (Collier 1981; 1982; McDermott et al. 1994), and standardization (Lee and Tang 1997; Ulrich and Eppinger 1995). The concept of functional modularity should be incorporated with the requirements of product families from the product life cycle perspectives. Ulrich and Tung (1991) give a summary of different types of modularity. Chen et al. (1996) describe a family of products as a "family of designs" which conforms to a given ranged set of design requirements and recommend designing product families by changing a small number of components or modules. Ishii and

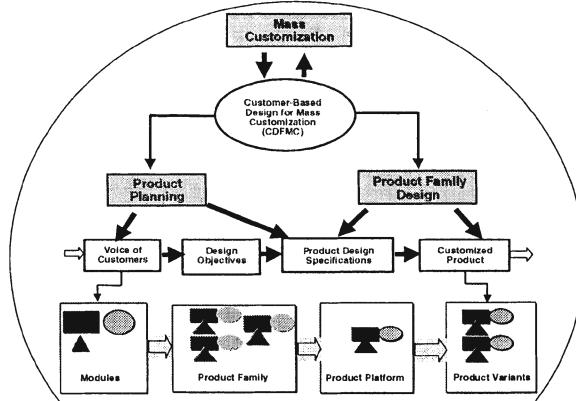
his team (Ishii et al. 1995; Martin and Ishii 1996; Chang and Ward 1995) work along the direction toward the computational approaches for product variety design. A series of studies focus on the representation, measurement and evaluation of product varieties in design. Design for Variety refers to product and process designs that meet the best balance of design modularity, component standardization, and product offering. Uzumeri and Sanderson (1995) emphasize flexibility and standardization as a means for enhancing product flexibility and offering a wide variety of products. McDermott (1994) and Collier (1981) stress commonality across products within a product family as an effective means to provide product variety. Ulrich (1995) and Ulrich and Eppinger (1995) investigate the role of product architecture and the impact on product change, product variety, component standardization, product performance, and product development management.

In summary, there are generally two approaches for product family design for mass customization. One is the top-down approach that adopts platform-based product family design (Simpson 1998). The other is the bottom-up approach which implements family-based product design through re-design or modification of constituent components of product. But the former one is dominant. Current research and development work is being focused mainly on academic area and no effective and efficient tools are available for industry, especially for knowledge support system. The most recent work in the area of product family design comes from Fujita et al (1999, 2000) and Simpson et al (2001). Much of their work lays a solid foundation for the work proposed in this paper. The approach advocated in this work is for companies to realize a family of modular products that can be easily modified, configured and quickly adapted to satisfy a variety of customer requirements or target specific market niches with knowledge support.

### **3. Customer-Based Product Family Design for Mass Customization**

There are three major technical challenges identified as reusability, product platform, and integrated product life cycle in implementing mass customization (Tseng and Jiao 1996, 1998). In view of these challenges, this research investigates mass customization from a product development perspective, namely customer-based design for mass customization (CDFMC), based on the belief that mass customization can be effectively approached from design. It is intended essentially to include customers into

the product development life cycle through proactively connecting customer needs in product planning stage to the capabilities of a company. The CDFMC elevates the current practice from designing individual products to designing product families. Product family has most impacts on a firm's ability to efficiently deliver large product variety and have profound implications for subsequent product development activities. A product family may have its origin in a differentiation process of a base product or in an aggregation process of distinct products. To support customized product differentiation, a product family platform is required to characterize customer needs and subsequently to fulfill these needs by configuring and modifying well-established building blocks. Figure 1 outlines a proposed concept framework for CDFMC used in this research. Figure 2 shows a product family architecture for mass customization.



*Figure 1. A framework for CDFMC using the modular product family design*

Recognizing the rationale of platform-based product family design with respect to mass customization, the whole design process can be divided into two main stages, product planning and family design. It ranges from capturing voices of customers and market trends for generating product design specifications to designing product family and customizing products for customers' satisfaction. Further, the product planning stage is to embed the voices of customers into the design objective and generate product design specifications, while the product family design is to realize sufficient product variety- a family of product to satisfy a range of customer demands.

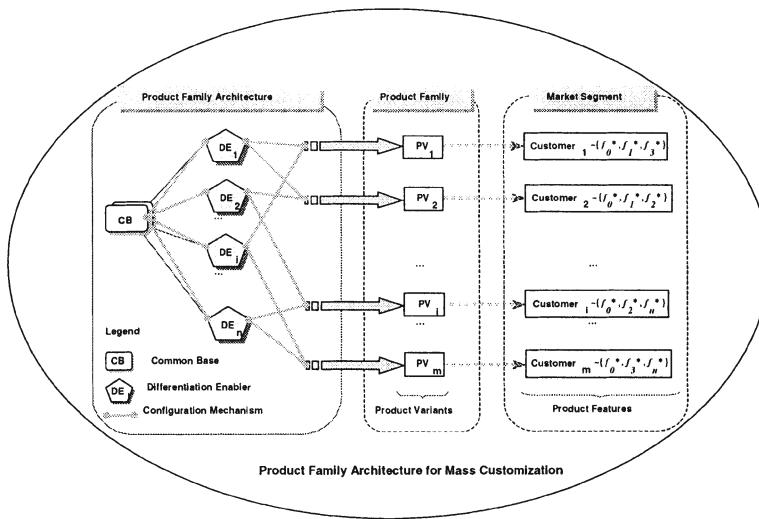


Figure 2. Architecture of product family for mass customization

#### 4. Module-Based Product Family Design

Modular systems provide the ability to achieve product variety through the combination and standardization of components. The platform-based modular product family design process advocated in this research is to develop a re-configurable product platform that can be easily modified and upgraded through the addition, substitution, and exclusion of modules to realize module-based product family. The bottom of Figure 1 displays the concept of module-based product family design. The focus of discussion below is on product family modeling, product platform generation, and product family evaluation for mass customization.

##### 4.1 PRODUCT FAMILY MODELING

###### 4.1.1 Product family architecture modeling

The product family architecture represents the conceptual structure and logical organization of product families from viewpoints of both customers and designers. A well-developed product family architecture can provide a generic architecture to capture and utilize commonality, within which each new product instantiates and extends so as to anchor future designs to a

common product line structure. Thus, the modeling and design of product architectures is critical for mass customizing products to meet differentiated market niches and satisfy requirements on local content, component carry-over between generations, recyclability, and other strategic issues.

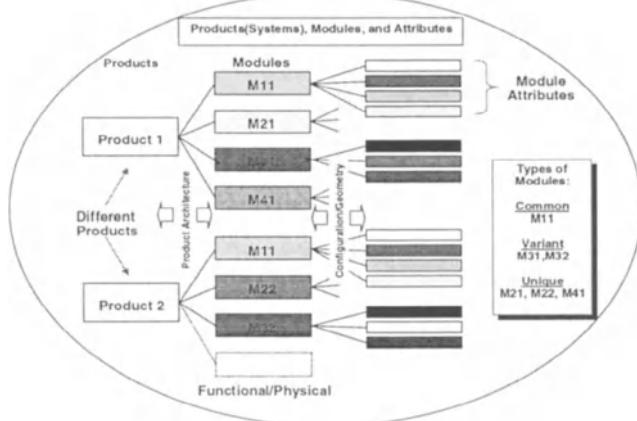


Figure 3. Products, modules, and attributes (Fujita and Ishii 1997)

The modeling and representation scheme used in this research is to combine recent developments in product representation coming from Fujita and Ishii (1997), Zha and Du (2001) and Rosen (1996) and integrate them into a hybrid approach. The hybrid approach hierarchically decomposes product families into products or systems, modules, and attributes, as shown in Figure 3. Under this hierarchical representation scheme, product variety is implemented at different levels within the product architecture. The discrete mathematics and matrix are used as a formal foundation for configuration design of modular product architectures. Based on the hybrid representation, a knowledge support product module reasoning system is currently being developed. Details will be discussed later.

#### *4.1.2 Product family evolution representation*

Product family maps or catalogs are intended for strategic management and can be used as product platforms to generate product variants to form a product family (Wheelwright and Sasser 1989). In this research, the product family map or catalog is used to trace the evolution of a product family, as shown in Figure 4. The market segmentation grid is used to facilitate identifying platform leveraging strategies in a product family (Meyer et al.

1997). The major market segment serviced by products is listed horizontally in a market segmentation grid and the vertical axis reflects different tiers of price and performance within each market segment. Similar to the work in (Simpson 1998), the market segmentation grid is applied to identify module-based product platform scaling opportunities from overall design requirements. As a qualitative approach, the beachhead method is most helpful for this research to identify and develop a common platform within a product family.

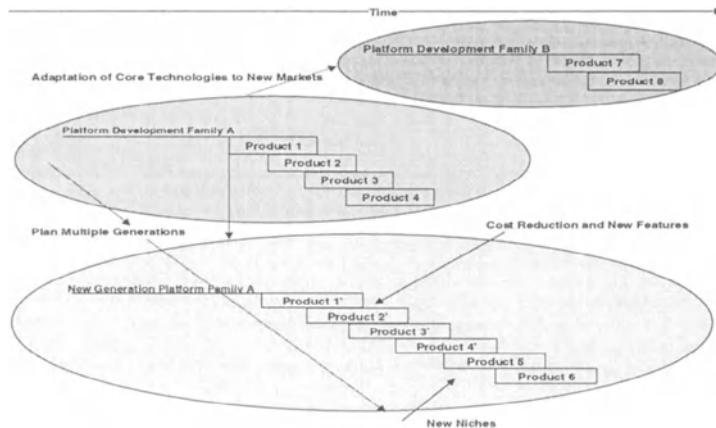


Figure 4. Product family evolution (Ref: Meyer and Utterback 1993)

#### 4.2 PRODUCT FAMILY GENERATION

Product family is generated through configuration design, in which a family of products can widely variegate the selection and assembly of modules or pre-defined building blocks at different levels of abstraction so as to satisfy diverse customer requirements (Tseng and Jiao 1996; 1998; Fujita et al 1998;1999). The essence of configuration design is to synthesize product structures by determining what modules or building blocks are in the product and how they are configured to satisfy a set of requirements and constraints.

In this research, the structured genetic algorithms (sGA) based product representation and evolutionary design scheme (Dasgupta and McGregor 1994) are employed for product family generation through modules configuration, as shown in Figure 5. The sGA product representation uses

regulatory genes that act as a switch to turn genes on (active) and off (passive). Each gene in higher levels acts as a switchable pointer that has two possible targets: when the gene is active (on) it points to its lower-level target (gene), and when passive (off) it points to the same-level target. At the evaluation stage only the expressed genes of an individual are translated into the phenotypic functionality, which means that only the genes that are currently active contribute to the fitness of the product. The passive genes do not influence fitness and are carried along as redundant genetic material during the evolutionary process. Therefore, the utilization of the sGA approach to product families can be summarized as follows. First, genes would represent modules that are either active or passive, depending on whether or not they are part of the product architecture. Then, a family of products relied on the addition or subtraction of modules meeting customer requirements could be evaluated by alternating different "active" and "passive" modules. A product family would thus correspond to product variants that have different active and passive combinations of modules.

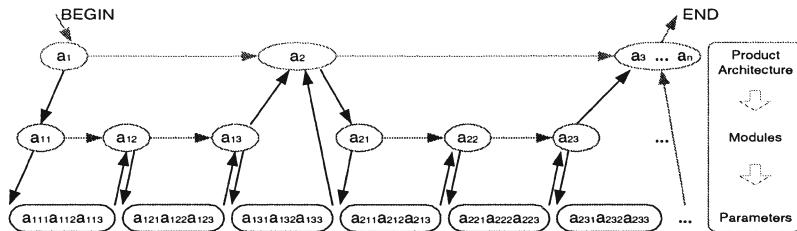


Figure 5. Structured GA for product design

#### 4.3 PRODUCT FAMILY EVALUATION FOR CUSTOMIZATION

The customization stage aims at obtaining a feasible architecture of product family member through reasoning product family module space according to customer requirements (Meyer et al 1997). There are two steps involved in this stage. First, custom requirements such as function, assemble, reuse, etc, need to be converted to constraints (Suh 1990). Then, the reasoning is performed at two levels: namely module and attribute levels, to determine feasible product family member architecture.

In order to evaluate a family of products for mass customization, suitable metrics are needed to assess the appropriateness of a product platform and the corresponding family of products. The metrics should also be useful for measuring the various attributes of the product family and assessing a

platform's modularity. Currently, there are many marketing or business, econo-technical metrics (Jiao and Tseng 1998; Sanderson 1991; William and Jordan 1999) that can be used for customer-based design for mass customization. For example, the following two typical metrics (Simpson 1998) are typically used for evaluation for mass customization in this research:

- (a) Market efficiency ( $\eta_M$ ), embodying a tradeoff between the marketing and the engineering design, offering the least amount of variety so as to satisfy the greatest amount of customers, i.e., targeting the largest number of market niches with the fewest products.
- (b) Investment efficiency ( $\eta_I$ ), embodying a tradeoff between the manufacturing and the engineering design, investing a minimal amount of capital into machining and tooling equipment while still being able to produce as large a variety of products as possible.

Therefore, both the market efficiency and the investment efficiency can be represented by the following two equations:

$$\eta_M = N_{tm}/N_M \quad (1)$$

$$\eta_I = C_m/N_v \quad (2)$$

where,  $N_{tm}$  and  $N_M$  are the number of the targetable market niches and the total market numbers, respectively;  $C_m$  and  $N_v$  are the manufacturing equipment costs and the number of the product varieties, respectively. Of course, a tradeoff also exists between the market efficiency and the investment efficiency as an increase in the investment efficiency through a decrease in product variety can cause a decrease in the market efficiency.

## **5. Knowledge Support for Customer-Based Design for Mass Customization: Framework**

Design process is knowledge intensive as there is a large amount of knowledge that designers call upon and use during the design process to match the ever-increasing complexity of design problems. Modularity, commonality and reusability, and standardization are the main characteristics for product family design. Designing product families requires the knowledge defining these characteristics and the use of knowledge during the design process (Stone et al 2000). Thus, there is a need for knowledge support.

In Section 3, a concept framework was suggested for the process of customer-based product family design for mass customization. To assist the designer in this process, a knowledge support framework is developed, as

illustrated in Figure 6. Design knowledge is classified into two categories: product information and knowledge, and process knowledge. These two categories of knowledge are utilized to support the process of customer-based design for mass customization. The knowledge support in the product planning stage is to assist the designer to capture voices of customers and market trends and embed them into the design objective for generating product design specifications. Accordingly, the knowledge support for product family design is to offer help for the designer to realize sufficient product variety- a family of product to satisfy a range of customer demands or requirements.

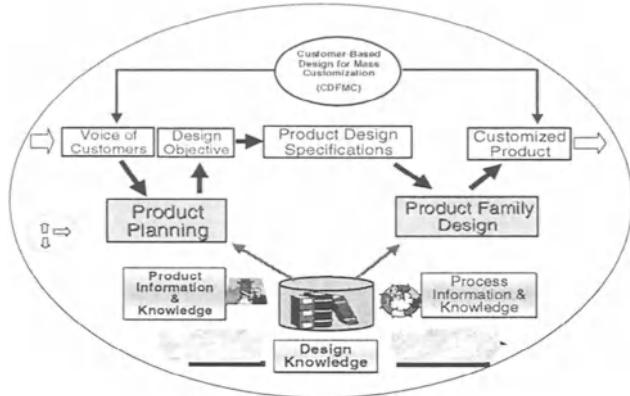


Figure 6. Knowledge support framework for CDFMC

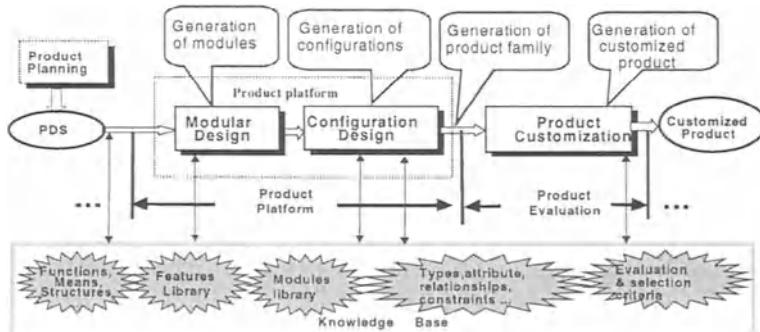


Figure 7. Knowledge support scheme for modular product family design

With understanding of the fundamental issues in product family design, a knowledge support scheme as shown in Figure 7 is proposed for modular product family design for mass customization. The knowledge support here is focused on the two main stages: product platform generation and product family evaluation, which are implemented through product planning for design specifications generation, modular design, configuration design and product family evaluation for customization.

## 6. Knowledge Support for Customer-Based Design for Mass Customization: Knowledge Modeling and Design Support Process

The implementation of knowledge support scheme proposed in the previous section is achieved through the following two steps: knowledge modeling and knowledge based support process, which will be discussed in this section.

### 6.1 KNOWLEDGE MODELING

Design knowledge refers to the collection of knowledge needed to support the design activities and decision-making in design process. The knowledge modeling in this research is to capture, represent, organize and manage product information and design knowledge in the process of product family design for mass customization, and to establish a comprehensive knowledge repository.

The product family design knowledge is abstracted and classified into different categories, e.g., off-line and on-line, product information and process knowledge, through analysis of product family design process. Different categories of design knowledge are represented in different ways from multiple views of product families. Since design knowledge includes all product information needed throughout the whole family design process, a new product information model is proposed. Thus, the following elements are essential in the knowledge repository: customer requirements, function-behavior, structure, product variety, assembly structure, component details, and product family parameters.

Based on a combination of elements of semantic relationships with the object-oriented data model, a multi-level hybrid representation schema (meta-level, physical level, geometric level) is adopted to represent the design process knowledge in different design stages at different levels (Suh 1990). To effectively manage and utilize the design knowledge, a generalized design knowledge matrix is proposed for organizing design

knowledge, in which all tasks in the design process are listed in column while all information and design knowledge are categorized in row. The contents of design knowledge for each task are recorded in the corresponding cells of the design knowledge matrix with appropriate representations.

## 6.2 KNOWLEDGE BASED DESIGN SUPPORT PROCESS

Once the design knowledge repository is built up, the user or designer can utilize the knowledge in it to solve problems in product family design for mass customization. As discussed in Sections 3 and 4 above, the whole design process was roughly divided into two main stages: product platform formulation for family generation and product evaluation or assessment for mass customization. Thus, the knowledge based design support process will cover these two stages.

### *6.2.1 Product platform formulation for family generation*

The knowledge based design support for modular product family design in Section 4 is fulfilled through the following steps:

- (1) The requirement analysis and modeling for a product (family) is carried out both from customer and designer viewpoints using design function deployment (DFD) and Hatley/Pirbhai technique (Sivaloganathan et al 2001, Rushton and Zakarian 2000). A function-function interaction matrix is generated.
- (2) The combination of heuristic and quantitative clustering algorithms is used to modularize the product (family) architecture, and a modularity matrix is constructed.
- (3) All modules in the product (family) are identified through the modularity matrix, and the types (functions) of all these modules can be further identified according to module classifications.
- (4) The functional modules are mapped to structural modules using the function-structure interaction matrix.
- (5) The hierarchical building blocks or design prototypes (Gero 1990) are used to represent the product (family) architecture from both the functional and the structural perspectives (Zha and Du 2001).
- (6) A hybrid genetic and simulated annealing algorithms is used to optimize product family architecture to achieve one main objective (Dasgupta and McGregor 1994, Reddy and Cagan 1995). Other design objectives

- are transformed into constraints for modules or their attributes. In addition, cost and profit models are also built as system constraints.
- (7) The product family architecture is rebuilt to form a hierarchical architecture by using the optimized modules from both the functional and the structural perspectives.
  - (8) The product family module space forms a product platform. The product family portfolio is derived from the product family module space.
  - (9) Standardize interfaces to facilitate addition, removal, and substitution of modules.

#### *6.2.2 Product evaluation for customization*

As discussed in Sections 3 and 4, design evaluation and selection are essential for CDFMC. This process experiences the elimination of unacceptable alternatives, the evaluation of candidates, and the final decision making under the customers' requirements and design constraints. While a number of methods have been investigated, there is still much to be desired due to the hindrance inherent in the conceptual evaluation and selection process (Jiao and Tseng 1998).

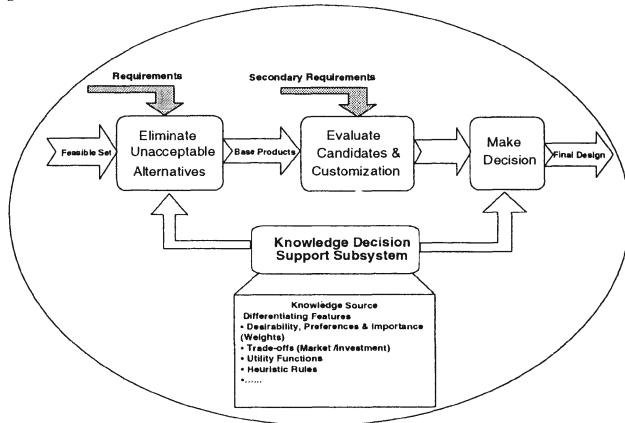


Figure 8. Knowledge support for product evaluation for mass customization

With respect to the traditional approaches (Pahl and Betiz 1996; Du and Tseng 1999; Jiao and Tseng 1998), an approach to conceptual evaluation and selection is proposed for product customization from the knowledge support viewpoint. The knowledge resource utilized in the process

extensively include differentiating features, customers' requirements, desires, preferences and importance (weights), trade-offs (e.g. market vs investment), and utilities functions, and heuristic knowledge, rules, etc. Figure 8 shows a knowledge decision support scheme for product evaluation and customization process. The kernel of the knowledge decision support engine is fuzzy clustering and ranking algorithms for design evaluation and selection (Jiao and Tseng 1998; Zha and Lu 2002).

## 7. Knowledge Support for Customer-Based Design for Mass Customization: System Architecture and Implementation

To assist the designer in CDFMC, a knowledge support system is required to generate, select and evaluate product families. Figure 9 displays an Internet and web enabled client/server implementation architecture for the knowledge support system. As shown in Figure 9, the web-based design framework adopts the design with modules, modules network, and knowledge support paradigms. The modularity of knowledge-based systems can thus be exploited, in that the inference engine and knowledge bases are located on server computers and the user interface is exported on demand to client computers via network connections. Therefore, modules under the framework are connected together so that they can exchange services to form large concurrent integrated models. The module structure leads itself to a client (browser) / knowledge server oriented architecture using distributed object technology.

The implementation of the web-enabled knowledge support system uses the two-tiered client-knowledge server architecture to support collaborative design interactions with a web-browser based graphical user interface (GUI). The underlying framework and the back-ended knowledge engine are written in Java. They could be integrated with the existing application packages such as CAD and database applications. CORBA could serve as an information and service exchange infrastructure above the computer network layer and provides the capability to interact with existing CAD applications and database management systems through other Object Request Brokers (ORB). In turn, the framework provides the methods and interfaces needed for the interaction with other modules in the networked environment.

Based on the architecture of the knowledge support system, its functionality can be achieved through implementing the following subsystems: Web GUI, knowledge repository, and modular design advisory

system. The knowledge repository is able to capture, store and retrieve design knowledge, including customer requirements, design objectives, design modules, design rationales, evaluation criteria, and product varieties, etc. The modular design advisory system (Design Advisor) includes decision-making mechanism and product module reasoning engine. The knowledge support product module reasoning engine is currently being developed to reason about sets of product architectures, to translate design requirements into constraints on these sets, to compare architecture modules from different viewpoints, and to enumerate all feasible modules using the "generate-and-test" or heuristic approaches.

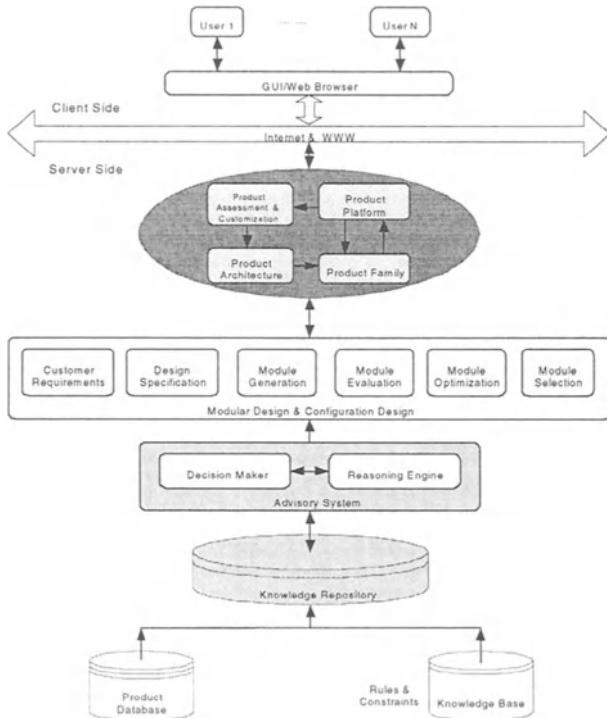


Figure 9. Internet and web-enabled knowledge support system architecture

The web GUI is a pure client of a knowledge server, delegating all events to the associated server. For wide accessibility and interoperability, the GUI is implemented as a web browser based client application. The front-end side of the application is implemented as a combination of XML documents,

VRML and Java applets. The back-end side system components may include knowledge repository, modular design server, product family generation server, product evaluation server, models and modules base server, and even knowledge assistant and explanation facilities (Siegel 1996; IONA 1997). The commercial ORB implementation of Java applets (OrbixWeb) is employed for the CORBA-based remote communication between the GUI Java applets and the back-end side system components.



Figure 10. Screen snapshot in power supply family design and evaluation

Figure 10 gives a screen snapshot for a preliminary implementation of the system. The prototype Design Advisor subsystem originated and modified from the work in (Samuel and Bellam 2000), but extended to a fuzzy comprehensive evaluation system that is still being developed. With this subsystem, the designer can represent the design choices available as a fuzzy AND/OR tree. The depth-first search algorithm employed in this subsystem is able to select the best design solution with minimum cost (or other minimum/ maximum measures for customization). The selected design choice is highlighted in the represented tree.

## 8. Case Study: Power Supply Family Design and Customization

To illustrate and validate the knowledge support scheme, scenarios of knowledge support for the design and evaluation of disk drive power supply family are provided. As discussed above, the whole design process of power supply family for mass customization ranges from capturing voices of customers and market trends to designing the family and customizing family members for customers' satisfaction.

From customers' point of view, a power supply product is defined on the required features: power, output voltage, output current, size, regulation, mean time between failure (MTBF), etc. From engineers' point of view, the power supply product is designed by determining these parameters: core of transformer, coil of transformer, switch frequency, rectifier, heat sink type, heat sink size, control loop, etc. Using the proposed modular product family design approach, three product families were generated based on three different topologies: Families II III, and I, which have 5,3, and 4 base products respectively, Figure 10. Each topology has its own limitations / ranges with regard to particular product features and/or design parameters.

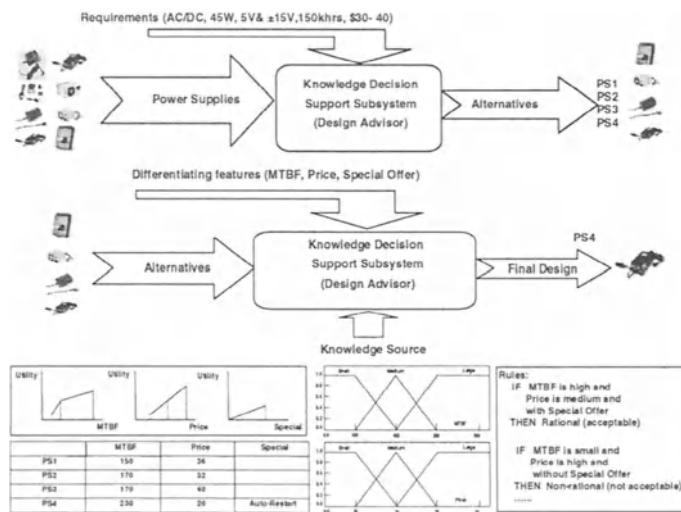


Figure 11. Scenario of knowledge support for power supply family evaluation

With reference to the knowledge decision support scheme for product evaluation, a scenario of knowledge support for power supply product

customization in Family I is shown in Figure 11. The customers' requirements for Family-I power supplies include AC/DC, 45W, 5V &  $\pm 15V$ , 150khrs, \$20-50, etc. The knowledge decision support system first eliminates all the unacceptable alternatives and determines four acceptable alternatives (PS1, PS2, PS3, and PS4). It then reaches the final design decision based on the knowledge resources given in the bottom of Figure 11, including differentiating features (MTBF, price, and special offer) and their utility / membership functions, fuzzy rules, and etc. The final design decision made by the system is PS4 as it has maximum MTBF, medium price and special offer of auto-start function and it is acceptable based on the rules. The system can explain the reasoning process, which makes a great difference between the knowledge support system and the traditional design and evaluation program.

## 9. Summary

This paper presented a preliminary work on the knowledge support for product family design for mass customization. A modular integrated product family design scheme is proposed with knowledge support for customer requirements' modeling, product architecture modeling, product platform establishment, product family generation, and product assessment for customization. The developed systematic methodology and framework can be used for capturing, representing, organizing, and managing product family design knowledge and offer support in the design process. Finally, the issues related to the implementation of the knowledge support framework are addressed. The system implementation architecture and functionality are provided to support product family design for mass customization. When fully developed, the system can effectively and efficiently support product family design for mass customization and improve customer satisfaction.

## References

- Agarwal, M and Cagan, J: 1998, A blend of different tastes: the language of coffeemakers, *Environment and Planning B: Planning and Design* 25(2): 205-226.
- Chang T-S and Ward AC: 1995, Design-in-modularity with conceptual robustness, *Design Technical Conference ASME*, DE-Vol. 82, pp. 493-500.
- Chen, W, Allen, JK, Mavris D and Mistree, F: 1996, A concept exploration method for determining robust top-level specifications, *Engineering Optimization* 26: 137-158.

- Chen, W, Rosen D, Allen J and Mistree, F: 1994, Modularity and the independence of functional requirements in designing complex systems, *Concurrent Product Design* **74**: 31-38.
- Collier, DA: 1981, The measurement and operating benefits of component part commonality, *Decision Sciences* **12**(1): 85-96.
- Collier, DA: 1982, Aggregate safety stock levels and component part commonality, *Management Science* **28**(22): 1296-1303.
- Dasgupta, D and McGregor, DR: 1994, A more biologically motivated genetic algorithm: the model and some results, *Cybernetics and Systems: An International Journal* **25**: 447-469.
- Du, X and Tseng, MM: 1999, Characterizing customer value for product customization, *Proceedings of DETC'99, ASME Design Engineering Technical Conference*, Paper No., DETC99/DFM-8916.
- Erens, F and Verhulst, K: 1997, Architectures for product families, *Computers in Industry* **33** (2-3): 165-178.
- Fujita, K: 2000, Product variety optimization under modular architecture, *Proceedings of Third International Symposium on Tools and Methods of Competitive Engineering (TMCE2000)*, pp. 451-464.
- Fujita, K, Sakaguchi, H and Akagi, S: 1999, Product variety deployment and its optimization under modular architecture and module commonalization, *Proceedings of the 1999 ASME Design Engineering Technical Conferences*, Paper No., DETC99/DFM-8923, ASME.
- Fujita, K, Akagi, S, Yoneda, T and Ishikawa, M: 1998, Simultaneous optimization of product family sharing system structure and configuration, *Proceedings of the 1998 ASME Design Engineering Technical Conferences*, Paper No., DETC98/DFM-5722, ASME.
- Fujita, K and Ishii, K: 1997, Task structuring toward computational approaches to product variety design, *Proceedings of the 1997 ASME Design Engineering Technical Conferences*, Paper No. 97DETC/DAC-3766, ASME.
- Gaithen, N: 1980, *Production and Operations Management: A Problem-Solving and Decision-Making Approach*, The Dryden Press, New York.
- Gilmore, JH and Pine, BJ: II, 1997, The four faces of mass customization, *Harvard Business Review* **75**(January-February): 91-101.
- Gero, JS: 1990, Design prototypes: a knowledge representation schema for design, *AI Magazine* **11**(4): 26-36.
- Gonzale-Zugasti, JP: 2000, *Models for Platform-Based Product Family Design*, PhD Thesis, MIT, Cambridge, MA.
- Ishii, K, Juengel, C and Eubanks, F: 1995, Design for product variety: key to product line structuring, *Proceedings of the ASME Design Theory and Methodology Conference*, DE-Vol. 83: pp. 499-506.
- IONA, 1997: *Orbix2 Programming Guide*: IONA Technologies Ltd.
- Jiao, JX and Tseng, MM: 1998, Fuzzy ranking for concept evaluation in configuration design for mass customization, *Concurrent Engineering: Research and Application* **6**(3): 189-206.
- Jiao, J, Tseng, MM, Ma, Q and Zou, Y: 2000, Generic bill of materials and operations for high-variety production management, *Concurrent Engineering: Research and Application* **8**(4): 297-322.
- Kotler, P: 1989, From mass marketing to mass customization, *Planning Review* **17**(5): 10-15

- Kusiak A and Huang C-C: 1996, Development of modular products, *IEEE Trans. on Components, Packaging, and Manufacturing Technology, Part-A* **19**(4): 523-538.
- Lee, HL. and Tang, CS: 1997, Modeling the costs and benefits of delayed product differentiation, *Management Science* **43**(1): 40-53.
- Martin, M and Ishii, K: 1996, Design for variety: a methodology for understanding the costs of product proliferation, in K Wood (ed.), *Design Theory and Methodology Conference*, ASME, Irvine, CA, Paper No., 96-DETC/DTM-1610
- McDermott, CM and Stock, GN: 1994, The use of common parts and designs in high-tech industries: a strategic approach, *Production and Inventory Management Journal* **35**(3): 65-68.
- McKay, A, Erens, F and Bloor, MS: 1996, Relating product definition and product variety, *Research in Engineering Design* **8**(2): 63-80.
- Meyer, MH: 1997, Revitalize your product lines through continuous platform renewal, *Research Technology Management* **40**(2): 17-28.
- Meyer, MH and Utterback, JM: 1993, The product family and the dynamics of core capability, *Sloan Management Review* **34**(Spring): 29-47.
- Meyer, MH, Tertzakian P and Utterback, JM: 1997, Metrics for managing research and development in the context of the product family, *Manage Science*, **43**(1): 88-111.
- Nutt, GJ: 1992, *Open Systems*, Prentice Hall, Englewood Cliffs, NJ.
- Pahl, G and Beitz, W: 1996, *Engineering Design - A Systematic Approach*, Berlin, Heidelberg, Springer, New York.
- Pfaltz, JL and Rosenfeld, A: 1969, Web Grammars, *Proceedings of First International Joint Conference on Artificial Intelligence*, Washington, DC, pp. 609-619
- Pine II, BJ: 1993, *Mass Customization - The New Frontier in Business Competition*, Harvard Business School Press, Boston, MA.
- Rosen, DW: 1996, Design of modular product architectures in discrete design spaces subject to life cycle issues, *1996 ASME Design Automation Conference*, Irvine, CA. 96-DETC/DAC-1485
- Reddy, G and Cagan, J: 1995, An improved shape annealing algorithm for truss topology generation, *Journal of Mechanical Design* **117**: 315-321.
- Rothwell, R and Gardiner, P: 1990, Robustness and product design families, M Oakley, (ed.), *Design Management: A Handbook of Issues and Methods*, Basil Blackwell Inc., Cambridge, MA, pp. 279-292.
- Rushton, G and Zakarian, V: 2000, *Development of Modular Vehicle Systems*, Department of Industrial and Manufacturing Systems Engineering, University of Michigan, Dearborn.
- Sanderson, S and Uzumeri, M: 1995, Managing product families: the case of the sony walkman, *Research Policy* **24**, 761-782.
- Samuel, A.K and Bellam, S: 2000. <http://www.glue.umd.edu/~sbellam/>
- Sanderson, SW: 1991, Cost models for evaluating virtual design strategies in multi-cycle product families, *Journal of Engineering and Technology Management* **8**, 339-358.
- Siddique, Z and Rosen, DW: 1999, Product platform design: a graph grammar approach, *Proceedings of DETC'99, 1999 ASME Design Engineering Technical Conferences*, Paper No., DETC99/DTM-8762.

- Siddique, Z and Rosen, DW: 2001, On discrete design spaces for the configuration design of product families, *Artificial Intelligence in Engineering, Design, Automation, and Manufacturing* 15: 1-18
- Siegel, J: 1996, *CORBA: Fundamentals and Programming: OMG*, John Wiley and Sons, New York.
- Simpson, TW: 1998, *A Concept Exploration Method for Product Family Design*, Ph.D Dissertation, System Realization Laboratory, Woodruff School of Mechanical Engineering, Georgia Institute of Technology, Georgia.
- Simpson, TW, Maier, JRA and Mistree, F: 2001, Product platform design: method and application, *Research In Engineering Design* 13: 2-22
- Sivaloganathan, S, Andrews, PTJ and Shahin, TMM: 2001, design function deployment: a tutorial introduction, *Journal of Engineering Design* 12(1): 59-74.
- Stadzisz, PC and Henrioud, JM: 1995, Integrated design of product famalies and assembly systems, *IEEE International Conference on Robotics and Automation* 2: 1290-1295
- Stone, RB, Kristin, LW, and Crawford, RH: 2000, A heuristic method for identifying modules for product architectures, *Design Studies* 21(1): 15-31.
- Suh, NP: 1990, *The Principles of Design*, Oxford University Press, New York.
- Tseng, MM and Jiao, JX: 1996, Design for mass customization, *CIRP Annals* 45(1): 153-156.
- Tseng, MM and Jiao, JX: 1998, Product family modeling for mass customization, *Computers in Industry* 35(3-4): 495-498.
- Ulrich, K and Tung, K: 1991, Fundamentals of Product Modularity, *Proceedings of ASME Winter Annual Meeting Conference*, ASME, DE-Vol. 39, pp. 73-80.
- Ulrich, K: 1995, The role of product architecture in the manufacturing firm, *Research Policy* 24(3): 419-440.
- Ulrich, KT and Eppinger, SD: 1995, *Product Design and Development*, McGraw-Hill, New York.
- Uzumeri, M and Sanderson, S: 1995, A Framework for model and product family competition, *Research Policy* 24: 583-607.
- Wheelwright, SC and Sasser, WE, Jr: 1989, The new product development map, *Harvard Business Review*, 67(May-June): 112-125.
- Wheelwright, SC and Clark, KB: 1992, Creating project plans to focus product development, *Harvard Business Review* 70(March-April): 70-82.
- William LM and Jordan JL: 1999, Using conjoint analysis to help design product platforms, *Journal Production Innovation Management* 16: 27-39
- Zha, XF, Du, H: 2001, Mechanical systems and assemblies modeling using knowledge intensive petri net formalisms, *Artificial Intelligence for Engineering Design, Analysis and Manufacturing* 15(2): 145-171.
- Zha, XF and Lu, WF: 2002, Knowledge intensive support for conceptual evaluation and selection in customer-based design for mass customization, submitted to *ASME DECT 2002*, Montreal, Canada.

## ELUCIDATING THE DESIGN REQUIREMENT FOR CONVENTIONAL AND AUTOMATED CONCEPTUAL DESIGN

MANSUR DARLINGTON AND STEPHEN J CULLEY  
*University of Bath*  
UK

**Abstract.** It is acknowledged that the design requirement capture phase of design is an important one, and that failures in this phase of the design often occur to the detriment of the end product. Thus assisting designers in this phase of design is useful. Elicitation of the design requirement from the customer (however defined) and evolving and translating it into a representation appropriate for driving design is a knowledge-intensive activity. In particular, both the customer and the designer must have knowledge of the domain in which they are working, and have overlapping sets of this knowledge. Without this overlap communication would be impossible. Similarly, in order to develop computerized designer support systems for this phase of the conceptual design process, it is necessary that the knowledge brought to this process by the human practitioners be shared between the human user and the computer. This is essential if meaningful communication and inference is to take place. This paper considers the basis for knowledge sharing and communication, and how knowledge 'contexts' provide the foundation for mutual understanding. This model is used to suggest a mechanism by which incomplete domain knowledge can be identified and embodied. The domain knowledge can then be applied so that communication between human and machine can be enhanced to provide design support in the design requirement elicitation process.

### 1. Introduction

Few would argue about the importance of the design requirement in ensuring that a design episode results in a successful product (see for example, Pahl and Beitz 1996; Ullman 1997).

At the core of the design requirement capture process is the elicitation of customer needs by the designer and their transformation into appropriate terms that can be satisfied in the design solution (see for example, Ulrich and Eppinger 1995; Wootton et al. 1997). Clearly, it is a process reliant on

communication, and when communication failures occur, so too can the capture process itself, to the detriment of the designed product.

Communication between human beings is dependent on a mutual understanding of the area being discussed, which in turn is dependent on shared knowledge. This is developed through shared experience of the real world.

The purpose of the work reported in this paper is to provide better computer-based support for the design requirement capture process, based on improved communication both between humans and between human and machine.

Section 2 presents an analysis of knowledge acquisition through experience, and how concepts and conceptual schemata are used to guide communication by providing appropriate shared *contexts*. In addition the relationship between humans and machines is considered. It is then shown how the different basis for knowledge acquisition makes communication difficult and the source of error. The insights gained from these considerations suggest a method of knowledge sharing to aid communication between humans and machines using artificial ‘contexts’. This is presented in Section 3. Section 4 then illustrates the application of this method in providing contexts for assisting in the development of the engineering design requirement for product surface finish.

## 2. Knowledge and Communication

Humans gain their knowledge of the world incrementally, through their experience. By this means they build up a complex internal model which more or less reflects the external reality. Additions to the knowledge of an individual is a function of new experience applied to the current knowledge structure.

When thinking about the world consciously – that is when modelling it mentally – concepts are used. The term ‘concept’ is widely used, in a number of ways, and definitions abound. The authors formulate this definition:

*A concept is a collection of propositions about a separable component of the world designated by a label.*

Concepts may relate to concrete or abstract objects, and can be more or less complex depending on the entity to which they refer. The concept *apple*, for example, is associated with an indefinite number of propositions<sup>1</sup> that relate

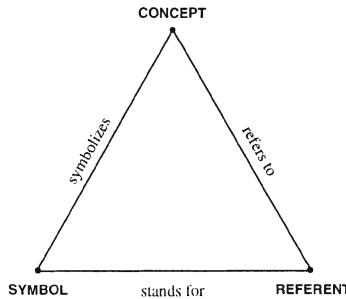
---

<sup>1</sup> A proposition is an assertion about a belief that is held

to that concept such as, for example, that 'apples are round', that 'apples are sometimes red', that 'apples have mass', 'apples and pears are similar things' etc. Each proposition itself contains references to other concepts. This cluster of propositions that 'support' a concept is referred to as the conceptual schema for that concept. Conceptual schemata embody the general knowledge that the individual has about the world, and thus its meaning to the individual (Stillings et al. 1995). Although concepts and conceptual schemata are private and unique to each individual, they are developed as a result of experience of parts of the same world, which allows sharing in knowledge and meaning, and thus mutual understanding to occur. By being a separable component of the world, the concept is demonstrating that the referent in the world to which it refers is demonstrating a perceivable regularity that makes it stand out and be recognizable from one encounter to another. Importantly, this supports the view that the concept may be recognizable, and thus shareable, by others.

Schemata can be extended to embrace whole clusters of concepts that are associated with a larger aspect of the world. Thus, the conceptual schema for apple can be incorporated into a conceptual schema for fruit, which in turn can be incorporated in the conceptual schema which relates to the individual's knowledge about fruit-growing, and so on.

Ogden and Richards (1923) provide an exemplary clarification of the relationships between concepts, labels (symbols) and meaning in their 'meaning triangle'.



*Figure 1. Ogden and Richards' (1923) 'concept triangle'*

Thus, a concept refers to a referent (an entity in the world). The symbol or label is used to point to the referent and symbolizes the meaning of the concept.

The set of propositions that constitute the meaning of the concept will be different for each individual. Communication between individuals is possible just because common experience assigns an overlapping (although

not isomorphic) meaning to a referent *a* that is assumed to exist in the ‘real’ world, which by agreement is given a label.

The existence of a label or a symbol is in itself evidence that an entity in the world has been recognized in common experience. Thus, if labels can be identified which characterize a domain of discourse, it can be assumed – subject to contrary evidence – that they symbolize concepts that refer to aspects of the real world which are similar across individuals and thus shared. It follows that the association or relationships between concepts will also have some commonality. Labels are evidence of shared associations of meaning, and thus can be used to ‘point’ to the underlying shared knowledge.

## 2.1 THE POWER OF CONTEXT

Complex schemata are important in the way that people deal with the world. Once a schema has been activated by stimuli from the outside world, or through mental activity, it provides a focus for further activity. In particular it allows irrelevant information to be discarded or put in the background, and brings into focus that which is relevant to current processing. It is this constraining mechanism that underpins the intuitive notion that given some ‘seed’ stimulus: ‘one idea leads to another (related) idea’. This ability to shift the focus of attention from one sub-area of world knowledge to another is vital: without it the individual would become swamped with conflicting and irrelevant information, and a coherent train of thought could not be developed. These complex schemata can be thought of as providing *Context*.

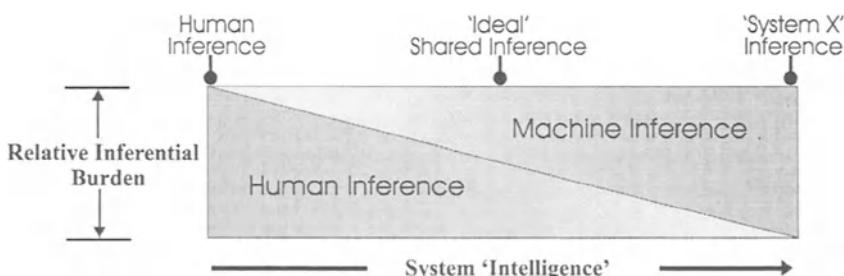
In communication – such as the dialogue between a customer and designer – clues to what is the currently appropriate context are provided by the background information that the listener and speaker tacitly assume. Context doesn’t provide a fixed boundary, but some graduated measure of appropriateness or appositeness based on meaning. By circumscribing the conceptual ‘locality’, context is vital in providing a basis for making judgements about the likely meaning of terms when used by others, gives clues about the mental states of others, and suggests what might be appropriate and relevant for discussion and deliberation (Lenat 1998). Indeed ‘the power of contexts is that they greatly restrict the possible inferences so that it is easier to deduce the relevant facts about any particular situation. In addition the context may provide extra facts to be used in any such inference’ (Edmonds 1997). Without context, the intelligent inferences required for communication would be difficult to achieve.

In summary, then:

- Modelling the world mentally – i.e. thinking – involves bringing concepts into play and manipulating them.
- Individuals acquire concepts through experience.
- Common experience develops overlapping concepts and conceptual schemata in different individuals, by which are constructed intersecting internal models of external reality.
- Identifying and labelling these overlaps provides a means of communication.
- At any time all the concepts held by an individual are potentially available for use in thinking, but only those that seem useful or appropriate for the nonce are used.
- Appropriate concepts are constrained by a context of associated concepts.

## 2.2 THE HUMAN–MACHINE RELATIONSHIP

Figure 2 proposes one way in which the relationship between humans and machines might be thought of. It introduces the idea of ‘relative inferential burden’ which merely reflects where the task of reasoning is carried out, and shows how the reasoning task is shifted progressively from human to machine as inferential power of the computer increases. On the far left is represented the situation where, since the machine has no inferential power, the human alone has the reasoning burden. On the far right, ‘System X’ represents the situation where machine inference is sufficiently powerful that the human can sit back and let the machine do all the work.



*Figure 2. The human–machine relationship in the context of designer support*

It remains to be shown whether System X will ever eventuate. Even then, it might be argued, the use of System X alone would not be desirable. Simply: for all its power and subtlety, human reasoning is faulty, thus there is no evidence to suggest that machine reasoning will be any more perfect.

The ideal human/machine relationship, then, becomes one where the inferential powers of the combined systems is greater than the sum of the parts, the machine supplying reasoning in those areas where human weakness requires support, and vice versa. This reflects Cross's (Cross 2001) assertion that 'the computer should be asking questions of the designer, seeking from him those decisions which it is not competent to handle itself'.

For this to occur communication between human and machine is a prerequisite. For communication between human and machine to be meaningful, in the sense that it is meaningful between humans, the basis for inference about the world must be similar. As argued above, for this to be the case in some way the knowledge must be shareable.

It has been shown above how knowledge is built up through experience, and how shared experience allows shared knowledge as the basis for communication.

Humans gain their experience of the world through the five senses supported by a human body, and develop and refine their knowledge through the use of their human mind. Because of this, they are 'situated' in the world in a uniquely human way. Clearly, since computers are not situated in the world as are humans, having neither the same senses, nor the same intellectual faculties with which to develop their knowledge, they cannot acquire shared knowledge. Thus, if communication is to be effected then artificial ways must be found by which 'sharing' of knowledge becomes possible.

### **3. Mapping the Design Requirement Capture Concepts**

Bearing the previous issues in mind, the following sections detail one method by which domain knowledge might be partially revealed and structured in such a way as to enhance human-machine communication, using the idea of concepts and context. This method is developed below in consideration of the concepts that might be associated with thinking about engineering design requirements.

The insights gained from the above discussion are:

- That since computers are not situated in the world in the same way as humans, acquiring knowledge through the shared experience is not possible. Thus, for meaningful interactions to take place between humans and computer 'knowledge' of appropriate structure and content must be introduced (this, after all, is the chief preoccupation of AI).
- Through the labels used to refer to them and an analysis of their use it may be possible to build up clusters of concepts into contexts that reflect the sort of knowledge and its organization that is shared in

developing the engineering design requirement. By doing this the knowledge content of the subsumed domains can be mapped at a low level of resolution.

- The content and organization of the conceptualizations might constitute a basis for providing artificial ‘contexts’<sup>2</sup>, analogous to and usefully conformal with the contexts used in the real world.
- Making the content explicit in this way, and agreeing a commitment to its use in the form agreed, would provide a basis for supporting communication between humans during the elicitation process. It would, similarly, provide the basis for knowledge sharing between user and machine upon which meaningful interaction can take place.

### 3.1 ONTOLOGIES

Amongst others, one vehicle that suggests itself for the investigation of contexts and the specification of ‘contexts’ is the ontology. There is some discussion as to the proper definition of ontology; for the purpose of this work the two that follow are illuminating. According to Gruber (1993), an ontology is a ‘specification of a conceptualization’, where a conceptualization is ‘the objects, concepts, and other entities that are of interest and the relationships that hold among them (Genesereth and Nilsson 1987). As Gruber observes elsewhere, a conceptualization is an abstract simplified view of the world. For the purposes of this work, the conceptualization will be the context for a particular domain of discourse, specifically, those things that it is appropriate to talk about when developing the design requirement.

The Knowledge Systems Laboratory at the University of Stanford provide a supporting definition of ontology: "... it is a formal and declarative representation which includes the vocabulary (or names) for referring to the terms in that subject area and the logical statements that describe what the terms are, how they are related to each other, and how they can or cannot be related to each other. Ontologies therefore provide a vocabulary for representing and communicating knowledge about some topic and a set of relationships that hold among the terms in that vocabulary." They provide the basis of making implicit conceptualizations explicit and therefore accessible.

---

<sup>2</sup> Throughout this paper the word *context* will be used to refer to the conceptual ‘locality’ that might obtain when thinking and talking about a particular subject. When the intention is to convey the idea of representing a context in an artificial way, then the word will placed in single inverted commas, thus: ‘context’.

Noy and McGuinness (2000) identify five reasons for the development of an ontology, each one of which seems apposite in relation to supporting the elicitation of the design requirement. The reasons are:

- To share common understanding of the structure of information amongst people or software agents
- to enable reuse of domain knowledge
- to make domain assumptions explicit
- to separate domain knowledge from the operational knowledge
- to analyse domain knowledge

The premiss for this work on supporting the design requirement capture process is thus:

*Context* is a structure of conceptualizations that represents some of the knowledge about a domain of discourse.

*Context* provides the means for aiding communication.

*Ontology* provides the means for defining context.

#### 4. Methodology

The methodology adopted to explore and develop a conceptual framework for the engineering design requirement is presented below.

Initially, a hierarchical framework of meta-concepts relating to the engineering design requirement was formulated. Because of space constraints only the top level of the hierarchy is shown, together with an elaboration of the *aesthetics* branch, in Figure 3. The complete hierarchy represents a starting-point in identifying the areas of interest that might be mapped ontologically when attempting to provide a full set of contexts for supporting discourse during design requirement capture. The hierarchy represents an amalgam of the topics identified as being crucial for development of the design requirement by influential engineering design methodologists (Hales 1993; Pugh 1991; Pahl and Beitz 1996; Ullman 1996; Bath 2001). The hierarchy is not intended as a prescription for organization nor an indication of the level at which design requirement domain as a whole must be specified, but a pointer to the topics that must be covered in general, each one of which might be a candidate for elaboration as a 'context'. Although it is not one itself, the hierarchy represents the basis for a top-level ontology for the subject area.

For the development of the context ontologies themselves, the approach taken is based on the methodology espoused by Noy and McGuinness (2000) for the development of a declarative frame-based ontology. They key elements of this methodology is as follows:

- Determining the domain and the scope of the ontology.

- Considering reuse of existing ontologies
- Enumerating important terms in the ontology
- Defining the classes and the class hierarchy
- Defining the properties of classes
- Defining the values for the properties

Creating instances of classes

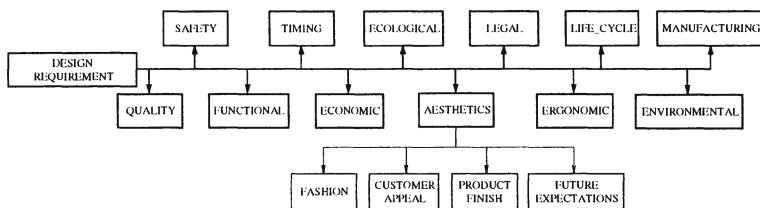


Figure 3. The design requirement hierarchy with the aesthetics branch elaborated

Space limitations preclude discussion of each of the elements of the methodology; instead two core elements (common to any approach to ontology design) are treated here in the context of a case-study relating to the development of an ontology for ‘Product Finish’. These elements – *determining the scope of the ontology* and *enumerating important terms* – are considered below.

#### 4.1 DETERMINING THE SCOPE OF THE ONTOLOGY

Crucial to the success of the ontology development is determining the domain and the scope of the ontology. Establishing this can be assisted by answering the following three questions:

*1. What domain of interest will the ontology cover?*

In this case the domain is that of product and component surface finish.

*2. For what will the ontology be used?*

The purpose of the Product Finish ontology is to provide a guiding ‘context’ to support a method that will ensure that between them a *customer* and a *designer* will raise and answer all the questions necessary to complete the design requirements relating to the finish of a manufactured product.

*3. For what types of question will the information in the ontology provide answers?*

These questions are referred to as *competency questions*. They are of immense importance in focusing in on what the ontology is to be used for,

and providing guidance as to the structure and content of the ontology. The use to which the ontology is to be put is critically important in deciding the level of description for the entities in the conceptual space. Competency questions assist in clarifying what the entities are and at what level they might best be described. In addition, the competency questions provide a means by which the ontology, and its implementation in some problem-solving method, can be validated.

The competency questions need not be exhaustive, merely indicative. However, they do require that predictions be made about the use to which the ontology is to be put.

Listed here are some of the sorts of questions posed for the 'Product Finish' ontology. The classes of concept suggested by the questions are given in square brackets; they (perhaps with different labels) will appear as main categories. The questions can be grouped loosely under a number of headings relating to 'finish', at various levels of generality, for example *type, function, properties, aesthetics, performance, safety, contractual and regulatory*. These, too, represent candidates for inclusion as elements of the ontology.

#### *Type*

1. Is a particular specific finish method or process a design requirement?  
[finish application method entities required.]
2. Does the finish have to comply with a formally designated specification? [specification entity required]
3. Is a particular surface material (e.g. paint) a design requirement?  
[material type entities required].

#### *Function and purpose*

4. What is the function of the finish? [function entities required]
5. Is the finish functional performance quantified? [metrics properties required].

#### *Properties*

6. What are the required properties of the finish? [property entities required].

#### *Operating conditions*

7. What operating/services conditions must the finish withstand?  
[operating condition entities required].

#### *Safety and use*

8. Is user safety an issue in relation to the finish? [hazard entities required?]
9. Does the finish have to conform to a specific safety standard/specification?

#### *Contractual and regulatory*

10. Does the finish have to meet a designated regulatory standard/specification?
11. Does the finish have to meet a designated contractual standard?

#### 4.2 ENUMERATING IMPORTANT TERMS

The process of developing an ontology is one of generation and revision, and, as can be seen below in the example, an elaboration of conceptual hierarchies at ever-greater levels of detail. The approach adopted for enumerating the terms for the ‘contexts’ is:

*Conceptual mapping.* This is an informal process assisted by brainstorming and discussion, which seeks to identify important concepts in the domain of interest. It also begins to suggest important associations with other concepts.

*Definition.<sup>3</sup>* This process includes selecting the label or term which is to be used to symbolize each concept, defining the label and recognizing and excising synonyms. Importantly, definition helps eradicate the ambiguity and contradiction that is inherent in human discourse.

*Assigning relations.* This is the principal means by which concepts can be assigned properties and associated with other concepts.

In the case study, conceptual mapping resulted, after a number of revisions, in a second-level hierarchy for Product Finish, shown in Figure 4. The entities identified here were considered to be those relating to product finish that are candidates for exploration during elicitation of the design requirement for this aspect of new product.

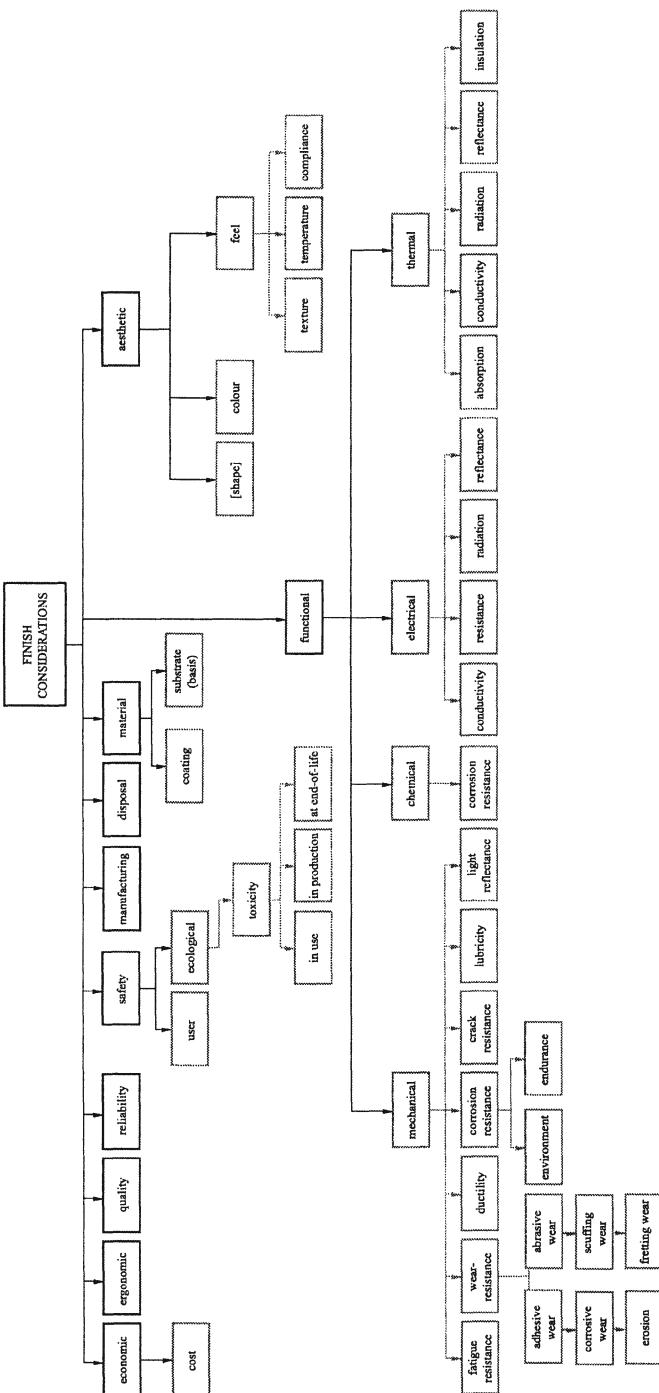
Using this and the competency questions as guidance the complete ontology was then developed and definitions and relations assigned. A basic understanding of finish methods and types, and surface engineering technology was augmented by technical reference works on the subject (Gabe 1972; Durney 1984) and reference to a industry available expert system for finish selection (APTICOTE-ISIS 2001).

In addition, the ontology was agreed as usefully representing the domain by surface engineering experts. The hierarchy for the completed ontology contains 170 linked entities.

---

<sup>3</sup> As an aside to the work reported here, a top-level ontology has been developed in order to capture and define the terms used in referring to entities in the design requirement domain. The content of this, too, is influenced by the terms used by these methodologists. Not only does the top-level ontology provide a means by which clarification can be made of terms used in design requirement elicitation by humans and so aid communication, so too is it useful in providing a basis for defining elements of the design requirement for computational use. This is available online at:

[www.bath.ac.uk/~ensmjd/ontologies/Design\\_Requirement/Design\\_Requirement.htm](http://www.bath.ac.uk/~ensmjd/ontologies/Design_Requirement/Design_Requirement.htm)



*Figure 4.* This hierarchy of meta-concepts is the first elaboration of a leaf in the main Design Requirements hierarchy. It constitutes the starting point for the iterative process of developing the Product Finish context ontology

By way of illustration, Figure 5 shows the hierarchy for the *function* branch of the ontology<sup>4</sup>.

#### 4.3 USING A 'CONTEXT' ONTOLOGY

The purpose of developing a 'context' ontology is to attempt to:

- Identify at a low level of resolution the knowledge content needed when talking about the domain of interest; and,
- to structure the knowledge content in such a way that it provides a 'context' for talking about the subject when eliciting the design requirement for this aspect of a new product.

There are various ways in which the 'context' provided by the ontology can be used, a number of which are identified below by way of illustration. The most basic level at which the ontology can be implemented is as *checklist*. When used as a checklist, the ontological approach to defining the 'context' helps ensure that it is specified in a coherent and agreed manner. Making the best of the ontology, however, requires a more sophisticated implementation in some support system.

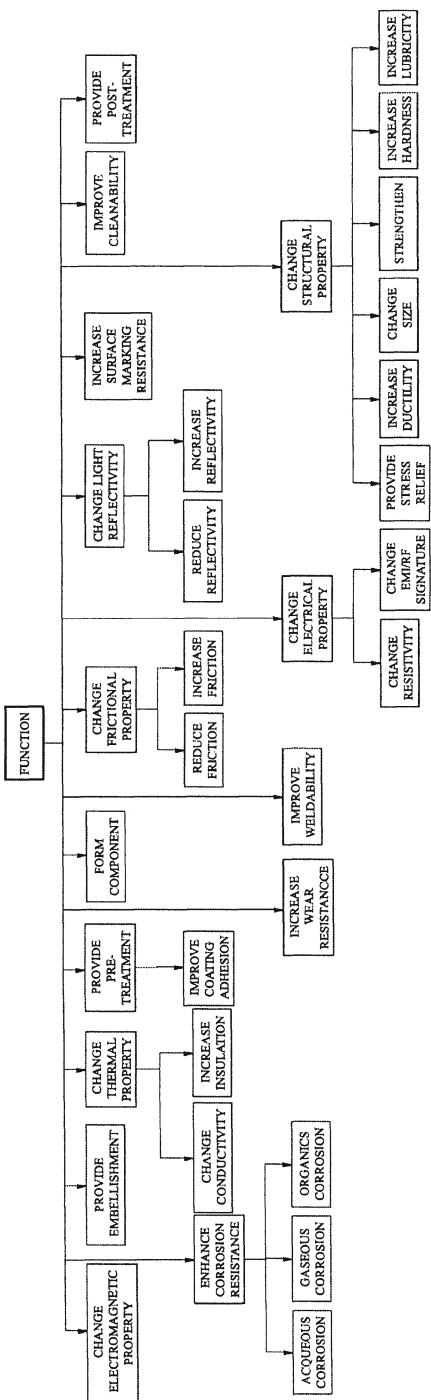
Built into the Product Finish ontology is the class concept: 'finish design requirement', which is specified in terms of a set of attributes and relations. An instance of this concept represents a single product finish design requirement, and the general entity 'design requirement' can be specialized as necessary to any sub-area of the design requirement as a whole, as it has been here.

Protégé 2000 (Protégé 2000), the ontology editor used to develop the context ontologies, incorporates a knowledge acquisition mechanism that allows instances of concepts classes in the ontology to be instantiated. A single instantiation of the class 'finish design requirement' represents a design requirement for a single design episode. Thus, it can be shown that this mechanism can be used to capture a design requirement. This method of capturing the requirement is representative of those that are essentially *static elicitation* methods, in that the means for eliciting case-specific responses is limited; nonetheless it is clearly an improvement over the basic checklist.

A more satisfactory approach is to provide a *dynamic elicitation* method for capturing the design requirement, where the system can guide the elicitation not only within the context, but sensible of the developing current situation.

---

<sup>4</sup> The completed Product Finish ontology is available on-line at [www.bath.ac.uk/~ensmjd/ontologies/Product\\_Finish/Product\\_Finish.htm](http://www.bath.ac.uk/~ensmjd/ontologies/Product_Finish/Product_Finish.htm).



*Figure 5.* The completed hierarchy for the Function elements of the Product Finish ontology.

This is analogous to the way that context is used by humans. One approach to achieving this has been reported in full in Darlington *et al.* (2001) using an informal ontology to provide context in the domain of machine motion. This is described briefly in the next Section to show one possible approach to the implementation of a dynamic system that supports elicitation of the design requirement capture process.

Another approach to developing dynamic ontologies is suggested by one of the uses for ontologies cited by Noy and McGuinness (2000): ‘to share common understanding of the structure of information amongst people or software agents’. As well as defining concepts by means of relations, it is also possible to extend the inferential potential of ontologies by means of embedding constraints or inference rules. By this means additional knowledge can be embedded in and imparted by the ontological ‘contexts’. In addition to providing an enhanced basis for dynamic implementations of ‘context’, this approach raises the prospect of using a cohort of intelligent agents to co-develop, with the human user, a design requirement during an elicitation episode.

The three types of approach can be overlaid onto the model of human-machine inference, indicating how sharing knowledge can progressively move the inferential burden between humans and machines rightwards towards the ideal.

## 5 A Prototype Design Requirement Elicitation Tool

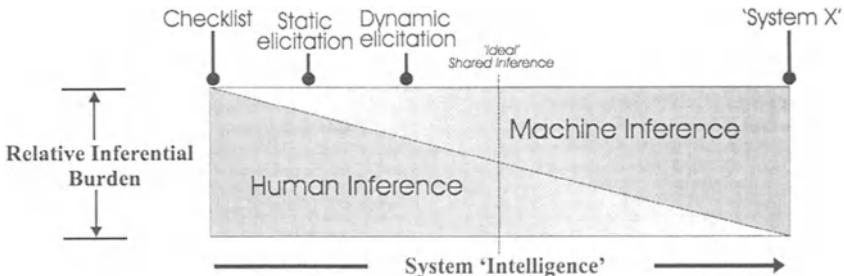
The discussion in Section 2 clearly identifies for representation, shareable concepts – which will be identified by labels – and associations or relations between the concepts. The method chosen here for representing concepts and associations is a simple graph, consisting of nodes (representing the concepts/labels) and arcs representing a direct relation. The relations adopted between the concepts, principally based on meaning, are: parent, child and co-activee, elaborated as follows.

**Parent.** Each concept, excepting the graph root concept, will have a single parent.

**Child.** Any concept may have one or more child concepts. These may be associated by the logical relations: AND, OR and XOR. For example, AND implies that all child concepts are logically entailed if the parent concept exists.

**Co-activee.** This is a pragmatic relation that allows logical entailment of remote concepts (those placed in different graphs).

A set of labels and their relations as defined above together form a *domain specification* which constitutes the domain knowledge for the domain to be considered.



*Figure 6.* A comparison of the suggested context-based design requirement elicitation mechanisms compared in terms of inference sharing.

### 5.1 DEVELOPING THE DOMAIN SPECIFICATION

As a focal point for the domain content of the ontology for this exploratory implementation, the domain of machine motion was chosen. That is to say, of interest in developing the ontology were all the terms that might be used in the description of the motion of a machine.

The domain scope was based on the following assertions relating to function:

1. When considering some of the tasks that a 'machine' can carry out, the following basic activities can be performed:
  - moving an object,
  - acting on an object (e.g. when clamping or drilling),
  - acting on itself (e.g. in moving a mechanism).
2. The requirements that a mechanical system may be required to meet can be expressed in terms of the function (the purpose) of the machine and in terms of the action of the machine.
3. The actions that a machine can perform can be redescribed or 'chunked' as functions.
4. The manner of describing these actions and functions is unrestricted (given the generative nature of English).
5. The set of concepts invoked when describing the function or actions is restricted or, at least, a restricted set of concepts can be identified that can usefully describe them.

### 5.1.1 *The lexicon of terms (concept labels)*

The starting point for developing a lexicon to describe this domain was the lexicon developed in the authors' earlier work (Darlington and Potter 1998) relating to the automation of configuration design. From this have been taken those words which were appropriate for this particular purpose. Additional words have been adopted as seemed appropriate in order to increase the richness of the expression. In addition the existing general lexical ontology WordNet (Miller et al. 1990) has been used as a principal source of words and definitions for this specialist lexicon. In particular, WordNet was used as a means of eliminating synonyms. Development of the lexicon was carried out iteratively with development of the conceptual structure in the form of a set of graphs.

### 5.1.2 *The graphs*

To control graph complexity and facilitate administration and maintenance, the domain is described using a number of graphs, although, in principle, a single graph might be used. Each graph relates to a different descriptive dimension of the domain as a whole, viz:

- Function
- Motion
- Force
- Attribute
- Control

The *function* graph represents the entry-point conceptually for the domain, in that it is as a function that the embryonic design requirement is expressed. Requirements described in terms of function can also be described as activity. These aspects of the domain are contained (redescribed at a different level) within the *motion* and *force* conceptual structures.

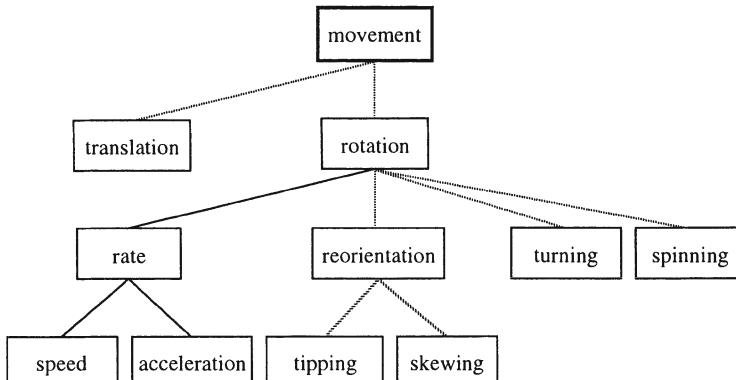
A separate graph, *control*, contains concepts relating aspects of controlling the activity of the solution system. The *attribute* structure contains concepts and associations relating to physical objects and their attributes. A fragment of the *motion* graph is illustrated as an example in Figure 7.

### 5.1.3 *Implementation issues*

In order for the conceptual structures discussed above to be operationalized, an algorithm has been developed to interpret the data expressed as a domain specification and implemented in a prototype computer programme. The implementation consists of:

1. A representation of the domain specification as a conceptual structure.
2. A method for assessing association-relation decision criteria.
3. A representation of the current episode knowledge as a conceptual structure.
4. A method of adding to the current episode conceptual structure as concepts are accepted.
5. A method of eliciting and accepting information from the user.

In addition to containing knowledge about the domain concepts and relations, provision is implemented for numerical and text values to be associated with specific concepts. This allows the system (the designer) to prompt the user (the customer) to provide these values when appropriate, and for these values to be output together with the design requirement concept structure. Together these provide what can be thought of as a Design Requirement Record. From this record can be extracted those elements that are useful in a given context as the Design Requirement itself.



*Figure 7.* A fragment of the motion graph, showing concept labels, AND relations (solid lines) and XOR relations (hatched lines).

#### *5.1.4 A computer-supported design requirement elicitation episode*

The initialized system can be thought of as taking the part of the designer, about to embark on a information-gathering episode. At this stage the current knowledge consists of the general knowledge about the domain, but none about any specific episode. The process that ensues can be seen as one of progressively – by inference and by eliciting user-input – eliminating from consideration irrelevant elements of the closed domain.

The episode is initiated by the system prompting the user to enter a 'key concept': the principal functional requirement that the user desires to be satisfied by the design. This corresponds to one of the concepts in the domain specification *function* conceptual structure.

The system 'activates' this concept by placing it as the first entry in a concept-list. This represents episode-specific knowledge. The system now looks for the next most closely related concept to the one selected. This is activated and placed in the concept list. This process is repeated. At each stage the current knowledge – the domain knowledge and the growing episode-specific knowledge – is searched, which allows inferences about the next nearest concept to be considered. The chaining process continues until the choices presented to the system cannot be reconciled using the current knowledge state (in which case it prompts the user to provide the information necessary to continue) or until all concepts that might be appropriate to the current episode have been activated (in which case the elicitation episode is at an end). During the elicitation, the system will prompt the user for numerical or text data, where the domain specification prescribes that a given concept has such an associated value.

Output from the system consists of reporting the content of the episode-specific knowledge in the form of a set of concepts, ordered in a relational hierarchy. In addition, numerical or textual data, associated with particular concepts, are attached. This constitutes the full design requirement record, which can be modified or augmented manually to form the design requirement. A fragment of a record and the resulting design requirement is shown in Figure 8.

## 6 Conclusions

The way that humans share experiences of the world is central to the way that they build up intersecting internal models of the external reality, which is represented by conceptual schemata. Agreement on the intersections allows the use of labels, which not only are the means by which communication occurs, but also point to the underlying concepts. Communication between individuals is assisted by the grouping of concepts together which provide context, enhancing inference and constraining the train of thought.

These insights suggest a method by which domain knowledge at a low level of resolution might be identified and usefully mapped into artificial 'contexts', analogous to those used by humans. This method has been adopted using an ontological approach for the purpose of designer support in the elicitation of the design requirement, and a number of applications

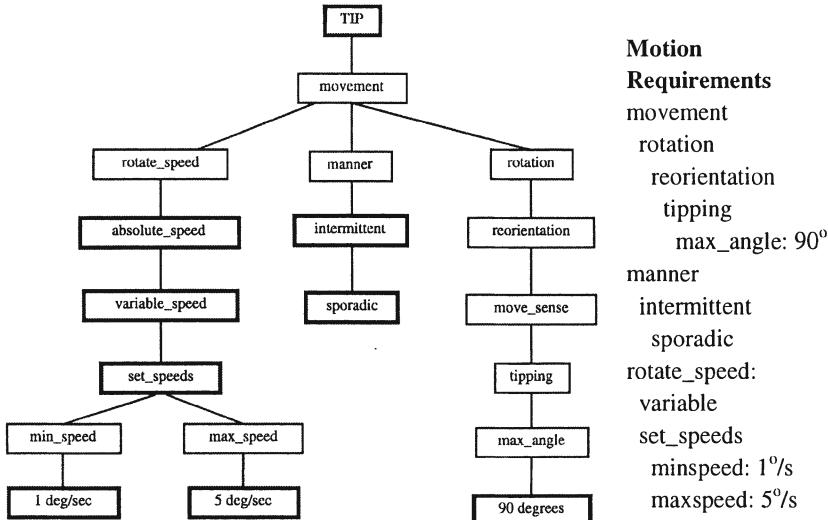


Figure 8. A fragment of the design requirement report. concepts shown in bold outline are those that were selected by the user from choices presented by the system; others were chosen by the system. The text output forms the basis for the design requirement itself.

suggested to illustrate the way in which communication between humans and machines can be improved for this important phase of conceptual design. A prototype implementation has been presented illustrating just one possible approach to developing a dynamic design requirement elicitation support tool.

## References

- APTICOTE-ISIS Surface Engineering Selection Expert System, URL: <http://www.poeton.co.uk/fr-pands.htm>
- Bath Engineering Design Group: 2001, *Design Requirements Guidelines*, Department of Mechanical Engineering, University of Bath.
- Cross, N: 2001, Can a machine design? *Design Issues* 17: 44-50.
- Darlington, M J and Potter, SE: 1998, Engineering design requirements formalization: the development of a constrained natural language and an elicitation scheme formalizing aspects of the design requirement, *Internal Report No. 041/98*, Engineering Design Centre in Fluid Power Systems, University of Bath.
- Darlington, M, Culley, SJ and Potter, S: 2001, Using domain knowledge to support design requirements elicitation, in S Culley, A Duffy, C McMahon and K Wallace (eds), *Proceedings of 13th International Conference on Engineering Design (ICED 01)*, Professional Engineering Publishing, Edmunds, pp. 83-90.
- Durney, LJ: 1984, *Electroplating Engineering Handbook*, 4<sup>th</sup> Edition, Chapman Hall, London.

- Edmonds, B: 1997, A simple-minded network model with context-like objects, *Proceedings European Conference on Cognitive Science*, Department of Computer Science, University of Manchester, Manchester, pp. 180-184. <http://www.cpm.mmu.ac.uk/cpmrep15.html>
- Gabe, DR: 1972, *Principles of Metal Surface Treatment and Protection*, Pergamon Press, Oxford.
- Genesereth, MR and Nilsson, NJ: 1987, *Logical Foundations of Artificial Intelligence*, Morgan Kaufmann, San Mateo, CA.
- Gruber, T: 1993, A translation approach to portable ontologies, *Knowledge Acquisition* 5(2): 199-220.
- Hales, C: 1993, *Managing Engineering Design*, Longman Scientific and Technical, Harlow.
- Lenat, D: 1998, *The Dimensions of Context-Space*, Cycorp. URL: <http://www.cyc.com/publications.html>
- Miller, GA, Beckwith, R, Fellbaum, C, Gross, D and Miller, KJ: 1990, Introduction to WordNet: an on-line lexical database, *International Journal of Lexicography* 3(4): 235 - 244. <ftp://ftp.cogsci.princeton.edu/pub/wordnet/5papers.ps>.
- Noy, N and McGuinness, DL: 2000, *Ontology Development 101: A Guide to Creating your First Ontology*, Stanford Medical Informatics Technical Report No. SMI-2001-0880. URL: [http://smi-web.stanford.edu/pubs/SMI\\_Abstracts/SMI-2001-0880.html](http://smi-web.stanford.edu/pubs/SMI_Abstracts/SMI-2001-0880.html)
- Ogden, CK and Richards, IA: 1923, *The Meaning of Meaning*, Harcourt, Brace and World, New York.
- Pahl, G and Beitz, W: 1996, *Engineering Design – A Systematic Approach*, 2<sup>nd</sup> Edition, Springer-Verlag, London.
- Protégé: 2000, *Ontology Editor and Knowledge Acquisition Tool*, URL: <http://protege.stanford.edu/index.shtml>.
- Pugh, S: 1991, *Total Design: Integrated Methods for Successful Product Engineering*, Addison-Wesley, Wokingham.
- Stillings, NA, Feinstein, MH, Garfield, JL, Rissland, EL, Chase, CH, Weisler, SE and Rissland, EL: 1995, *Cognitive Science: An Introduction*, 2<sup>nd</sup> Edition, MIT Press, Cambridge, MA.
- Ullman, DG: 1997, *The Mechanical Design Process*, 2<sup>nd</sup> Edition, McGraw-Hill, NY.
- Ulrich, KT. and Eppinger, SD: 1995, *Product Design and Development*, McGraw-Hill, New York.
- Wootton, AB, Cooper, R and Bruce, M: 1997, Requirements capture: where the front end begins, in A Riiahuhta (ed.), *Proceedings of 11<sup>th</sup> Int. Conference on Engineering Design (ICED 97)*. Tampere University of Technology, pp. 75-80.

## CASE-BASED DESIGN FACILITATED BY THE DESIGN EXEMPLAR

*Retrieval and archival of 3D mechanical engineering knowledge*

JOSHUA D SUMMERS, ZOE LACROIX AND JAMI J SHAH  
*Arizona State University*  
USA

**Abstract.** Case Base Design (CBD) systems aid designers in addressing new design problems by allowing the designers to search previously stored valid designs, retrieve the ones that satisfy or nearly satisfy the problem specifications, and adapt the cases to the new requirements. Many existing systems use ad-hoc functions and algorithms to perform these tasks. The design exemplar is introduced as a tool to index, query, and transform design cases in CBD. Unlike other approaches, design exemplars rely on a generic domain independent representation of entities and their relationships and are evaluated by a general algorithm based upon constraint validation and satisfaction. A retrieval mechanism of archived 3D knowledge as facilitated by the design exemplar is described. Sample sessions for the retrieval tool are provided.

### 1. Introduction

Design problems may be divided into two basic classes: routine and non-routine design (Gero 1994). Routine design, in terms of knowledge, is defined as when the engineer knows what type of knowledge is needed in the design. When the designer knows what knowledge is needed, the designer may reuse previous designs containing the same or similar knowledge. Knowledge is a set of facts, data, and relationships that is associated with semantic meaning. Non-routine design may be divided into innovative and creative design. Innovative design generates new designs from a known set of designs, yet differs from the existing designs in a substantial way. The class of design considered in this research is primarily routine design. A further division of design is the distinction between variant and generative design. Variant design modifies a previous design to

satisfy new problem requirements. The modification may be through parametric sizing, reconfiguration, or other types of adaptation tasks. Adaptation of past design solutions is a type of variant design. Generative design creates a new design solution from scratch. While engineering design knowledge may be used to create the new design, the design solution is created without any design template upon which to work. The scope of this paper focuses on routine variant design.

One observation of engineering designers is that they generally tend to get better with experience. Designers gain experience about which solutions are effective and about the scenarios used to satisfy different problems. When a design problem is provided to an experienced designer, the designer first searches for a similar past design. This proves beneficial for two reasons: redesign generally is less time consuming than initiating a design from scratch and the performance of the base design is known. This performance may include design objectives, economic requirements, or manufacturing concerns. Reuse of design information generates significant cost savings cutting product development time by as much as fifty percent (Baya and Leifer 1995).

Case-based design consists of three successive tasks: *retrieve*, *evaluate*, and *adapt*. The retrieve task finds past design cases that may be appropriate in the new design scenario. The evaluation task validates these cases with respect to the new design requirements. The adaptation task modifies and combines the cases to fully satisfy the design requirements of the new problem. In general, case based systems have three basic components: a *case base*, a *case search engine*, and an *adaptation module*. Past designs are indexed and stored in the *case base*. The *case search engine* allows the designer to express retrieval queries. The *adaptation module* encapsulates a large amount of design knowledge to divide or combine retrieved cases, and to validate the combinations of previous designs to generate a new design.

In this paper, the concept of the design exemplar as a tool to query designs is offered. The design exemplar has been used for representing many types of engineering knowledge, including rules, procedures, and features (Bettig et al. 2000). Exemplars are designed to query (retrieve), validate (evaluate), and modify (adapt) design models. It is clear that these three tasks found in CBD are the same tasks provided by exemplars. This parallel between CBD technology and exemplar technology is an important relationship that may be leveraged in the use of exemplars in a design re-use system. In addition, exemplars extend the expressive power of retrieval queries by allowing users to ask complex queries as opposed to Boolean expressions of valued or constant characteristics in other pre-indexed CBD systems. Some key characteristics of design exemplars that are useful in

querying and indexing are that: they operate at the data level, users define or reuse exemplars as they are needed for specific situations, the tasks are generic and embedded in data structure, they are capable of explicit and implicit reasoning, declaratively defined, and they are capable of representing different types of knowledge (features, rules, procedures).

## 2. Case-Based Design

Case base design (*CBD*) systems are intended to assist designers in various stages of design by providing them access to previous valid designs that satisfy or nearly satisfy the defined requirements, functions, or characteristics and by modifying the retrieved design cases to satisfy these requirements. Examples of case based systems in engineering include: DDIS (Wang and Howard 1991), CADRE (Hua et al. 1992), KRITIK (Goel and Chandrasekaran 1989), CADET (Sycara and Navinchandra 1989), HICAP (Munoz-Avila et al. 1999), and FABEL (Schaaf 1996).

### 2.1 CASE REPRESENTATION AND ORGANIZATION

Representation and organization of cases are fundamental issues in the development of a case based system. The design of the case base constrains the ability and the efficiency of the *CBD* to retrieve, evaluate and adapt cases. Maher et al. (1995) identify three major issues associated with case representation: design case content, representation paradigms for case memory organization, and user view of the cases.

Design case content is the design data and documents available within the case. It may include strings of parameters and values, solid models, collections of design tasks undertaken to achieve the design solution, collections of all design documents generated in the process, etc. Content of design cases must be processed (manually or automatically) to capture their characteristics and properties. These characteristics and properties are added to cases through annotations. Storage and performance issues in memory organization are addressed by providing the best organization of design cases, usually with respect to their similarities. Finally, users view and access the cases with their understanding and the specifications of their design problem. Some tools exist for authoring cases interactively, such as CASCADE (McKenna and Smyth 2001), CBR Works (Empolis Knowledge Management 2001), or k-Commerce (Inference Corporation 2001).

The goal of indexing cases in a case base is to achieve good performance (Bareiss and Porter 1987; Kolodner 1984; Feret and Glasgow 1997). The indexing problem poses some inherent trade-offs between the desire to provide a flexible problem definition approach and an efficient retrieval

algorithm. As the size and complexity of the case base grows, the indexing scheme provides quicker access. However, the scheme may provide only a limited view of each of the design cases, a view that may not be entirely compatible with new problems. Indices should be chosen such that they aid in predicting the solution, that they be abstract enough to span several different cases, and that they be concrete enough to be applicable in future problems (Kolonder 1991). Much of the effort in developing indices for case-bases focuses on domain specific classifications.

Most of the CBD approaches rely on indexing design data with respect to annotations capturing their attributes. Selecting the properties and characteristics to represent design data has significant consequences. Indeed, in existing systems selected attributes constrain the level of expression in the retrieval phase. Different techniques and approaches are used to perform indexing: generated design signature (Elinson et al. 1997), memory based automated indexing (Lee et al. 1997), clustering (Schaaf 1996), object-oriented approach (Perera and Watson 1996), or feature based (Krampe and Lusti 1997). These approaches enable the indexing of the entire case and do not offer additional granularity. A design that contains a boss and a hole will be indexed with two attributes *boss* and *hole*, both valued as *true*. However, the geometry of the design itself is generally not indexed and limited access is provided directly to the portion of the geometry that the boss or hole features represent.

KRITIK uses functional models based upon the structure-behavior-function (SBF) representation of Goel and Chandrasekaran (1989). This representation links functions with realized components in the design models. The user defines a query based upon the function structure desired (Murdock et al. 2001). KRITIK queries the case bases with respect to the function structures that are identified for each case. Thus, for each case, there are at least two models associated with it: the functional representation and the component structural representation. KRITIK provides for problem definition (function structure), case retrieval (based upon SBF), case adaptation (based upon generate and test using system dependent methods), and case verification (based upon the constraints and modified model). Case storage indexes new cases and places the cases in the case base structure.

Indexing techniques have impact on all three major activities of CBD. Some approaches do not use the indexing to optimize the retrieval query evaluation. For a flat case organization structure, as seen in Figure 1, all attributes are added to the cases as annotations and all cases are parsed when evaluating each retrieval query. The performances of the evaluation

of retrieval queries can be dramatically improved by using the indexing for the organization of cases in the base.

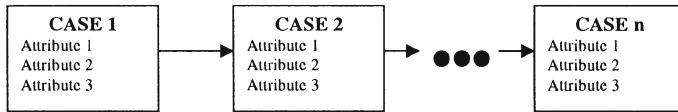


Figure 1. Flat structure (Maher et al. 1995)

A hierarchical organization of cases, such as illustrated in Figure 2, permits the evaluation of retrieval queries against a tree. Each evaluation corresponds to a path in the tree and leads to a set of cases. The lack of granularity influences the design case representation and the memory organization. Indeed, an organization with respect to components of cases as opposed to the entire case may capture better the similarities among cases, resulting with improved performances of the approaches.

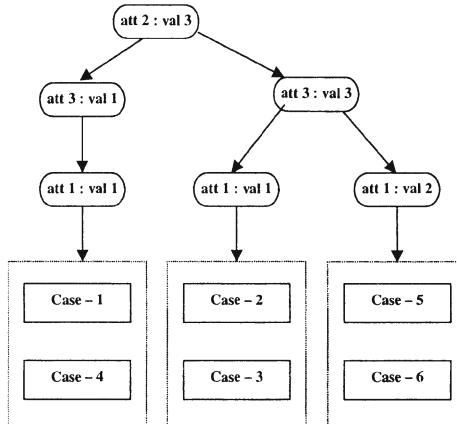


Figure 2. Tree structure (Maher et al. 1995)

## 2.2 CASE RETRIEVAL

Some examples of query interfaces that rely heavily upon indexed systems include expert driven questionnaires in Conversational CBD (Aha and Breslow 2001), tabular (Amen and Vormacka 2001), frame based (Bilgic and Fox 1996), textual (Chang et al. 1996), or function structures (Murdock 2001). No existing approach appears to offer a real query language to retrieve cases for engineering design cases as represented by the geometry and topology of a finished design. Retrieval queries usually are Boolean

expressions of terms consisting of valued attributes as represented through the indices. FABEL provides user specified weightings to create attribute functions for assessing the appropriateness of retrieved cases (Schaaf 1996).

In most approaches, designers search cases by expressing Boolean expressions of valued attributes. The attributes used in the expression of a retrieval query correspond to the indices of the case base. A designer searches design cases for designs that satisfy or nearly satisfy the problem requirements. The evaluation process of a retrieval query allows two different semantics: *exact match* or *close match*. A case exactly matches a retrieval query when its annotated properties exactly correspond to all values expressed in a query.

An example of the need for retrieving close matches in a CBD system is that in embodiment design of a gear based transmission reducer. The design specifications may require a design that reduces the speed from 600 rpm to 200 rpm. These values for the attributes may not return an exact match from the case base. However, the implicit relationship of a reduction 3:1 may match a past case that has values for 3600 rpm to 1200 rpm. The design configuration of this case may be appropriate for the specified problem, despite the need to adjust the sizing. Thus, this case may be appropriate as a foundation for the design of the new solution. The close match semantics relies on different techniques to capture similarities among designs.

The CBD system of Krampe and Lusti (1997) for information system design calculates two types of similarities in the design and the cases. The global similarity is found to compare the combination of features present in the queried case with the combination of features in the design. The second is a local similarity, which compares the design features of the case with the design features of the design. Similarity comparisons are required to retrieve matches.

Zdrahal and Motta (1996) have developed and implemented four basic CBD algorithms focused on retrieval of design cases: simple match, subsystem sensitive match, design cost estimation, and adaptation cost optimization. These algorithms focus on methods for selecting from possible retrieved matches. The NIRMANI (Perera and Watson 1996) system retrieves cases in two levels. The primary retrieval process uses high level information while the secondary retrieval process finds cases based upon the more problem specific criteria. These cases are then adapted for use by the user. This is an example of a system without automated adaptation.

Generally, it is unlikely to find an exact match for the design (Elinson et al. 1997). Portions of the knowledge base set of designs are needed to generate a plan that will satisfy the desired design plan. Therefore, a system

ought to be capable of retrieving portions of designs based upon matching pieces of the higher-level signatures. This system slices designs using reasoning about associated features. The dependencies may include: intersecting features, tolerances of one feature based upon data of a second feature, features in close proximity, same tool used in two features, or same approach direction used in two features. This approach is used to index slices, which may be retrieved by the system if the slice signatures may be found in the original design signature.

The retrieval strategy suggested by Schaaf (1996) is designed for situations when static case structure and indexing is not acceptable. These situations may include: cases are sought with different viewpoints because of complexity, changing the frame of reference is supported by knowledge stored in cases, changing the frame of reference leads to different aspects of the cases, comparing similarities between cases is possible, and dissimilarity between the case and the query reduces the search space.

The FABEL system represents the case base as a network connecting cases through weighted attributes. These weightings are used to define the distances between cases. The user defines a query (based upon the attributes defined *a priori* in construction of the case base) and a weighting function. The retrieval algorithm uses a bounded range for the different attributes to retrieve cases, pruning neighboring cases where not retrieved. In this manner, the retrieval algorithm finds similar cases (fishes) and culls the case base of those cases that are distant or dissimilar from the retrieved cases (shrinks). The retrieval algorithm is intended to retrieve different cases, not cases that are so similar that they do not add significantly to an understanding of the problem or solution.

These techniques not only provide semantics for close matches, but they are used for the adaptation of retrieved cases. The internal adaptation by the case base modeler is done to modify the cases into an acceptable design based upon the use of integrity rules or reference model rules. Integrity rules are used to assure a consistent design at the meta-model level. For example, an integrity rule can verify that a casting does not have an undercut or a V-Belt design has at least two pulleys. Integrity rules are domain specific and capture design knowledge. Up-gradable knowledge and static knowledge classify the case knowledge (Perera and Watson 1996). The up-gradable knowledge is typically considered to be the design cases and may be increased with the development of more design cases. The static knowledge is the rules in the knowledge base that may only be modified at the system level. The adaptation process begins with the retrieved cases and continues with the application of the static knowledge. Regardless of the adaptation approach, the resulting adapted case must be

evaluated against the retrieval criteria to ensure that no changes or modifications were applied that would invalidate the design.

### 3. Design Exemplar in CBD

Exemplars are based upon the basic observation that all parametric and geometry-related queries about engineering designs deal with some characteristic of the design (Bettig et al. 2000). These characteristics derive from a pattern of entities and constraints found explicitly or implicitly in the design artifact. Exemplars represent these characteristics as patterns of topologic, geometric, algebraic, and semantic relationships. They constitute a query language upon their geometric representation. Exemplars express queries whose input and output are geometric entities and constraints. Entities and constraints are mathematically grounded, providing greater confidence in manipulation of the design problem. Constraint solvers perform the evaluation of exemplars against design models. Existing constraint solvers include numeric solvers, geometric solvers, topologic solvers, or semantic solvers. The Geometric Constraint Solving System (Bettig et al 2000) offers an integrated constraint solving approach for geometric and topologic design problems. This system uses a general algorithm to apply different constraint solvers to the design problem. These constraint solvers include commercial solvers (DCM), open source solvers (Newton-Raphson), and in-house solvers (Geometric Measures Solver). Currently, there are about ten solvers that are available, but new solvers may be added as needed.

#### 3.1 EXEMPLAR BACKGROUND

The exemplar is composed of two pairs of orthogonal components: match/extract (used for validation) and alpha/beta (used for transformation) as represented in Figure 3 (Bettig et al. 2000). Each of these components is composed of bi-partite graphs of entities and constraints. The entities include numeric parameters, points, lines, curves, planes, surfaces, or directions. The constraints include equations, distance, identification, angle, coincident, incident, or boundary. While each instance of an entity or constraint is exclusively found in either the match or extract exemplar components, an individual instance of an entity or constraint may be found in both the alpha and beta components. This is used for propagation of changes to the design model upon which the exemplar is applied.

Part of the pattern (the *match* part) corresponds directly to entities specified in the design model and part of the pattern (the *extract* part) expresses relationships. These relationships must hold true in order for the

characteristic of the *matched* part of the design to be valued true, thus facilitating reasoning about the matched part. The extract part of an exemplar is used for defining variables, binding variables, or capturing implicit properties satisfied by the design model but not expressed explicitly. The general algorithm of applying the exemplar is to first match the alpha match entities and constraints of the exemplar with the entities and constraints of the design model. If a match is found, then the extract entities and constraints of the exemplar are used to reason about the design model, extracting information after satisfying the extracted relations. The design model entities and constraints are fixed during the constraint solving process.

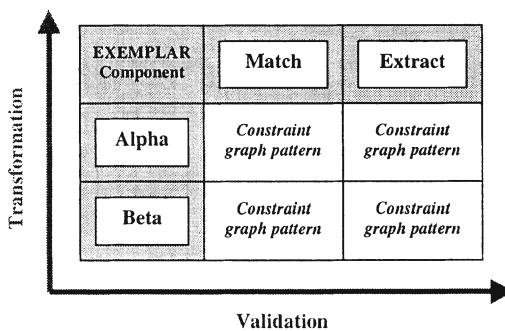


Figure 3. Components of an exemplar (Bettig et al. 2000)

In addition to the ability to match and extract information and characteristics from a design model, exemplars have the ability to modify the design model from an *alpha* state to a *beta* state. The optional *beta* component allows exemplars to represent the situation when a design does not have the characteristic, thus facilitating modifications of the design. In Figure 3, the two axes, transformation and validation, illustrate two of the main operations for exemplars (modify and verify). To query a design, the Match/Alpha constraint graph pattern is all that is required. Exemplars may be composed of any combination of the four constraint graph patterns, depending upon the desired functionality of the exemplar. Exemplars may be used to find characteristics in a design (casting undercut surfaces), modify characteristics in a design (widen thin walls), or validate a design (ensure an acceptable v-belt design).

The parallel between CBD technology (search, evaluate, and adapt) and exemplar technology (query, validate, and modify) is an important relationship that may be leveraged in the use of exemplars in a design re-use

system. This section discusses how exemplars may be employed in Case-based design in the development of a computer aided embodiment design reuse system. In developing a construct for the cases to be used in the exemplar based design re-use system, several issues need to be investigated. As described earlier, one option for developing a case structure includes using a collection of key parameters (attributes), values, and relations. Exemplars can express most of the properties expressed in indexing approaches for case base reasoning. The geometric properties can be easily captured in terms of entities and constraints, when additional annotations can be added to each case expressed in terms of entities and constraints.

### 3.2 DESIGN EXEMPLAR EXAMPLE

Figure 4 illustrates a boss model. The explicit entities are solid lines and include a cylinder bounded by two curves, each curve bounding another surface, and all the surfaces bounding a volume. The extracted information is illustrated in dashed lines and includes a line that is concentric with the cylinder, points that are incident on the line and on the top or bottom surface, the distance between the points, and the radius of the cylinder.

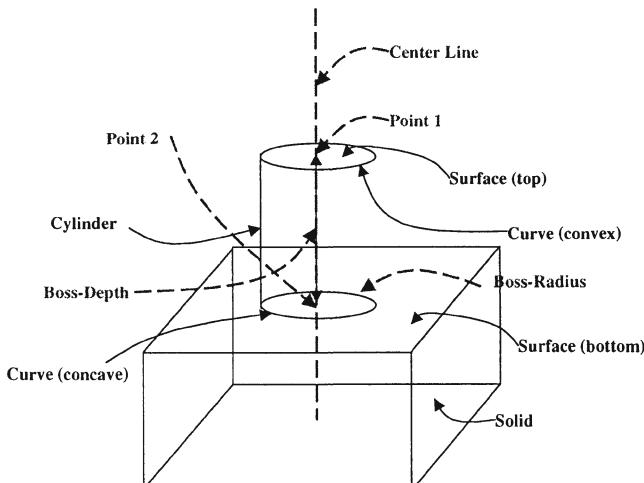


Figure 4. Boss description

As an example of a design exemplar, the property *has a boss* can be captured by the exemplar given in Figures 5 (in text format). The boss exemplar is satisfied when a boss exists in the design case. No transformation is expected, thus only an alpha condition is represented in the exemplar.

```

Alpha Match:
 Volume "Solid";
 Plane "Top Surface";
 Cylinder "Boss Surface";
 Plane "Bottom Surface";
 Curve "Top Curve";
 Curve "Bottom Curve";
 Boundary (Solid, {Top Surface, Boss Surface, Bottom Surface});
 Boundary (Top Surface, {Top Curve});
 Boundary (Bottom Surface, {Bottom Curve});
 Boundary (Boss Surface, {Top Curve});
 Boundary (Boss Surface, {Bottom Curve});
 ID "Convex" (Top Curve);
 ID "Concave" (Bottom Curve);

Alpha Extract:
 Point "P1";
 Point "P2";
 Line "boss_CL";
 Parameter "boss_radius";
 Parameter "boss_depth";
 Incident (Top Surface, P1);
 Incident (Bottom Surface, P2);
 Incident (boss_CL, P1);
 Incident (boss_CL, P2);
 Radius (Boss Surface, boss_radius);
 Distance (P1, P2, boss_depth);
 ID (boss_CL);
 ID (boss_radius);
 ID (boss_depth);
 ID "part_with_boss" (Solid);

```

*Figure 5.* Text representation for boss

Exemplars may be made specific or general depending upon the type of entities used in the declarative representation. In this situation, the boss height is determined from finding the distance between the points found on the centerline of the boss that are incident with the top surface and bottom surface. This height may also be found by determining the distance between points incident on the top surface and bottom surface that are concentric with the cylindrical boss. Either manner of extracting the boss height is acceptable and will extract the same information. Exemplars may be made that are generally semantically equivalent (from general to specific) or truly semantically equivalent (using different extract entities and constraints to extract properties).

In addition to selection of the entity types to refine the exemplars, the parametric relationships found in the model may be used to further limit the scope of the exemplar. If the exemplar developer knows that the boss has been parameterized and constrained to have a distance between the top face and the bottom face, this constraint may be moved from the extract portion

to the match portion of the exemplar. This refines the initial pattern-matching step of the general exemplar application algorithm.

### 3.3 SEARCHING CASES BY “EXEMPLARS”

Exemplars, simple and complex, may be used to query the explicit data of the case models and reason about implicit characteristics of the cases. Exemplars offer a significant improvement than retrieval queries in traditional CBD systems. In addition to simple Boolean expressions of pre-selected attributes, users can express real queries against the design.

An example based on the combinations of two properties *presents a boss* and *presents a through-hole* illustrates this additional expressive power. Consider three cases satisfying these two properties independently. In the first case, the boss and the hole have no interaction, Figure 6a. In the second and third cases respectively, boss and hole interact on the same solid, Figure 6b, and on different solids, Figure 6c. In a standard approach, with an indexing relying on the properties through-hole and boss, a user searching the case base for the feature “through-hole in boss”, would only be able to ask a retrieval query as follows: *through-hole=true AND boss=true*. The retrieval query would return all three cases. The designer would have to complete the search manually in order to only select the case of Figure 6b.

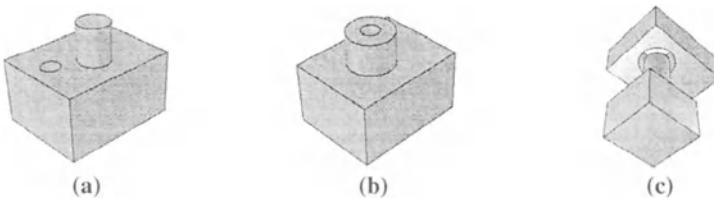


Figure 6. Hole and Boss Models

Combination of exemplars into complex queries goes beyond simple semantic topological features such as described above. They allow the expression of geometric queries. For example, consider a user searching a case base to retrieve mating boss and hole found in an assembly. Standard feature recognition of the geometric features would be limited to assist the search. Again, a standard approach would only assist in indexing all cases containing a hole and a boss, and all three cases would be retrieved. However, a complex exemplar query may be used to find the all boss and hole mating pairs of unique solids.

The complex query is defined with respect to six exemplars: (1)  $\text{Ex}_{\text{boss}}$  an exemplar to match all boss features on a solid and extract the cylindrical axis, boss height, and the diameter; (2)  $\text{Ex}_{\text{hole}}$  an exemplar to match all hole features on a solid and extract the cylindrical axis, hole depth, and the diameter; (3)  $\text{Ex}_{\text{radial\_mating}}$  an exemplar to match extracted diameters; (4)  $\text{Ex}_{\text{longitudinal\_mating}}$  an exemplar to match extracted depths and heights; (5)  $\text{Ex}_{\text{center\_line}}$  an exemplar to match extracted center lines to extract the concentricity of the axes; and (6)  $\text{Ex}_{\text{two\_solids}}$  an exemplar to match two unique solids in the model. This query may be represented as a simple statement found in Figure 7.

$\text{Ex}_{\text{boss}} \text{Ex}_{\text{hole}} \text{Ex}_{\text{radial\_mating}} \text{Ex}_{\text{longitudinal\_mating}} \text{Ex}_{\text{center\_line}} \text{Ex}_{\text{two\_solids}}$

Figure 7. Compound Query

This statement does not convey the interrelationships between the exemplars, illustrating the limitation of simply Boolean operations in constructing complex geometric and topologic queries for retrieval of past cases. This approach would require evaluation against sequentially retrieved cases. The first is that all cases with a boss are retrieved. The second would retrieve all cases that have holes from the selected boss list. This list of cases is further refined to determine the radial clearances between the hole and the boss. Information needs to be added to the model to specify the radius of a boss and hole for this exemplar to extract any information. Likewise, step four and five are dependent upon information added to the model that relates the depth and height and the center lines. The final step would narrow the cases from the list down to those that contain two distinct solids. This incremental approach for refinement of the retrieved lists based upon applying the simplistic exemplars.

Exemplars can also express topological relationships such as distinguishing two solids in an assembly. Consider a user searching a case base to retrieve mating boss and hole mating conditions between two different solids. The previous complex exemplar can be combined to an exemplar that ensures that the solids of the boss exemplar and the hole exemplar are not the same, Figure 6c.

Consider the two features of a cylindrical through hole and a cylindrical boss. It may be desired to determine, in an assembly of two parts, the radial clearance and the longitudinal clearance between a mating of these two features. Rather than identifying the mating features interactively, an exemplar network may be created that using some standard exemplars from a catalog.

These exemplars may be a boss exemplar, a hole exemplar, a different solid exemplar, Figure 8a, a center line exemplar, Figure 8b, a radial clearance exemplar, Figure 8c, and a longitudinal clearance exemplar, Figure 8d. These exemplars may be connected in the network and compiled into a new composite exemplar. In this manner, the procedural application of the exemplars as illustrated in the directed graph is transformed into a declarative representation.

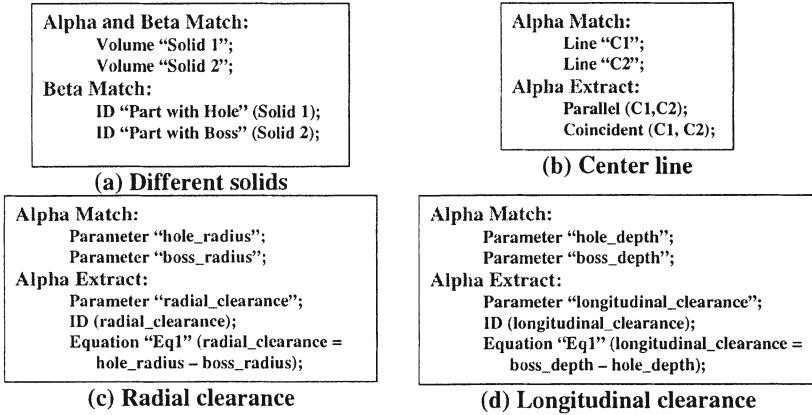
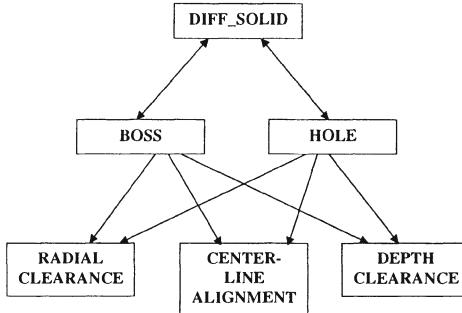


Figure 8. Validation Exemplars

The exemplar designers may create a node from an exemplar by identifying the entities that are to be used as inputs or outputs. Additionally, biputs are introduced for use when knowledge of the application of the node is not fore known. This node has the solid identified as an input and the boss center line, boss height parameter, and boss radius parameter, identified as the outputs.

In combining these exemplars into a network, Figure 9, the match and extract portions of the exemplars may be linked, creating a directed graph and building into the network an appropriate solution sequencing through the graph. A composite exemplar may be compiled from the network that may be applied to design models. In this manner, designers may interactively construct and compile networks or base exemplars. The compiled exemplar has no procedural solving steps built into it, except for the match and extract portions of all exemplars. Generally, match and extract entities are modified to agree depending upon the direction of connection. Further, the entities types are modified depending upon precedence of connection, allowing for curves to be connected to circles.

The network itself is not evaluated against a design model, rather the compiled exemplar is evaluated.



*Figure 9.* Boss/Hole mating exemplar network

Exemplars offer a better granularity in the query process than current implementations. Exemplars offer a wide variety of possible queries against cases. First, they allow the expression of retrieval queries as in a standard representation, as long as the properties they capture are expressed in the case as entities and constraints. They can express the following: explicit valued property (with the alpha-match component), implicit valued property (with the alpha-extract component), and Boolean expressions of valued properties (by combination). In addition, exemplars provide users with a much wider expressive power. Retrieval exemplar queries can express the following types of knowledge: rules, features and parametrics, and design procedures. The levels of abstraction at which exemplars operate include semantic, algebraic, topologic, and geometric, integrated into a single representation.

Contrary to other approaches, retrieval exemplar queries can be evaluated against indices of attributes or cases directly, even if the cases are not associated with any indices. Indices need only be used for optimizing the performances of the retrieval phase.

#### 4. Exemplar Based CBD system

The Exemplar Based Design Retrieval system, Figure 10, consists of four basic modules: Query Definition Module (QDM), Retrieval Module (RM), Base Management Module (BMM), and the Generic Geometric Constraint Solver System (GGCS). These four modules combine to provide the functionality for retrieving past designs from selected case bases depending upon user defined queries. Further, the retrieved cases may be stored in a

separate directory for future querying or modification. The user defines a query through the QDM. The user then selects a case base to search against through the BMM. The CMM feeds single cases from the selected base to the GGCS as the QDM supplies the queries. The GGCS determines if the characteristic as defined by the query exists in the case and returns a response to the CRM. The CRM maintains a list of matched cases from the case base. These matched cases then can be saved in existing case bases or in new case bases through the CMM. The Query Definition Module (QDM) allows users to interactively define simple design queries, combine compound design queries from existing situations, or extract design queries from existing models. The QDM allows designers to define queries from a set of geometric, topologic, numeric, and semantic entities and relationships (Bettig, et al. 2000). Additional tools provided include a networking tool (Summers and Shah 2001) and an interactive definition and extraction tool. As a query is defined, it is saved in a query library.

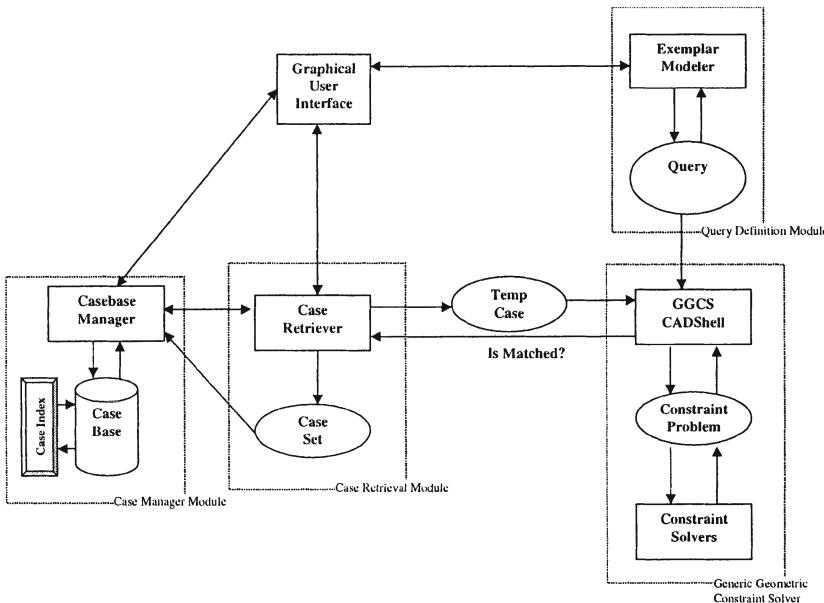
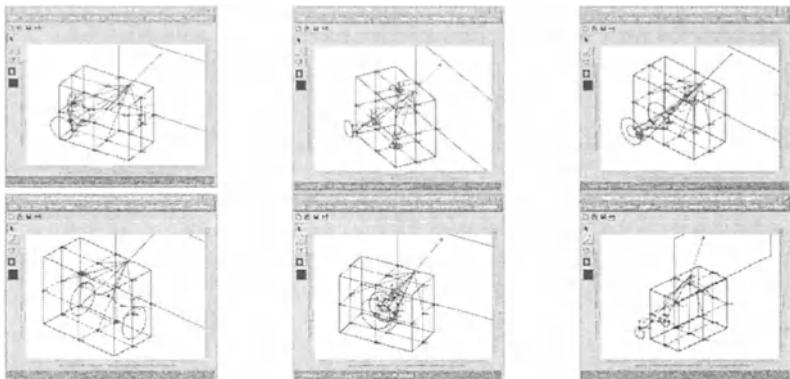


Figure 10. Case-based design system architecture

#### 4.1 FEATURE BASED RETRIEVAL SESSION

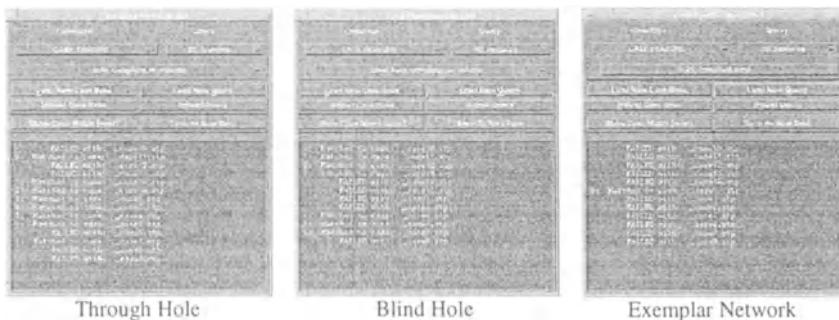
The Exemplar Based Design Retrieval (ExDRe) system provides users the opportunity to define a query, select a query from a library, and select a case

base upon which to query. A sample of the cases in a case base are illustrated in Figure 11. These cases are similar in that they are blocks with through holes, bosses, or blind holes. In allowing the designer to specify the query against these cases, several possible scenarios may develop.



*Figure 11.* Cases from “Feature Case Base”

Some simple queries may include searching for through holes, blind holes, or bosses. More complex queries may include searching for holes in bosses or assemblies of holes and bosses. The results of different queries are illustrated in Figure 12. The cases which satisfy the different queries are identified as Matched. ExDRe then allows the user to cycle through the cases to visually inspect as desired. An added functionality to the system allows the user to save the matched cases in a new case base or an existing case base. In this manner, a simple indexing scheme may be developed where the cases are organized by exemplar matches.



*Figure 12.* Results of different queries applied to “Feature Case Base”

#### 4.2 HEAT TRANSFER RETRIEVAL SESSION

Consider the instance of evaluating an existing design for heat loss. The design may have a boss on that might provide additional heat transfer cooling. In order to solve this problem, some basic tasks are required. First, the boss must be found on a given model. Once found, parameters describing the boss must be extracted. These parameters are then used to determine other values through tables and equations. Finally, the height of the boss may be adjusted to simulate an infinite rod for cooling.

Equation 1 is used to determine the average temperature between the ambient and the solid temperatures. Equation 2 is used to calculate the heat loss. Equation 3 is used to determine the temperature difference. Equation 4 is used to determine the minimum height of the boss for the top surface to be maintained at the ambient air temperature. Equation 5 is used to determine the perimeter of the circular fin.

$$T = \frac{(T_b + T_a)}{2} \quad \text{Equation 1} \quad q_f = \theta_b \times \sqrt{h \times P \times k \times A} \quad \text{Equation 2}$$

$$\theta_b = (T_b - T_a) \quad \text{Equation 3} \quad L = 2.65 \times \sqrt{\frac{k \times A}{h \times P}} \quad \text{Equation 4}$$

$$P = 2 \times r \times \pi \quad \text{Equation 5} \quad A = r^2 \times \pi \quad \text{Equation 6}$$

where:

$q_f$  is the heat loss;

$\Theta_b$  is the temperature difference between ambient and solid;

$h$  is the convective heat transfer coefficient;

$P$  is the perimeter of the fin;

$k$  is the thermal conductivity constant;

$A$  is the fin cross-sectional area;

$r$  is the radius of the fin;

$T_b$  is the solid body temperature;

$T_a$  is the ambient air temperature; and

$T$  is the average temperature between the solid and the ambient air.

These equations are combined with a table to determine the heat loss of a cylindrical fin. Table 1 illustrates an abridged table of thermophysical properties of selected metallic solids. The table is reformatted to the table format for the Geometric Constraint Solving System. The materials selected are: aluminum, copper, iron, stainless steel (AISI 304), and aluminum alloy (Alloy 2024-T6). This table takes as the row key a material type. The table's column key is the average temperature. The reference value that is sought from the table is the conductive heat transfer coefficient ( $k$ ).

TABLE 1. Thermophysical properties of metals (Incorpora and DeWitt 1990).

| 1 (aluminum) | 2 (copper) | 3 (iron)   | 4 (stainless steel) | 5 (aluminum alloy) |
|--------------|------------|------------|---------------------|--------------------|
| <b>100</b>   | <b>200</b> | <b>300</b> | <b>400</b>          | <b>600</b>         |
| 302          | 237        | 237        | 240                 | 231                |
| 482          | 413        | 401        | 393                 | 379                |
| 134          | 94         | 80.2       | 69.5                | 54.7               |
| 9.2          | 12.6       | 14.9       | 16.6                | 19.8               |
| 65           | 163        | 177        | 186                 | 186                |

In developing a complex exemplar network to find a boss, calculate the heat loss from the boss, and determine if the height of the boss is acceptable for considering it to be “infinite” with the top surface of the boss being the same temperature as the ambient air, a set of exemplars are written. These exemplars include the boss exemplar, the heat loss equation exemplar Figure 13a, the area calculating exemplar, Figure 13b, and a height verification exemplar, Figure 13c.

```

Alpha Match:
Parameter T_infinite;
Parameter T_solid;
Parameter h;
Parameter material;
Parameter radius;
ID (T_infinite);
ID (T_solid);
ID (h);
ID (material);
ID (radius);

Alpha Extract:
Parameter (heat_loss);
Parameter (k);
Parameter (P);
Parameter (area);
Parameter (radius);
Parameter (L);
Parameter (T_average);
Parameter (T_difference);
Equation (eq1);
Equation (eq2);
Equation (eq3);
Equation (eq4);
Equation (eq5);
Equation (eq6);
Table (material, T_average, k);
ID (heat_loss);

```

(a) Heat loss equations exemplar

```

Alpha Match:
Surface "S1";
Alpha Extract:
Parameter "area";
Area (S1, area);

```

(b) Area exemplar

```

Alpha Match:
Parameter "height";
Parameter "min";
Alpha Extract:
Equation "bool1", (height < min);

```

(c) Height comparison exemplar

Figure 13. Exemplars for Heat Loss Network

These exemplars are converted to exemplar nodes by selecting the inputs, outputs, and biputs of the entities and constraints from each of the exemplars. In developing an exemplar network, the individual exemplar nodes must be first loaded into the active model. This is done through the Networking Toolbar. Figure 14 illustrates the exemplar network for cylindrical cooling fins.

The retrieval system provides users the opportunity to define a query, select a query from a library, and select a case base upon which to query. Carrying through the example provided for determining whether a boss on a part might be acceptable as a circular fin to dissipate heat such that the boss tip surface temperature is in equilibrium with the ambient air, a case of models is provided.

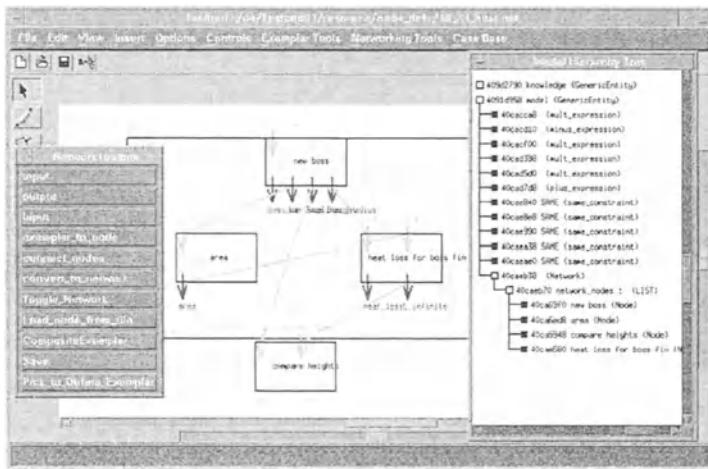


Figure 14. Heat loss network

Applying the query developed in Figure 14 against the case base illustrated in Figure 11, the results of the case retrieval are found, Figure 15. These results indicate that for the temperatures, the material, the radius, the height, and the convective heat transfer coefficients specified in the models, some cases were found to be acceptable as circular fins while other were not.

## 5. Conclusion

Design exemplars offer the expected functionalities to be used in CBD. The geometric exemplar operates at the same level as the design case data, providing a close match between design tools and design data. The

exemplar relies upon the algebraic, semantic, topologic, and geometric entities and constraints, facilitating multiple levels of domain independent design queries.



Figure 15. Results of heat loss query applied to “Sample Case Base”

Exemplars can be used to model and organize the cases by extracting the key elements of an approved design and storing this graph of elements as the case. The approach may assist in (1) selecting the properties that are suitable for organizing the case base and express them through exemplars, (2) evaluating each exemplar against the cases, and (3) organizing the cases. Exemplars capture precisely the geometry and topology in the design that satisfy the property expressed in the alpha condition. By doing so, they provide the approach with a good degree of granularity: both the case and the feature are retrieved.

Using exemplars can significantly improve the development of case-based reasoning technology. First, exemplars can be used to design the case-base itself. The approaches described in Section 2 rely on algorithms that capture a variety of properties used to clarify and annotate cases. Similarly, exemplars can be used to capture properties and annotate cases. But using exemplars allows a significant improvement in case storage. Properties captured by exemplars can be linked to the constraints in the design model that matched the exemplar and the ones satisfying the extract part of the exemplar. Therefore, this set of constraints of the design model captures exactly the property. Instead of annotating the whole model, the extracted constraints can be annotated instead, offering a better granularity of the entire approach. In addition, exemplars capture explicit as well as

implicit design properties. A case validates an exemplar when both its *match* and *extract* components of the alpha condition are true. Indexing cases with exemplars extends standard approaches by allowing different but semantically equivalent representations of properties to be indexed in a single representation format.

A second contribution that exemplars can offer to case-based reasoning is the ability to combine pre-indexed as well as computed on-the-fly properties. All approaches of Section 2 rely on a retrieval phase limited to properties pre-computed and used to index the case base. Using exemplars allow a retrieval phase significantly more expressive by combining exemplars that may have been used to organize the base and thus are indexed with other computed on-the-fly. Cases may be searched against a set of rules used to describe a manufacturing process. All cases that would be available to be manufactured by this process may be retrieved. Thus, past cases may be evaluated against new technology, not available at time of initial design.

As of today, well over one hundred properties have been expressed with exemplars. The Generic Constraint Solving System and the Exemplar Modeler (Summers and Shah 2001) designed and implemented in the Design Automated Laboratory (DAL) at Arizona State University provide users with the ability to define and store new exemplars and evaluate exemplar queries against geometry in STEP format.

Key contributions of this research in design automation systems include:

- *Designer-Centric Approach* – Queries are expressed intuitively by the design engineer (end-user).
- *Data-Centric Approach* – Queries are defined in terms of the actual case data.
- *Real Query Language* – Moves beyond annotations and Boolean expressions.
- *Dynamic Indexing* – Intelligent support to redefine the index as design needs evolve.
- *Multilevel Reasoning* – Provides support for reasoning about explicit and implicit characteristics of the design cases.

While exemplar technology is still in the infancy and much additional investigation is still required, it is believed that an exemplar based approach for case based retrieval and reasoning may provide flexibility in definition of queries, adaptation, and indexing. Exemplars do not provide solutions to all case based design problems, but offer interesting directions of investigation to solve these issues. A future extension to this system is in developing an adaptation module. As mentioned earlier, exemplars have the capability to transform design models through standard algorithms. This

next stage of development requires developing complex and compound exemplars for specific design domains and situations.

### Acknowledgements

This work was supported by NSF through grant DMI-9812977.

### References

- Aha, D and Breslow, L: 2001, Conversational case-based reasoning, *Applied Intelligence* **14**: 9-32.
- Amen, R and Vomacka, P: 2001, Case-based reasoning as a tool for materials selection, *Materials and Design* **22**: 353-8.
- Bayarri, V and Leifer, L: 1995, Understanding design information handling behavior using time and information measure, *Design Engineering Technical Conferences*, American Society of Mechanical Engineers, **2**: 555-562.
- Bettig, B, Shah, J and Summers, J: 2000, Domain independent characterization of parametric and geometric problems in embodiment design, *ASME*, Baltimore, MD, DETC-2000/DAC-14259.
- Bilgic, T and Fox, M: 1996, Constraint-based retrieval of engineering design cases: context as constraints, in JS Gero and F Sudweeks (eds), *Artificial Intelligence in Design'96*, Kluwer, Dordrecht, pp. 269-288.
- Chang, K, Raman, P, Carlisle, W and Cross, J: 1996, A self-improving helpdesk service system using case-based reasoning techniques, *Computers in Industry*, **30**, 113-25.
- Elinson, A, Herrmann, J, Minis, I, Nau, D and Singh, G: 1997, Toward hybrid variant/generative process planning, *Proceedings of DETC'97: Design for Manufacturing*, Sacramento, CA, DETC97/DFM-4333.
- Empolis Knowledge Management: 2001, *Products: CBR Works*, <http://www.km.empolis.com/start.htm>, September 26, 2001.
- Gero, JS: 1994, Computational Models of Creative Design Processes, *AI in Creativity*, ed. T. Dartnall, Kluwer, Dordrecht, pp. 269-281.
- Goel, A and Chandrasekaran, B: 1989, Functional representation of designs and redesign problem solving, *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence (IJCAI-89)*, Detroit, Michigan, pp. 1388-94.
- Hua, K, Schmitt, G and Faltings, B: 1992, Adaptation of spatial design cases, in JS Gero and F Sudweeks (eds), *Artificial Intelligence in Design'92*, Kluwer, Dordrecht, pp. 559-75.
- Inference Corporation: 2001, *Inference Corporation - eGain*, <http://knowledge.egain.com/>, September 26, 2001.
- Krampe, D and Lusti, M: 1997, Case-based reasoning for information system design, *Case Based Reasoning, Research and Development, Second International Conference, ICCBR-97*, Providence, RI, Springer-Verlag, Providence, RI, pp. 63-73.
- Lee, D, Kang, J, Ryu, K and Lee, K: 1997, Applying memory-based learning to indexing of reference ships for case-based conceptual ship design, *Case Based Reasoning, Research and Development, Second International Conference, ICCBR-97*, Providence, RI, Springer-Verlag, Providence, RI, pp. 74-83.
- Maher, M, Balachandran, M and Zhang, D: 1995, *Case-Based Reasoning in Design*, Lawrence Erlbaum, Mahwah, NJ.

- Munoz-Avila, H, McFarlane, D, Aha, D, Ballas, J, Breslow, L and Nau, D: 1999, Using guidelines to constraint interactive case-based htn planning, *Proceedings of the Third International Conference on Case-Based Reasoning*, Springer-Verlag, Seeon, pp. 288-302.
- Murdock, J, Goel, A, Donahoo, M and Navathe, S., 2001, A framework for method-specific knowledge compilation from databases, *Journal of Intelligent Information Systems* **17**: 5-21.
- Perera, S and Watson, I: 1996, NIRMANI: an integrated case-based system for strategic design and estimating, in ID Watson (ed.), *Progress in Case-Based Reasoning, First United Kingdom Workshop*, Springer-Verlag, Salford, pp. 185-200.
- Schaaf, J: 1996, Fish and shrink. a next step towards efficient case retrieval in large scaled case bases, in I Smith and B Faltings (eds), *Advances in Case-Based Reasoning, Third European Workshop, EWCBR-96*, Springer-Verlag, Lausanne, pp. 362-376.
- Summers, J and Shah, J: 2001, The design exemplar: a new data structure for design automation, *Journal of Mechanical Engineering Design*, ASME, (submitted).
- Sycara, K and Navinchandra, D: 1989, Integrated case-based reasoning and qualitative reasoning in engineering design, in JS Gero (ed.), *Artificial Intelligence in Design'89*, Springer-Verlag, New York, NY, pp. 231-50.
- Wang, J and Howard, H: 1989, A design-dependent knowledge for structural engineering design, in JS Gero and F Sudweeks (eds), *Artificial Intelligence in Design'91*, Butterworth-Heinemann, Oxford, pp. 151-70.
- Zdrahal, Z and Motta, E: 1996, case-based problem solving methods for parametric design tasks, in I Smith and B Faltings (eds), *Advances in Case-Based Reasoning, Third European Workshop, EWCBR-96*, Springer-Verlag, Lausanne, pp. 473-486.

## **SYSTEMIC SUPPORT**

---

*Connectivity as a key to supporting design: A state-action approach to planning*  
Claudia Eckert and P John Clarkson

*Automated (re-)design of software agents*  
Frances MT Brazier and Niek JE Wijngaards

*Automated toolset selection for feature manufacturing*  
Farsad Badjgholi, Burkhard Kittl and Markus Stumptner

## CONNECTIVITY AS A KEY TO SUPPORTING DESIGN

*A state-action approach to planning*

CLAUDIA ECKERT AND P JOHN CLARKSON  
*University of Cambridge*  
*United Kingdom*

**Abstract.** In the past, support for design has largely concentrated on specific design problems. The great challenge for many designers and design managers working on complex problems is to understand the connections between different aspects of design represented by different types of design models. Designs comprise connected components, and processes comprise linked tasks, but tasks are also connected to components. Each design project is embedded in the context of other projects. This paper discusses an on-going research project into design process improvement, which is centred on the use of process models and product models, designed to show important aspects of connectivity. These models are inevitably provisional due to the uncertain nature of design. The process model describes tasks and their linking parameters; the product model incorporates a probabilistic measure of the risk of change to one component spreading to others in terms of likelihood and impact. Planning and scheduling activities before the project begins can support the allocation of resources and indicate critical points in the process. Change prediction can aid change planning and decision making about possible design moves. Explicit models of connectivity enable designers to question each other's assumptions and aid communication in design teams. However, the challenge lies in building sufficiently accurate models cost effectively.

### 1. Introduction

Over the past decades support systems for all aspects of design have made enormous progress. Intelligent applications have been built for many specific design tasks, starting with domain specific expert system modules in the 1980s and moving on to generic systems in the 1990s. Hot research issues of the past, such as mismatch detection in geometric components are

now standard tools within commercial CAD systems. In addition to supporting different individual aspects of design, support systems are moving towards integrating different support modules on the level of translating between different data formats. However, there is little support for designers and design managers to handle those tasks in the most efficient way during the development of a complex design.

This paper discusses on-going research into an integrated framework for design support, that models the connectivity between tasks to support information flow, task selection and task planning. This is an ambitious endeavour and the paper will characterise design as a complex activity; it will not only report on the progress we have made so far, but also on some the challenges we are still facing. This paper does not describe one particular intelligent system in detail but discusses the breadth of support that can be provided by taking a holistic approach. Some of the support modules include significant artificial intelligence components, others are centred on mathematical models or provide visualisation for complex data sets.

In the 1980s and early 1990s the focus of AI in design was on the automation of design tasks with the aim of taking over one part of design. Generative systems have looked at generating new designs using genetic algorithms (Rosenman 1996), shape grammars (Stiny 1980) and functional synthesis (Chakrabarti and Bligh 1994, 1996). Some of these systems provided interesting theoretical insights into the nature of design, others were commercially successful (up to a point) for routine design tasks. Only very few achieved industrial relevance for novel designs (for example, Shea and Cagan 1999). Most research systems targeted conceptual design and have taught us much about the nature of conceptual design, but have been of little practical use. Industry is rarely short of good ideas, but struggles to select those that they can complete on time and on budget with the given resources. Therefore this line of research is now shifting its emphasis from fully automatic systems to interactive systems (Eckert et al. 1999; Chase 2002) or components for larger interactive systems. A nice example of successful integrated intelligent modules are to be found in the commercial knitwear CAD systems produced by knitting machine manufacturers such as Stoll and Shima Seiki, that use expert systems with thousands of rules to translate a symbolic representation of knitted structure into knitting machine instructions.

### 1.1 SIGNIFICANT CHARACTERISTICS OF DESIGN

Over the past 30 years design research has learned much about the characteristics of design processes (see for example, Cross 1989; Pahl and

Beitz 1996; and Ehrlenspiel 1995, for a discussion). The extensive body of empirical design research supports some conclusions about the nature of the design of complex products that we have taken as the starting point for the research discussed in this paper, namely that:

- Design is dynamic: During the design process new requirements, new understanding and unexpected problems occur. Therefore design is inherently uncertain. For most design problems it is impossible to plan the entire design process completely and accurately. However, by attempting to do so we can learn much about the problem in hand.
- Design is chaotic: Small changes to a design or seemingly unimportant actions in the design process can sometimes have a huge effect. It is inevitable that large complex design processes will be impacted by unforeseeable contingent events or unforeseen consequences of design decisions, where other equally likely events would cause parts of the process to take a significantly different form. Hence models that represent processes in terms of individual tasks need to avoid closed-world assumptions and treat design as inherently non-deterministic.

These factors have a huge impact on the behaviour of design processes, because all design activities and all components of a design are connected to each other directly or indirectly (as will be discussed in section 0). There is a need to consider this connectedness, and the dynamic nature of design, when providing support for the planning and scheduling of design activities, and supporting decision making and design communication, as well as when considering options for manufacturing.

## 1.2 OUR EMPIRICAL STUDIES

The ‘Signposting’ research has developed out of an attempt to capture the complexity of rotor blade design for helicopters, a design task so highly inter-linked that it defeats conventional stage based design models (Clarkson and Hamilton 2000). Following on from this study, the entire helicopter was studied to understand the connectivity between different parts of the craft to predict the impact of change in a customisation process (Clarkson et al. 2001). In addition, the engineers involved in helicopter design were interviewed to understand how they handled change processes (Eckert et al. 2000) and questioned about their communication processes (Eckert et al. 2001). Most recently a study at a British automotive company has investigated communication and planning behaviour. Current case studies are looking at planning and change in a number of UK companies from the aerospace, automotive and consultancy sectors.

### 1.3 OVERVIEW OVER THE PAPER

Section 2 discusses the issue of connectivity within a design and its process, as well as that of a design within the context of other designs. Different models of design processes that address the connectivity between elements are introduced in Section 3. Connectivity as a key to design support is discussed in Section 4, looking in particular at planning, change and communication. The success of support systems for process planning, change management and communication within design teams depends vitally on building good models, and some of the challenges of model building are outlined in Section 5.

## 2. Connectivity in Design

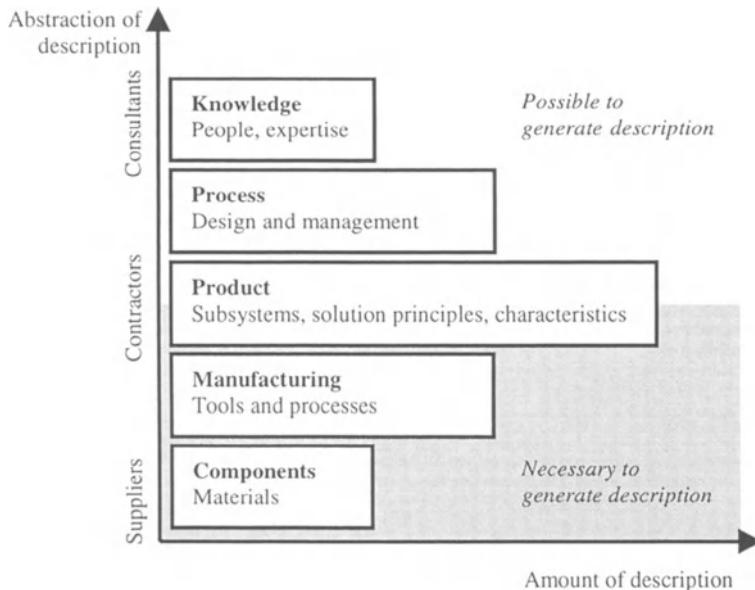
Many design methodologies and design support tools embody the assumption that designs can be decomposed into individual parts or subsystems, which in turn can be designed or purchased in isolation. Similarly, it is assumed that design processes can be decomposed into individual tasks and activities. For example, an aeroplane can be decomposed into its wings, fuselage, cockpit, engines etc. In turn these can be broken down hierarchically into their parts, for example, the shape of the wing and the wing struts. Such decomposition is possible under certain conditions, and specific tasks can be undertaken pretty much in isolation and can be supported by independent intelligent systems. For example, Airbus has successfully used ICAD to speed up the routine task of wing strut design by at least a factor of 10 (Thomas 2000).

However, this is only part of the story. As Simon (1996) points out, design processes for complex products are inherently complex, because design systems are only almost decomposable, but never fully decomposable. Like other highly complex systems such as animals, economies and ecosystems, they function by having relatively weak and simple interactions between many complex subsystems. Many design processes impose strict hierarchical decompositions with simple and clear defined interfaces between subsystems, and a predetermined or arbitrary order of design decisions, in order to make the process of designing a complex system simple enough to be tractable. But for some products, for example helicopters, this approach is infeasible or leads to significantly suboptimal designs. While a hierarchical breakdown of a system can provide both an accurate understanding of its inherent hierarchical structure and a useful structure to conceptualise design complex problems, making them tractable, it may hide or over-simplify many critical dependencies between individual components of relatively-weakly-interacting subsystems. Using a

particular hierarchical breakdown of the structure of a complex product biases the design that is generated. In our research we are concerned with ways to handle these more complex and subtle interactions in engineering products, to avoid or alleviate the problems they cause.

## 2.1 LAYERS OF DESIGN

A design can be described and analysed from different perspectives. For example, Figure 1 illustrates these perspectives on design descriptions as layers with differing degrees of abstraction in the description and a different amount of information to describe each layer (Earl et al. 2001). Component descriptions, such as bills of material, are the most specific; they are also the most parsimonious descriptions because part numbers suffice. Manufacturing uses fairly standard processes, which can be described in standard ways. The richest descriptions are on the layer of the product itself.



*Figure 1. Layers of design description by abstraction and amount of description*

Design processes have many features in common and include widely-applicable standard procedures; but they can rarely be described completely, because of the uncertainty in design. Knowledge is very general and widely applicable but notoriously hard to capture. Decisions about the product or its design and manufacturing process have an impact on all of the layers and

thus their impacts on one layer cannot really be assessed in isolation. The importance of each description changes over the course of the project, as the descriptions become more specific.

Every design project is embedded in the context of all other activities in the company that are going on at the same time, or have happened in the past, or are planned or anticipated for the future, Figure 2. Products are often developed in ranges of closely related products; but products can also have a bearing on other products in the company by sharing components, subsystems and solution principles as well as design and manufacturing processes.

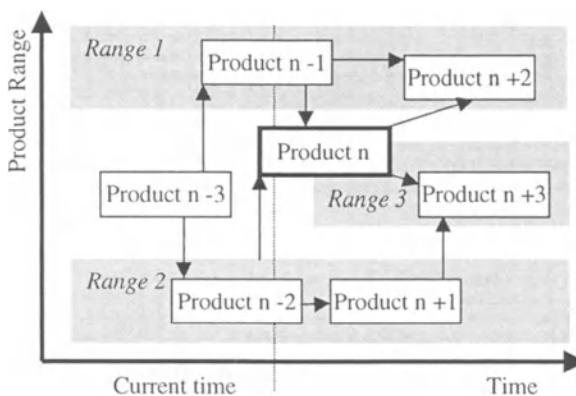


Figure 2. A product in context

Different products can be developed through similar or even isomorphic design processes. But they can also share design tasks. For example, two products can be based on the same market research process. If the requirements of both designs are sufficiently similar then a substantial saving can be achieved through shared activities. However, if they are too different then adapting the results of a design task for one product to the needs of another product might require more time than doing the task again from scratch. The merit of sharing tasks between projects also depends on the human resources. Companies make trade-offs between projects to make optimal use of their existing expertise, but also to develop expertise for future projects.

The top three layers of the model in Figure 1 are similar to Eppinger's division into product, process and organisation (Eppinger 2001). However, it is also important to integrate manufacturing and components into a complete picture of design. Manufacturing resources and constraints drive

many design processes, and fundamental design decisions are often made to accommodate manufacturing. Design decisions can be made to circumnavigate limited internal manufacturing resources. Increasingly, research in manufacturing engineering focuses on the integration of manufacturing concerns into design processes to achieve more effective design for manufacturability, as well as enabling early and efficient manufacturing planning.

On the component layer, different designs can share the same components. This says very little about the design or its process, but is in fact the layer on which company often records and compares designs in their bills of material.

## 2.2 SIMILARITY

How efficiently two products and their processes can be integrated depends on how much they have in common. Products can be similar to each other on all the layers, but the nature of this similarity is very different. Table 1 summarizes a much longer discussion in Earl et al. (2001) on what can be similar on different layers of design.

TABLE 1 Similarity on different layers of design

| LAYER         | SIMILARITY                                      |
|---------------|-------------------------------------------------|
| Components    | Identity                                        |
| Manufacturing | Similar product → similar manufacturing process |
| Product       | Systems, solution principles                    |
| Process       | Procedures, tasks, strategies                   |
| Knowledge     | Expertise                                       |

Similarity is not black or white; it is ordinarily a matter of degree depending on the distance between two things that are compared. A new design will share many aspects on all layers with past and future designs. Only by understanding the exact nature of the similarity relations between engineered products can the distance between products be assessed and new tasks identified that will generate truly new aspects of a design or adapt existing ones.

## 2.3 UNDERSTANDING CONNECTIVITY

Families of products form a network through what they share across all layers of design. Therefore decisions about an element of a product should

not be made without considering its impact on other layers. For example, a component should not be altered without considering:

- the impact on other components in the same design;
- the effect on the design process, for example, new tasks that need new or different resources;
- the impact on other products in the company, for example, the stock handling of a component that is no longer in a product.

In addition, many connections between products are not static, but change in a dynamic process. However, some connections remain static and can be assessed at the beginning of a design process. It is important to identify those connections that could cause difficulties later, since even if predicted connections or problems turn out to be absent, an explicit representation allows designers to question assumptions and understand a hidden bias in design process planning.

### **3. Models of Design and Design Connectivity**

Models are used for a variety of purposes in design – product models based on components, functions, or behaviours; process models; manufacturing models. But most models concentrate on one layer of the design phenomenon and do not address the connections between layers. An entire project, process or organisation can only be understood by the use of multiple models. This is reflected in the Common KADS knowledge systems development methodology (Wielinga et al. 1992; Tansley and Hayball 1993; Schreiber et al. 1993, 1999). Common KADS uses hierarchical models of organisational processes (prior to introducing the new system), the co-operation between participants, and the expertise required to solve problems as inputs to the design process of a system. In most engineering applications the requirements come from outside the organisation and the properties of the organisation are part of the constraints on the design process. However, the effort involved in building such multiple models is enormous.

Resource limitations are one of the sources of unpredictability in design processes. Limited resources cause bottlenecks and uncertainties that can not be anticipated at the beginning of the design process. This situation is familiar in manufacturing, where analytic information theory measures of complexity are based on operational uncertainties (Frizelle and Woodcock 1995). Manufacturing processes are easier to model and plan than design processes, because they hold less uncertainty. The lengths of tasks, the resources required and the risk associated with them are better known. Manufacturing research has drawn on research on complexity in many different fields. Complexities in designing and its associated decision

processes are described by related entropy maximising measures by Tribus (1969), March (1976) and Suh (1990; 1999). These indicate how tasks could be spread across available design processes or resources. The complexity of the distribution of tasks proposed by these methods reflects the complexity of the underlying structure of processes. Structural complexity is described by Johnson (1995) who concentrates on hierarchies of elements and relations.

Design research has concentrated for a long time on the development of high level generic models. While these can provide useful insights to practitioners, they are often of little practical use. Otto (2001) discusses a modelling approach for the connections between product modules in an attempt to construct integrated modular product families. Design structure matrices (Steward 1981; Eppinger et al. 1994) and Signposting models (Clarkson and Hamilton 2000) are approaches to modelling specific design processes.

### 3.1 GENERIC MODELS OF DESIGN

Design processes have been modelled in many different ways. Stage models of the design process, in terms of idea generation, conceptual design, embodiment and detailed design (for example, Pahl and Beitz 1996; Dym 1994; and Cross 1989 for an overview) provide useful guidance for the development of milestones, but do not address specific design activities or product properties. Activity models such as those proposed by Shigley and Mischke (1989) and Blessing (1994) give an indication of necessary task sequences, but don't describe a specific process. Stage-based design process models have been developed further using generic building blocks of the design and manufacturing processes, for example, by Bichlmaier (2000). The blocks represent typical activities and the hand-over between blocks is represented by deliverable documents. These models emphasize the links between design, manufacturing and assembly, but remain at a high level with a small number of blocks. Development methodologies based on generic stage models have been very influential in software engineering, for instance SSADM (see for instance Goodland with Slater 1995).

### 3.2 DESIGN STRUCTURE MATRICES

Design Structure Matrices (DSMs) (Steward 1981; Browning in press) provide a simple yet powerful representation of dependencies (either between tasks or between design parameters). A DSM is a square matrix with identical row and column labels. An off-diagonal mark signifies the dependency of one element on another. These matrices show properties of

processes and can be reordered to define processes with minimum iteration. Static DSMs show dependencies in elements that do not change with time, for example, components of a product or groups in an organisation; they can be analysed with clustering algorithms. In time-based DSMs, for example, modelling design tasks, the ordering of rows and columns indicates a flow of time, Figure 3. As Browning (in press) shows in his review article, practitioners and academics alike have applied DSMs successfully to problems in numerous sectors.

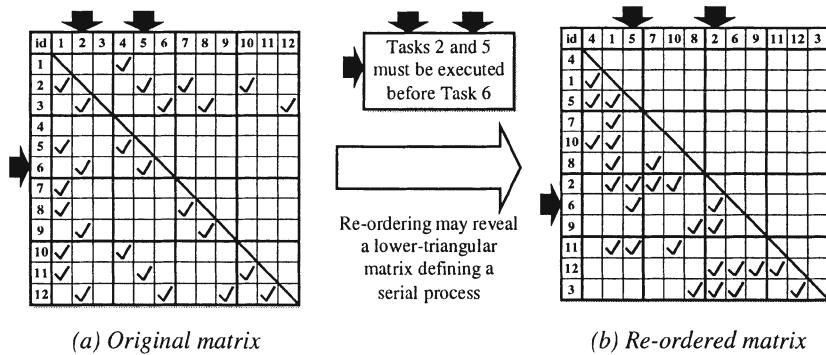


Figure 3. A time-based DSM

### 3.3 SIGNPOSTING

The Signposting approach models design processes through tasks and their connecting parameters – the inputs they require and the outputs they generate. Each parameter has a qualifier associated with it, which indicates the state of this parameter in terms of how reliable its current value is. Currently this is the subjective confidence that the designer has in the refinement of the value, but we are considering the use of other kinds of uncertainty metrics (Stacey and Eckert in press). A task can be executed when its input parameters are available with sufficiently high confidence values. Figure 4 illustrates a simple confidence mapping that defines the use of a task. New tasks can be added to the set of tasks at any time without interfering with the rest of the model. Iteration can be modelled to a limited extent by re-running a task with different confidence values. Initially the Signposting approach was used to guide designers to the next task, by showing the designers those tasks for which they had input data with sufficient confidence (Clarkson and Hamilton 2000). We are now extending

the Signposting approach to dynamic process planning. Section 0 explains the algorithms for generating plans in detail.

#### 4. Design Process Improvement

Computer support for complex design problems has different requirements from supporting individual design tasks. An individual task usually has one objective and one type of user. As long as the users are involved in key decisions, they are happy when laborious tasks are taken over by support systems (see Eckert et al. 1999).

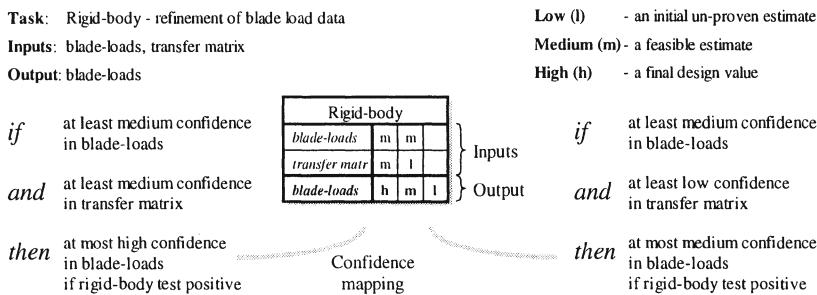


Figure 4. A typical signposting confidence mapping

Artificial intelligence can be included in the closed world of a specific task and is gratefully received if it brings real improvements. Integrated project support involves finding the right process as well as the right balance between potentially conflicting objectives. Designers and design managers must make trade-offs and are held accountable for them. Supporting and understanding the connectivity between different design tasks is a step in this direction. It allows users to put processes together and make design decisions based on an assessment of impact across a system. Once a project plan is generated it becomes explicit and can be assessed and discussed amongst participants. More subtle properties of task dependencies can also be teased out.

Due to the dynamic nature of design, task models and plans will inevitably change as new tasks are added and others become no longer relevant. However, designers can understand changes much more easily when they have an explicit plan that is modified, rather than no plan at all. It enables designers to question the assumptions of others and put their own activities in context. A trail of changes is also one way of capturing a design history.

This section reports on ongoing research which extends the Signposting approach (see section 0) to address the entire life cycle of a design project. It begins with a discussion of support for planning and scheduling before the design project begins, and looks at support for strategic planning, decision making and communication during the project, before addressing support for post-mortem analysis.

The system presented is under construction. The underlying research addresses different areas of design process improvement, which are linked by similar models and techniques, as illustrated in Figure 5.

Design processes are supported through planning and scheduling modules. As design is dynamic, plans will only be accurate close to milestones at which the project moves from one phase to the next, and schedules are likely to have to be changed.

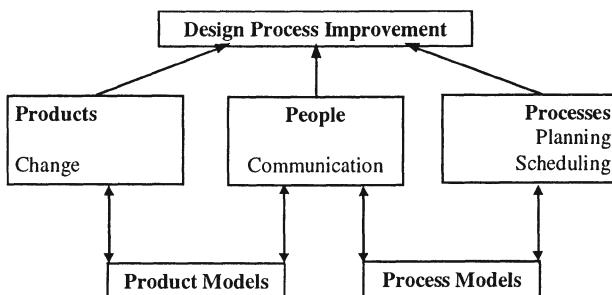


Figure 5. Design process improvement

People are supported by displaying information flows, so that they know where information comes from and whom they need to inform. Process and product models support decision making, because they allow the designer to assess the implications of a design decision. The design of products is supported by a tool that allows the assessment of the risk of a change to one component propagating to other parts of a system. The same functionality also supports planning at the beginning of a process, because it enables the designer to gauge modifications required to an initial 'starting' design.

The ideas are independent of the modelling approach we have chosen. In fact they are very similar to recent research based on DSMs (Eppinger et al. 2001) that also addresses support for people through organisational DSMs, support for processes by giving task orderings, and support for products by providing product connectivity matrices. However, DSMs are purely analytical tools while Signposting can be interactive.

#### 4.1 A PRIORI DESIGN PLANNING

Before a design project begins, design managers need to know as much as possible about the product that will be designed, and the process required to design it in order to allocate resources and avoid unnecessary iteration. Their understanding of the product depends on similar products they have developed in the past. If they want to adapt a past process they need to understand the similarity between the products, and also the specific circumstances that had influenced the processes associated with the previous products. For example, when the design process for a similar design was greatly inhibited by staff allocation problems, a process for a similar product that avoids these problems might be more effective. On the other hand, if novices work on the current project the same tasks could take longer. Understanding this dependency between product, process and context is vital.

##### *4.1.1 Planning design processes*

Process planning requires the identification of inherent process properties, for example, fixed sequences, bottlenecks and parallel clusters (Clarkson et al. 2000), Figure 6. In its simplest form, planning may be reduced to the identification of task precedences (Eppinger et al. 1994). However, that is an over-simplistic view that ignores iteration and the consequent risk of delays leading to an increase in project cost.

Melo and Clarkson (2001) propose the identification of the lowest risk policy for process planning where, for a range of different conditions, the best route to design completion is found. However, such an approach is difficult to undertake with incomplete task models or different levels of task detail. That being said, partial models are better than no models.

##### *4.1.2 Critical path analysis*

The next step from planning is the scheduling of design tasks. Scheduling in design is extremely difficult because of the unpredictability of the data. However, knowing which tasks lie on the critical path, projected from the current design knowledge, is crucial. In some ways design managers do this today when they are planning around lead-times for components. However, this often ignores the internal design tasks on the critical paths. Knowing which tasks are on the critical path, managers can either assign more resources to these tasks or look for alternative solutions to the problem. As in other scheduling applications, currently resources often are directed at improving tasks that might not lie on the critical path and therefore have very little effect on the overall design time.

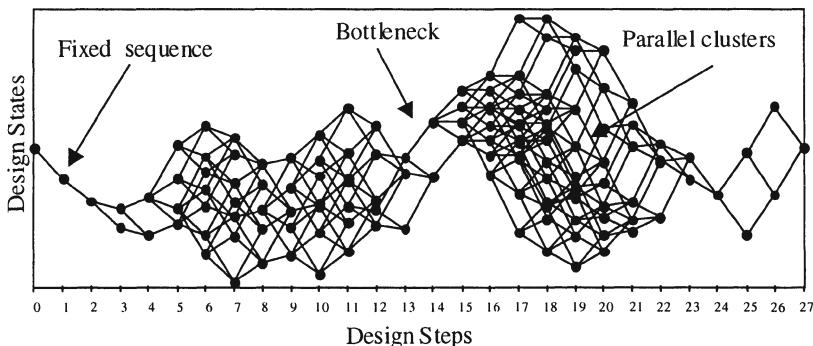


Figure 6. Routes through a design process

#### 4.1.3 Change to existing products

Drawing on a detailed empirical study of change processes in the customisation of complex products (Eckert et al. 2001), we have developed a probabilistic method to predict the risk of change propagating to other parts of a product (Clarkson et al. 2001). Practitioners can more easily assess the likelihood and impact of a change to one parameter on the aspects of the design it is directly connected to than they can assess indirect influences. Our method computes estimates of indirect influences from the users' estimates of direct influences. For example, changes to the engine of a helicopter may lead to changes to the engine auxiliaries. In addition, there may be indirect changes, for example, a change to engine resulting in a change to the fuselage via a change to the engine auxiliaries.

The process begins by decomposing the starting design into a set of components, Figure 7. In the initial case study a helicopter was broken down into 19 subsystems to provide a high-level model for initial estimates of change impact before a new design project begins. A more detailed model with about 400 components was developed to capture instances of past change propagation, to inform designers during the design process about the potential for changes propagating to other components.

In the model the components are identified as the rows and columns of a prediction matrix. The direct likelihoods and impacts were elicited from practitioners in interviews, while the indirect values were calculated by algorithms discussed in Clarkson et al. (2001). Recently similar results have been achieved using a Monte Carlo method, which reduces the computational complexity from  $O(n!)$  to  $O(n)$  and allows a further exploration of the search space defining the propagation behaviour of the components. For example, under some circumstances it is reasonable to assume that changes will decrease as they propagate, but they could equally

amplify. In addition, some components cannot be changed for technical reasons, for example, the rotor in a helicopter would not be altered because the entire design has been tuned to its rotational frequency.

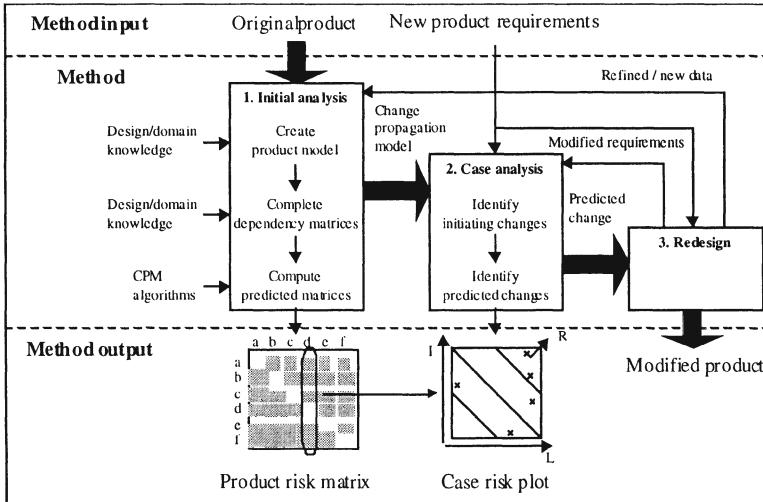


Figure 7. The change prediction method

In a complex product with many thousands of parts, for example, a helicopter or an aeroplane, these change prediction methods cannot express many details and do not capture the chaotic nature of design where small changes can have a huge impact. However, in case studies the method managed to predict all the changes that had occurred. The method makes potential connections explicit, which can then be discussed by interested parties. Design decisions can be made to avoid certain propagation routes.

#### 4.2 ON-GOING STRATEGIC SUPPORT

As design progresses and new requirements arise and problems occur, design plans need to be modified. Many unpredicted problems pull resources off other parts of a project – they are given to the most desperate problem. If companies were able to re-plan their design processes, this fire-fighting approach would become less frequent and at hoc by being planned and integrated into other design activities.

The planning tool discussed in Section 4.1 can also be used during the design when strategic planning is required, because more complete data about the risk of task failure and the resources required is available. Fire-fighting breeds fire-fighting, because new problems arise as experts are

pulled off tasks. A design plan gives design managers access to information about trade-offs between different parts of the design plan. They can pull representatives in from different tasks and work out strategic solutions.

#### 4.3 SUPPORTING ON GOING DESIGN ACTIVITIES

The applications discussed in the previous two sections were aimed at project managers who need to plan design activities. The same underlying model of design connectivity can also support the designers as they undertake their tasks.

##### *4.3.1 Communication*

Communication in design teams is difficult for numerous reasons even when people work in the same place at the same time sharing the same objectives and expertise (for example, Bucciarelli 1994). Problems become more difficult in other communication scenarios (see Eckert and Stacey 2001 for a discussion of communication scenarios) as designers have less in common and have less physical contact. The reasons for ineffective communication can be manifold, from specialists in different areas having different concepts and mental models and different external representations, through to wilful withholding of information (Bucciarelli 1994; Henderson 1999). Eckert (2001) identifies about a dozen different reasons why design teams in the knitwear industry fail to overcome problems arising from the inherent difficulty of describing knitted structures, and the incomplete, inconsistent and inaccurate representations that knitwear designers typically produce.

Not ignoring the complex social factors involved in design communication (Henderson 1999; Minneman 1991), many problems arise because designers do not understand the information flow through an organisation (Eckert et al. 2001). Designers and design managers comment over and over that problems arise as tasks are omitted when people are not informed about them, and that designers do not know where information they need has originated, and who, later on, depends on their results. In big projects designers often feel disoriented because they don't know how their activities fit into the bigger picture. This research makes the assumption that designers would communicate with each other more efficiently, if only they knew whom to communicate with.

Design plans that display the connectivity between tasks and product components can help to ensure that the right information reaches the right person at the right time in the right format. Looking back in the process designers can trace where a parameter has been generated and who has made use of it in the past. They can query the inputs to decisions about it

and discuss the scope for change. These communication trails can be generated from a design plan.

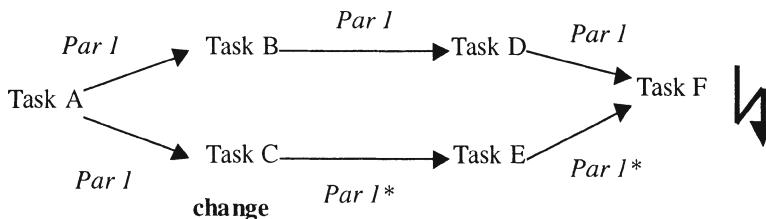
The task-based model can be used as a carrier for annotated design information to capture some of the richness of face-to-face communication in asynchronous situations (Stacey and Eckert *in press*). Designers can automatically be encouraged to inform each other of changes to a parameter, when they make changes (Gomes 2001).

Design plans also support communication by making hidden assumptions about a process explicit. At present companies may use a number of different plans, from many different angles, with strong implicit biases that are very hard to for outsiders understand. A common representation allows everybody to see a process in the same way. Problems often arise because people are not aware of changes to plans. Again a common plan that is updated as the process progresses can overcome these problems. Users of DSMs often comment that the greatest advantage of DSMs is making hidden processes explicit and forcing people to think about the connections. For them the merit might lie more in model building than in the use of the models.

#### *4.3.2 Decision making*

In making design decisions, designers are often unaware of the consequences of their decisions on other parts of the designs. For example, as Figure 8 illustrates, one designer (task C) changes a parameter ( $\text{par 1} \rightarrow \text{par 1}^*$ ) potentially unaware that another designer (task B) has based their decision on the original value. This can lead to problems later on (task F), which are then difficult to understand. Knowing what the dependencies are, designers might be able to make different decisions which would have less impact on the overall design.

The original signposting systems supported design decision making by showing designers those tasks for which they has sufficient knowledge and which would advance their knowledge of the design.



*Figure 8. An example of decision dependency*

#### *4.3.3 Integrated design environment*

The modules discussed in the previous sections support the designer or design manager throughout the entire design process and complement other support tools currently used. The system can also be set up to drive other support tools or interfaces to them to provide designers with one integrated design tool. One system architecture can be used to bring all the parts together (see Stacey et al. 2000), however the integration of domain specific modules has to be tailored to specific applications. Jarrett (2001) discusses an integrated design guidance system for turbine design, based on signposting, that is used by Rolls-Royce.

### 4.4 A POSTERIORI SUPPORT: FURTHER WORK

To date this research has not addressed the use of design plans after particular projects have been concluded. The following applications might be possible:

- Knowledge management: capture of knowledge associated with tasks as they are carried out to support the subsequent use of a similar task.
- Capturing design rationale for a sequence of changes: as plans are updated the rationale for each change could be captured. Putting these rationales together would give a complete account of the design process life cycle.
- Project post-mortem: an analysis of the most successful and least successful tasks and strategies in a design process.

## 5. Building task based models

All the research discussed in Section 4 is based on the assumption that it is possible to build models of design processes and products.

### 5.1 THE CHALLENGE OF BUILDING MODELS

There is no unique model of a product or design process, but many different models could be built. The structure of the model influences the analysis of the design problem and ultimately the results produced with any support system. Design research has put great effort into building very high-level generic models of design processes, but these provide little help for building applied models. It is also very simple to build high-level models of products and processes, for example, an aeroplane = {wings, cockpit, fuselage, undercarriage, etc} and therefore the aeroplane design process = {design wings, design cockpit, design fuselage, design undercarriage, etc}. The tasks on the lowest level are equally simple to describe, because they are almost generic, for example, low level activities = {look at brief, chase up missing

information, hypothesise solution, run analysis etc.). The challenge lies in modelling a product or a process on a meaningful level between the two extremes, which reflects the specific nature of the product and the problems associated with it. The model must show the connectivity between components and tasks, which is vital to the success of a project. Model building has to be a two-stage process:

- (1) Eliciting the data from the designers, design managers etc.
- (2) Making sure that the models are accurate, complete and consistent within a well understood scope.

While it might not be possible to built complete models of an uncertain process like design, users must at least be able to trust the model to describe aspects of a process as reliably as possible. Building meaningful models is inevitably time consuming and it is therefore critical to provide a favourable benefit to effort ratio. In products with a stable product architecture, task models can be reused and detailed models can be built up from generic building blocks. Austin (1999) has taken this approach successfully in the construction industry, where the entire design process can be described using about 300 generic building blocks, which can then be processed in a DSM.

### 5.2 THE PRESENT APPROACH

At present our group takes the approach of researchers eliciting models from practitioners through highly directed interviews. We analyse the data received from designers and design managers, identify gaps and question our informants about them.

### 5.3 A VISION OF THE FUTURE

For an integrated design support system, as described in section 4, it is essential that the user can build models independently of any researchers or trained knowledge engineers. Designers and design managers must build their own models and the system must help them in finding inconsistencies and missing tasks in the models. Future research will be directed to finding generic building blocks for engineering design, extending the work done at the Technical University of Munich (Bichlmaier 2000) in identifying typical task patterns; for example, at the end of most activities there will be a “write report” task. Most designs are based on existing designs. Design by modification is ubiquitous in engineering, and design reuse is a major topic of research in engineering (for instance, Shivaloganathan and Shahin 1998). Therefore it should be possible to use previous process plans as well as previous designs as starting points for the generation of new plans. Change

prediction (section 0) gives a handle on the tasks required to modify a product, and assists the building of a new process model. However, the question remains as to how to identify the right starting point from which to design each aspect of a new product – this raises the question of how to monify and integrate chunks of pre-existing product and process models.

## 6. Conclusions

Design product and process models make implicit assumptions about a process explicit and provide designers and managers the chance to reflect over their products and processes. This can aid them in many activities: planning design processes, assessing the impact of changes, and supporting the communication within an organisation. The key to understanding a process and predicting its behaviour lies in understanding the connectivity between different aspects of a process. Our future work is directed both to supporting designers building models that will represent the vital connections between components and tasks, and to supporting them throughout the entire process using these models.

## Acknowledgements

The authors acknowledge the support of the UK Engineering and Physical Sciences Research Council. The following past and present members of the Cambridge Engineering Design Centre have made major contributions to research on Signposting: Jamie Hamilton, Tim Jarratt, Jerome Jarrett, Andres Melo, Brendan O'Donovan, Caroline Simons. Professor Chris Earl of the Open University has contributed to our research on the role of complexity in design. Dr Martin Stacey of De Montfort University has contributed to our research on the use of Signposting systems for knowledge management in design. Dr Stacey also made helpful comments on an earlier version of this paper.

## References

- Austin, S, Baldwin, A, Li, B and Waskett, P: 1999, Analytical design planning technique: a model of the detailed building design process, *Design Studies* 20(3): 279-296.
- Bichlmaier, C: 2000, *Methoden zur flexiblen Gestaltung von integrierten Entwicklungsprozessen*, Doctoral dissertation, Technische Universität München, München.
- Browning, TR: in press, applying the design structure matrix system to decomposition and integration problems: a review and new directions, *IEEE Transactions on Engineering Management*.
- Blessing, LTM: 1994, *A Process-Based Approach to Computer Supported Engineering Design*, PhD thesis, University of Twente, Enschede, Netherlands.
- Buccarelli, LL: 1994, *Designing Engineers*, MIT Press, Cambridge, MA.
- Chakrabarti, A and Bligh, TP: 1994, An approach to functional synthesis of solutions in mechanical conceptual design. Part I: Introduction and Knowledge Representation, *Research in Engineering Design* 6: 127-141.

- Chakrabarti, A and Bligh, TP: 1996, An approach to functional synthesis of solutions in mechanical conceptual design. Part II: Kind Synthesis, *Research in Engineering Design* **8**: 52-62.
- Chase, SC: 2002, A model for user interaction in grammar-based design systems, *Automation in Construction* **11**(2): 161-172.
- Clarkson, PJ, Simons, C and Eckert, C: 2001, Predicting change propagation in complex design, *Proceedings of the ASME 2001 Design Engineering Technical Conferences: 13<sup>th</sup> International Conference on Design Theory and Methodology*, American Society of Mechanical Engineers, Pittsburgh, Pennsylvania.
- Clarkson, PJ, Melo, AF and Connor, A: 2000, Signposting for design process improvement, in JS Gero (ed.) *Artificial Intelligence in Design'00*, Kluwer, Dordrecht, pp. 333-353.
- Clarkson, PJ and Hamilton, JR: 2000, Signposting: a parameter-driven task-based model of the design process, *Research in Engineering Design* **12**: 18-38.
- Cross, N: 1989, *Engineering Design Methods*, Wiley, Chichester.
- Dym, CL: 1994 *Engineering Design: A Synthesis of Views*, Cambridge University Press, Cambridge, UK.
- Eckert, CM: 2001, The communication bottleneck in knitwear design: analysis and computing solutions, *Computer Supported Cooperative Work* **10**: 29-74.
- Eckert, CM, Clarkson, PJ and Stacey, MK: 2001, Information flow in engineering companies: Problems and their causes, *Proceedings of the 13<sup>th</sup> International Conference on Engineering Design: Design Management – Process and Information Issues*, Professional Engineering Publishers, Glasgow, pp. 43-50.
- Eckert, CM and Stacey, MK: 2001, Dimensions of communication in design, *Proceedings of the 13<sup>th</sup> International Conference on Engineering Design: Design Management – Process and Information Issues*, Professional Engineering Publishers, Glasgow, pp 473-480.
- Eckert, CM, Kelly, I and Stacey, MK: 1999, Interactive generative systems for conceptual design: an empirical perspective, *AIEDAM* **13**(4): 303-320.
- Eckert, CM, Zanker, W and Clarkson, PJ: 2001, Aspects of a better understanding of changes, *Proceedings of the 13<sup>th</sup> International Conference on Engineering Design: Design Applications in Industry and Education*, Professional Engineering Publishers, Glasgow, pp. 147-154.
- Ehrlenspiel, K: 1995, *Integrierte Produktentwicklung*, Carl Hauser Verlag, Munich.
- Eppinger, SD, Whitney, DE, Smith, RP and Gebala, DA: 1994, A model-based method for organizing tasks in product development, *Research in Engineering Design* **6**: 1-13.
- Eppinger, SD and Salminen, VK: 2001, Patterns of product development interactions, *Proceedings of the 13<sup>th</sup> International Conference on Engineering Design: Design Research – Theories, Methodologies and Product Modelling*, Professional Engineering Publishers, Glasgow, pp. 283-290.
- Frizelle, G and Woodcock, E: 1995, Measuring complexity as an aid to developing operational strategy, *International Journal of Operations and Production Management* **15**(5): 26-39.
- Gomes, S, Eynard, B and Magnon, L. 2001, A multi-site design experience using a computer supported collaborative work tool, *Proceedings of the 13<sup>th</sup> International Conference on Engineering Design: Design Applications in Industry and Education*, Professional Engineering Publishers, Glasgow, pp. 379-386.
- Goodland, M with Slater, C: 1995, *SSADM Version 4: A Practical Approach*, McGraw-Hill, Maidenhead, UK.
- Henderson, K: 1999, *On Line and on Paper*, The MIT Press, Cambridge, MA.

- Jarrett, JP and Clarkson, PJ: 2001, Improving the product by accelerating the process, *Proceedings of the 13<sup>th</sup> International Conference on Engineering Design: Design Research – Theories, Methodologies and Product Modelling*, Professional Engineering Publishers, Glasgow, pp. 141-148.
- Johnson, J: 1995, The multidimensional networks of complex systems, in J Casti, R Thord and D.Batten (eds), *Networks in Action*, Springer, Berlin, pp. 49-79.
- March, L: 1976, The logic of design and the question of value, in L March (ed.), *Architecture of Form*, Cambridge University Press, Cambridge, pp. 1-40.
- Melo, AF and Clarkson, PJ: 2001, Design process planning using a state-action model, *Proceedings of the 13<sup>th</sup> International Conference on Engineering Design: Design Management – Process and Information Issues*, Professional Engineering Publishers, Glasgow, pp. 345-352.
- Minneman, SL: 1991, *The Social Construction of a Technical Reality: Empirical Studies of Group Engineering Design Practice*, PhD Thesis, Department of Mechanical Engineering, Stanford University, CA.
- Otto, K: 2001, A process for modularizing product families, *Proceedings of the 13<sup>th</sup> International Conference on Engineering Design: Design Methods for Performance and Sustainability*, Professional Engineering Publishers, Glasgow, pp. 523-530.
- Pahl, G and Beitz, W: 1996, *Engineering Design*, 2nd ed., translated by K.M. Wallace, L.T.M. Blessing and F. Bauert, Springer, London, UK.
- Rosenman, M: 1996, The generation of form using an evolutionary approach, in JS Gero and F Sudweeks (eds.) *Artificial Intelligence in Design'96*, Kluwer, Dordrecht, pp. 643-662.
- Schreiber, ATh, Wielinga, BJ and Breuker, JA: 1993, *KADS: A Principled Approach to Knowledge-Based System Development*, Academic Press, London, UK.
- Schreiber, ATh, Akkermans, H, Anjewierden, A, de Hoog, R, Shadbolt, N, van der Velde, , and Wielinga, R: 1999, *Knowledge Engineering and Management*, MIT Press, Cambridge, MA.
- Shea, K and Cagan, J: 1999, The design of novel roof trusses with shape annealing: assessing the ability of a computational method in aiding structural designers with varying design intent, *Design Studies* 20: 3-23.
- Shigley, JE and Mischke, CR: 1989, *Mechanical Engineering Design*, McGraw-Hill, New York.
- Shivaloganathan, S and Shahin, TMM (eds.): 1998, *Design Reuse*, Professional Engineering Publishing, Edmonds.
- Simon, H: 1996, *The Sciences of the Artificial*, 3<sup>rd</sup> Edition, MIT Press, Cambridge, MA.
- Stacey, MK, Clarkson, PJ and Eckert, CM: 2000, Signposting: an AI approach to supporting human decision making in design, *Proceedings of the ASME Design Engineering Technical Conferences: 20th Computers and Information in Engineering Conference*, American Society of Mechanical Engineers, Baltimore, MD.
- Stacey, MK and Eckert, CM: in press, Against ambiguity, *Computer Supported Cooperative Work*.
- Stiny, G: 1980 Introduction to shape and shape grammars, *Environment and Planning B: Planning and Design* 7: 343-351.
- Steward, DV: 1981, The design structure matrix: a method for managing the design of complex systems, *IEEE Transactions on Engineering Management* EM-28(3): 71-74.
- Suh, NP: 1990, *Principles of Design*, Oxford University Press, New York.
- Suh, NP: 1999, A theory of complexity, periodicity and the design axioms, *Research in Engineering Design* 11: 116-133.

- Tansley, DSF and Hayball, CC: 1993, *Knowledge-Based Systems Analysis and Design*, Prentice Hall, London.
- Thomas, G: 2000, Cabin and system knowledge tools, *Proceedings of Knowledge-Based Organizations 2000*, <http://www.kboworld.com/kboeurope2000.html>.
- Tribus, M: 1969, *Rational Descriptions, Decisions and Designs*, Pergamon, New York.
- Wielinga, BJ, Schreiber, ATh and Breuker, JA: 1992: KADS: A modelling approach to knowledge engineering, *Knowledge Acquisition* 4: 5-53.

## AUTOMATED (RE-)DESIGN OF SOFTWARE AGENTS

FRANCES MT BRAZIER AND NIEK JE WIJNGAARDS  
*Vrije Universiteit Amsterdam*  
*The Netherlands*

**Abstract.** Autonomous software agents are dynamic entities: they are capable of discovering a need for change - for additional knowledge and/or functionality on the basis of their analysis of specific situations. Agent factories are capable of redesigning and reactivating agents on the basis of the information provided by agents and/or knowledge available within the agent factories. As a result agents may evolve in ways their designers could never have pre-conceived. The artefacts themselves are dynamic in a way that can not be compared to any other type of design in current design practice. A number of prototype agents and agent factories have been built to evaluate the feasibility of this concept and its consequences.

### 1. Introduction

Design and re-design are often intertwined. In a changing world in which needs, desires, requirements, manufacturing processes and technology continually change, designs often need to adapt. New designs are often based on existing designs: the question as to whether a design process is a re-design process or not, is not always easily answered or relevant. Human designers identify new needs and desires, and designs evolve.

In some cases of automated design processes, however, the situation may slightly differ. Automated software design is an example of a domain in which reference models, components and rules are used to automatically design an artefact (software). In many cases the design process is deterministic and can be fully tracked. It becomes more interesting when the software designed has a will of its own. This is the case for software agents. Intelligent software agents are autonomous pieces of code: they typically act in unpredictable environments on the basis of changing knowledge. They are designed to learn from their interaction with their environment and communication with other agents. They can also be designed to discover their own limitations. They have a degree of self-

awareness and evolve.

This paper describes an approach to automated re-design of agents, based on agents' self-awareness. Agents are capable of discovering a need for additional knowledge and/or functionality. On the basis of their analysis of specific situations and their mandates (the need to act in a given situation) agents can decide to be adapted. Agent factories provide this functionality: agent factories are capable of adapting agents' code and state and re-activating them. Agent factories perform the re-design. The agents themselves decide which agent factory they trust to perform a re-design task<sup>o</sup>. In contrast to non-automated design the artefact has acquired an active role in the design process. As a result an agent may evolve in a way its designer could never have pre-conceived: the design artefact becomes a dynamic artefact.

This paper describes the principles behind adaptive artefacts of the kind described above. Section 2 addresses the phenomenon of self-awareness in agents, Section 3 sketches the re-design process and Section 4 illustrates this process for an information retrieval agent. Section 5 discusses the results and places the results in the context of current and future research.

## 2. Agents' Self-Awareness

This section focuses on the role/meaning of self-awareness of agents. Section 2.1 introduces the concept of an adaptive system. Section 2.2 describes the types of knowledge, information and functionality needed to acquire self-awareness. Section 2.3 discusses degrees of self-awareness.

### 2.1 ADAPTIVE SYSTEMS

Adaptive systems, or dynamic artefacts, are artefacts that change due to changes in their environment. Examples of dynamic artefacts include buildings that adjust lighting and temperature on the basis of occupation of rooms (Mozer 1999), elevators that second-guess the behaviour of their clientele, auto-pilots of aeroplanes, which take, and relinquish, control to the human pilots, or self-configuration of autonomous (spacecraft) systems (Williams and Nayak 1996).

Software agents are a specific type of autonomous systems that adapt as a result of interaction with their environment and/or communication with other agents. Personification is an example: information gathering agents often maintain profiles of other agents (possibly human (Wells and Wolfers 2000)), and adapt these profiles on the basis of interaction with these agents.

---

<sup>o</sup> In theory the agent factory could be part of each and every agent but in current prototypes these functions have been separated.

Another example of adaptive agent behaviour, e.g. by (Rus, Gray and Kotz 1996) in distributed information gathering, occurs when an agent is capable of abandoning a previous goal or plan, and adopting a new goal or plan which better fits the current situation of the agent. Most often the learning techniques involved determine the type and level of adaptation e.g. as described by (Reffat and Gero 2000; Grefenstette 1992).

## 2.2 SELF-AWARENESS: TYPES OF KNOWLEDGE, INFORMATION AND FUNCTIONALITY

Self-aware agents need to have the following knowledge, information, and functionality:

- Self-awareness knowledge: the agent must have knowledge that describes the agent's functionality and behaviour, as well as non-functional characteristics of the agent.
- Monitoring information: the agent must be able to monitor its functioning and behaviour in a given situation.(e.g. when it wishes to be adapted and re-activated),
- Self-assessment knowledge: the agent must have knowledge that determines the extent to which an agent may function in a given situation
- Need formulation knowledge: the agent must have knowledge with which needs for adaptation can be determined
- Integration: self-awareness knowledge, monitoring information, self-assessment knowledge, and need formulation knowledge must be integrated in the (internal) functioning of a self-aware agent.
- Communication: the agent must have knowledge of the way in which it can choose an agent factory, and interact with an agent factory (e.g. protocols and languages for either an intermediary or direct interaction with the factory).

How this knowledge, information and functionality is acquired depends on an agent's design. If, for example, an agent's architecture is based on a generic agent model (e.g. Brazier, Jonker and Treur 2000) with a specific component for internal "own process control", this component may be a perfect candidate for self-awareness knowledge and functionality.

## 2.3 DEGREE OF SELF-AWARENESS

An adaptive agent needs to be aware of its own abilities (which may also be named skills, or tasks, or behaviours, etc.). The knowledge about its abilities may be simple in form (e.g., facts), or more elaborate (e.g., a model of its own behaviour). Most important is that an adaptive agent is able to recognise, or predict, situations in which it may, or may not, function

correctly.

Self-awareness knowledge is imprecise by nature. Although the absence, or presence, of certain abilities may result in a strong belief in success or failure, only by actually performing a certain task, can an adaptive agent discover actual success.

Irrespective of the form of the knowledge, an agent has a certain *degree of self-awareness*. At the one extreme of the spectrum is total non-awareness: the agent does not know which factors determine success or failure of a task. At the other extreme of the spectrum is total self-awareness: the agent fully understands its own abilities, and is able to predict with a high degree of certainty whether or not it is able to perform a certain task.

The less precise an agent's self-awareness knowledge, the less precise its needs for adaptation can be formulated, and the more responsibility is placed on an agent factory.

### 3. Adapting Agents

An agent factory is a facility that creates, and modifies, software agents. Section 3.1 describes the principles on which the agent factory is based. Section 3.2 describes the use of building blocks in the agent factory.

#### 3.1 AGENT FACTORY

An agent factory designs and adapts agents. The agent factory is based on five underlying assumptions (Brazier and Wijngaards 2001a): (1) agents have a compositional structure, (2) re-usable parts of agents can be identified, (3) two levels of descriptions are used: conceptual and detailed, (4) properties and knowledge of properties are available, and (5) no commitments are made to specific (programming or modelling) languages and/or ontologies.

The design of an agent within the agent factory is based on configuration of building blocks: a blueprint. Building blocks include cases and partial (agent) designs (cf. generic models / design patterns). This approach is related to design patterns (e.g., Gamma, Helm, Johnson and Vlissides 1994; Peña-Mora and Vadhavkar 1996), libraries of software with specific functionality (e.g., problem-solving models (Schreiber, Akkermans, Anjewierden, de Hoog, Shadbolt, van de Velde, and Wielinga 1999), and generic task models (Chandrasekaran 1986; Brazier, Jonker, and Treur 2000)).

The configuration-based approach relates to, e.g., model-based re-configuration (Stumptner and Wotawa 1998), design using re-usable design

patterns (Peña-Mora and Vadhavkar 1996), and IBROW (Motta, Fensel, Gaspari and Benjamins, 1999), in which re-usable parts of problem-solving methods are 'configured'.

An agent factory includes a number of processes, some of which are involved with account management (which client requires which quality of service), agent packaging and handling (how to prepare an agent for transfer to an agent platform), etc. One of the processes, the design centre, is responsible for the (re-)design of agents. The design centre is based on a model for (re-)design of compositional systems (Brazier, Jonker, Treur and Wijngaards 2001), in which explicit reasoning about strategies, requirements, and artefact descriptions is modelled. The blueprint of an agent functions as both its functional specification, but also as (part of the) design rationale concerning the agent (Peña-Mora and Vadhvakar 1996).

Within the design centre, needs for adaptation are interpreted into disambiguated, apparently non-conflicting, and specific requirements on the basis of which the blueprint of an agent may be modified. Strategies play an important role in this process: not only are strategies needed to decide when to manipulate requirements, and then to manipulate blueprints, but strategic knowledge is also needed to guide the re-design process: how to (re)solve conflicting requirements, what to modify in a blueprint to satisfy given requirements, etc.

The needs for adaptation, specified by an adaptive agent, are translated into sets of qualified requirements, on the basis of which the blueprint of the agent is modified. The process of modifying an agent's blueprint entails manipulating the configuration of conceptual building blocks, the configuration of detailed building blocks, and the mapping between these configurations. The library of building blocks is limited in size, and building blocks with specific characteristics (functional and non-functional) need to be retrieved, to be (correctly) configured with other building blocks. Partially matching building blocks may need to be adapted (by other building blocks) before combined with configured building blocks, etc.

The co-ordination of the process of manipulation of sets of qualified requirements and the manipulation of blueprints, is not trivial. When to switch manipulation processes, how to synchronise parallel activities, how to allocate time and resources to each manipulation process, are of importance.

Strategic knowledge is needed to decide when to issue which design strategies to one, or both, manipulation processes. And within each manipulation process, strategic knowledge is needed to decide what to work on, and when to do what. For example, multiple interpretations may exist for a need for adaptation, each interpretation alternative may be in a different

'context'. Strategic knowledge is needed to determine which context to choose.

### 3.2 BUILDING BLOCKS

In the agent factory building blocks are either components with open slots, fully specified components, and/or a combination of both. Building blocks are defined at one of the two levels of detail: conceptual and detailed. As a result, a mapping is needed between building blocks at conceptual level and building blocks at detailed level. (Note that this mapping may be structure preserving, but that this is not necessarily ideal.) A detailed description of a building block includes the operational code. For each conceptual description, a number of detailed descriptions may be devised and vice versa. These detailed descriptions may differ in the operational language (e.g., C, C++, Java), but also in, for example, the efficiency of the operational code. The conceptual descriptions may differ in the modelling paradigm (e.g., UML, DESIRE), but also in, e.g., the detail in which an agent's functionality is modelled. In the current prototypes of the agent factory building block descriptions in DESIRE, UML, C, C++ and Java are supported.

Building blocks themselves are configurable, but cannot be combined indiscriminately. The *open slot* concept is used to regulate the ways in which components may be combined. An open slot in a component has associated properties at both levels of detail that prescribe the properties of the entity to be 'inserted'.

Specific 'glue' may be needed to aid the insertion of a building block in an open slot. Glue, which exists at both conceptual and detailed levels of design, is used to transform certain information to the correct format/ontology.

A mapping between building blocks relates a building block containing a conceptual description to a building block containing a detailed description. The mapping relates open slots of the conceptual building block to the open slots in the detailed building block.

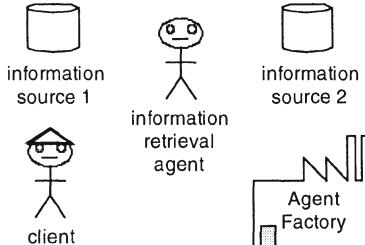
## 4. Example

The examples in this section focus on a self-aware information retrieval agent and an agent factory\*. A scenario is described in which a self-aware information retrieval agent discovers a need to adapt, finds and enters an agent factory and is adapted according to its needs. Afterwards, the agent

---

\* Issues such as security, authorisation, and payment (Wayner 1995) are not discussed in this paper.

continues its task. The elements in the scenario are shown in Figure 1. In this example, it is assumed that information is annotated by an ontology framework designed for the Semantic Web.



*Figure 1.* The actors in the scenario: a client of the information retrieval agent, the information retrieval agent, an agent factory, and a number of information sources.

Section 4.1 describes the scenario from the point of view of the adaptive information retrieval agent. The re-design process itself, is performed by an agent factory. Section 4.2 describes the re-design part of the scenario, from the point of view of the agent factory. Section 4.3 describes other examples.

#### 4.1 EXAMPLE OF AGENT SELF-AWARENESS

A client, possibly a human agent, issues a request for an airline reservation to an information retrieval agent. The information retrieval agent consults the specified information source, and discovers it cannot interpret information in that information source. Suppose, for example, that the information source is expressed in a Dutch vocabulary for seat-reservation of aeroplanes, and the agent only understands an English version of the vocabulary.

The information retrieval agent has a certain degree of self-awareness. The extent, or granularity, of its self-awareness influences the ability of the agent to formulate its needs for adaptation.

In our example the information retrieval agent possesses the following facts concerning interaction with other agents and external objects:

```

I_speak_protocol(http)
I_speak_protocol(fipa_acl_v1.0b)
I_understand_ontology_framework(xml)
I_understand_ontology_framework(rdf)
I_understand_ontology_framework(daml+oil)
I_understand_ontology(airplane_seat_reservations_English)

```

The information source, e.g., annotated in DAML+OIL (Horrocks, van

Harmelen, Patel-Schneider, Berners-Lee, Brickley, Connolly, Dean, Decker, Fensel, Hayes, Hefflin, Hendler, Lassila, McGuinness and Stein 2001) has a meta-description which states the following characteristics of its contents:

```
expressed_in_ontology_framework(daml+oil)
expressed_in_ontology(airplane_seat_reservations_Dutch)
```

The information retrieval agent is able to determine, on the basis of this information, that it is unable to understand the information source. Although it may download the specific ontology (e.g., by using Sesame by Broekstra, Kampman and Van Harmelen 2001), it doesn't know how to use the ontology, as it doesn't have any understanding of the ontology. The agent formulates a request for adaptation to the agent factory, in which the following is expressed:

```
request(possibility_for_adaptation)
required_functionality(I_understand_ontology(
airplane_seat_reservations_Dutch))
```

The agent factory acknowledges the request and the information retrieval agent contacts a directory service, discovers an agent factory close by that it trusts and is capable of performing the desired change, and migrates to this agent factory, where it is subsequently adapted. The information retrieval agent is, in this example, not 'alive' during its adaptation. When the agent factory has finished adapting the information retrieval agent, the agent is sent back to its original location, and 'awakened'.

The information retrieval agent becomes awake, receives a message from the agent factory with details about the success of its adaptation, and adds a new fact to its self-awareness knowledge:

```
I_understand_ontology(airplane_seat_reservation_Dutch)
```

The information retrieval agent continues its original task, and consults the information source in order to satisfy the request sent by its client.

#### 4.2 AGENT FACTORY EXAMPLE

Agents with a high degree of self-awareness are able to formulate specific needs for adaptation. For example, the information retrieval agent issued the following requirements for adaptation:

```
request(possibility_for_adaptation)
required_functionality(I_understand_ontology(
airplane_seat_reservations_Dutch))
```

For an agent to be adapted, yet retain its 'memory' after adaptation, an important issue needs to be resolved. This involves not only sending its blueprint to the agent factory, but also sending its *memory* (because its memory may need to be changed to 'fit' its adapted blueprint).

It is assumed that the blueprint of the information retrieval agent has become available to the agent factory. On the basis of the blueprint of the information retrieval agent, and the desired adaptation, the agent factory adapts the blueprint of the agent. This involves a number of steps.

One of the first steps is refining the needs for adaptation into more specific requirements. E.g., the need for adaptation

```
required_functionality(I_understand_ontology(
 airplane_seat_reservations_Dutch))
```

may be refined into a qualified requirement as shown below:

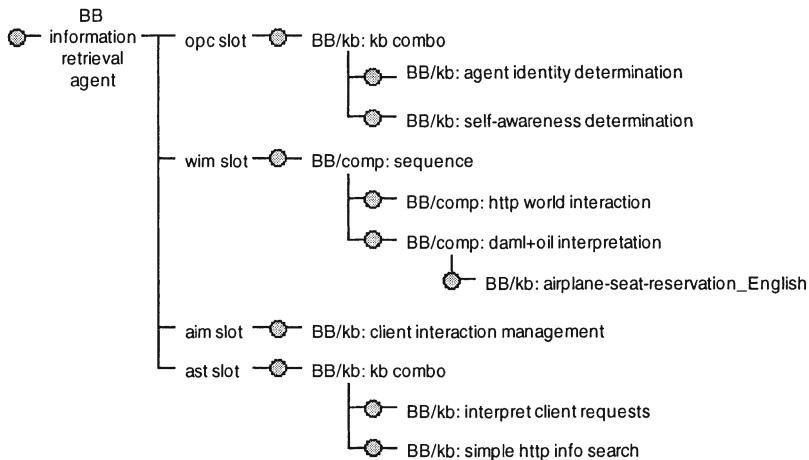
```
requirement(hard, interpret ontology(
 airplane_seat_reservations_Dutch)
 as ontology(
 airplane_seat_reservations_English))
```

That is, as both ontologies are quite similar, a mapping may be devised by which the Dutch ontology is interpreted in terms of the English ontology. This requirement may be further refined, resulting in a set of qualified requirements which can be (temporarily) committed to by the agent factory.

The blueprint of the agent is modified in accordance with this more specific set of qualified requirements. Figure 2 shows part of the conceptual building block configuration of the agent before it went to the agent factory, by only focussing on functional aspects (not including information flow within the agent). The abbreviation "BB/kb" denotes a conceptual building block containing a knowledge-base; "BB/comp" denotes a conceptual building block containing a component, which may be composed of other components.

The agent is based on a generic agent model (Brazier, Jonker and Treur 2000), in which separate processes are distinguished for control of the agent (own process control), interaction with objects in the external world (world interaction management), interaction with agents (agent interaction management), and specific tasks of an agent (agent specific task). In addition processes for maintenance of information on the world, or other agents are distinguished (not present in this specific agent).

The adaptation of an agent is a re-design process. The re-design process within the Agent Factory is based on the generic model of design (Brazier, Langen, Ruttkay and Treur 1994). This model explicitly distinguishes between reasoning about the overall co-ordination of the (re-)design process, manipulation of sets of qualified requirements, and manipulation of blueprints.

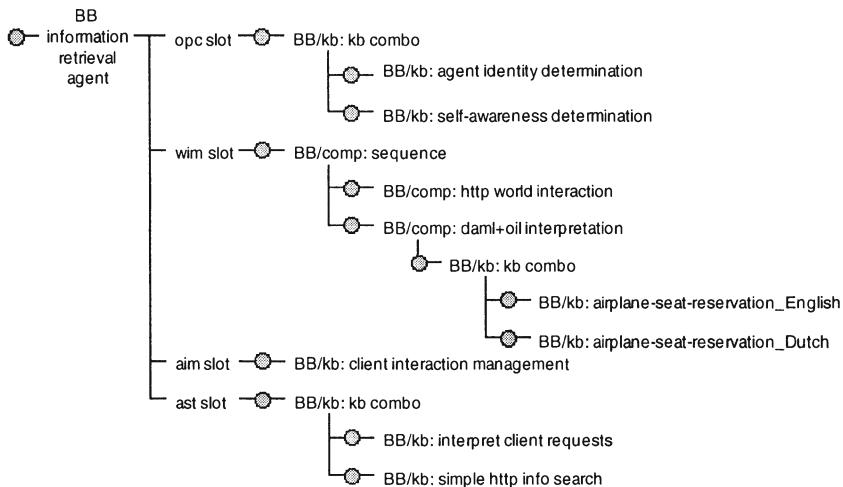


*Figure 2. Part of the initial building block configuration of the information retrieval agent.*

Although an agent is configured from building blocks, many alternatives may be considered during the re-design of an agent's blueprint. Strategic knowledge is used to guide the re-design process (Brazier, Langen and Treur 1998; Brazier, Splunter and Wijngaards 2001).

As an example of a situation in which strategic knowledge is needed, consider the following. Although the original version of the agent understands the English ontology for aeroplane seat reservations, this understanding is based on an old version of that ontology. For both the English and Dutch ontologies, new versions are available. The issue to be resolved is whether the old version of the Dutch ontology is used, or the English version is updated first, and then the new version of the Dutch ontology is used. The agent has not specified any need for adaptation concerning an update of its ontologies. As the agent is being re-designed, it is not available for comments, so it cannot aid in resolving the issue. The agent factory's strategic knowledge is responsible for resolving this issue.

The re-design process iterates between manipulating sets of qualified requirements, and manipulating blueprints (in order to satisfy requirements from a temporarily committed-to set of qualified requirements). After a number of these iterations, a blueprint has been devised which satisfies a set of qualified requirements, which is an interpretation of the needs for adaptation. Figure 3 shows the resulting conceptual building block configuration after adaptation of the agent by the agent factory.



*Figure 3.* Part of the resulting building block configuration of the information retrieval agent

The blueprint of an agent consists of both a configuration of conceptual building blocks and a configuration of detailed building blocks. Strategic knowledge is required to decide when to focus on which of the two configurations (either conceptual or detailed) of building blocks. In this example, the decision is to first complete the configuration of conceptual building blocks before modifying the configuration of detailed building blocks.

The rationale for this decision being that modifications in the configuration of detailed building blocks are more difficult to accomplish (and revoke), than modifications to the configuration of conceptual building blocks. With respect to the relation between conceptual and detailed building blocks another strategy is employed:

- New conceptual building blocks, need to be related to detailed building blocks for which a number of specific non-functional properties are the *same* (in this case: programming language and slot format) as in the detailed building blocks in the current configuration.

In our case this ensures that the agent is built entirely from detailed building blocks written in, e.g., Java and adhering to a specific format for the slots of the detailed building blocks. This strategy should most likely be replaced by the strategy that:

- New conceptual building blocks, need to be related to detailed building blocks with non-functional properties that can co-exist with the non-functional

properties (in this case: programming language and slot format) of the detailed building blocks in the current configuration.

The latter, alternative strategy would make it possible for, e.g., detailed building blocks written in Prolog to be 'wrapped' by a Java-based Prolog interpreter, and used in conjunction with other Java-based detailed building blocks.

A problem that may occur, is when none of the related detailed building blocks can be combined with the detailed building blocks in the current configuration. Again strategic knowledge is needed to decide what to do, e.g., should alternatives be sought for detailed building blocks in the current configuration, or should the configuration of conceptual building blocks be modified? In the current example the last option was chosen.

The assembly process is a subprocess of the agent factory, which assembles an executable agent on the basis of the configuration of detailed building blocks and compiles this into executable code. In the current prototype, this is a fairly straightforward process. However, during assembly incompatibilities between detailed building blocks may be discovered. Such incompatibilities are resolved by reactivating the re-design process. The binary, executable version of the agent, plus the memory of the information retrieval agent, is used to 'awaken' the agent in its original location with its new functionality.

#### 4.3 OTHER EXAMPLES

A self-aware agent may have knowledge about its own self-awareness: its degree and extent of self-awareness with respect to different situations. This enables an agent to formulate a need for adaptation concerning precisely this aspect. For example, if an agent notices its inability to successfully perform a number of tasks, it may formulate a need for adaptation to this purpose.

A self-aware agent may also employ an agent factory to (temporarily) remove part of its functionality, e.g. because it is too large (in binary size, an important aspect for mobile self-aware agents), or too computationally expensive. At a later point in time, it may request to have the functionality re-installed.

### 5. Discussion and Future Research

The agent factory is compared to a number of related approaches in Section 5.1, after which our research on agent factories is compared to our previous research. Results of this paper are discussed in Section 5.3.

### 5.1 COMPARISON

The agent factory can be compared to component-based development, agent construction kits, software reusability, case-based reasoning, and configuration design.

The agent factory's approach to combining components seems similar to the approach taken in *component-based development (CBD)* of software (Sparling, 2000). One distinction with our approach is that our approach includes annotations of components at two levels of abstraction (conceptual and operational). In CBD, interfaces are described for components (which are independent of an operational language); this correlates to the open slots in components. From our perspective CBD provides a useful means to describe operational descriptions of the building blocks used by the agent factory.

Currently a relatively large number of tools and/or frameworks exists for the (usually semi-automatic) *creation of agents*, however not automated adaptation. Examples include e.g. AgentBuilder (Reticular Systems 1999), D'agents/AgentTCL (Gray, Kotz, Cybenko, and Rus 1997), ZEUS (Nwana, Ndumu, Lyndon, and Collis 1999), NOMADS (Suri, et al. 2000), Sensible Agents (Barber, et al. 2001), and Tryllian's Agent Development Kit (Tryllian 2001). All of these approaches commit to a specific operational description of agents, and usually also commit to a specific conceptual description of their agents. The agent factory does not make such commitments, which makes the agent factory more general purpose (with all the common advantages and disadvantages).

The agent factory aims at pragmatically circumventing a number of issues related to *software reusability* (e.g., Biggerstaff and Perlis 1997). A major problem is annotating reusable pieces of software such that they can be retrieved at a later time (by other people) and reused with a minimal number of changes (Peña-Mora and Vadhwakar 1996). In the agent factory the latter is endeavoured as well. The former is currently solved in a pragmatic way: components are annotated, and, when needed, a mapping is provided to other annotations. This, however, is not a scalable solution, and, as such, one of our current foci of research. An important decision concerning standardisation is that the agent factory does not aim to adhere to one specific standard, but a number of standards.

In *case-based reasoning* approaches (e.g., Kolodner 1993; Maher and Pu 1997) libraries of cases are consulted to find a case which matches a problem, upon which the retrieved case is adapted. This approach differs from the agent factory in that cases are modified internally, instead of combined with other cases. Techniques for retrieving cases from case libraries are, of course, relevant to retrieving components from libraries.

The approaches taken by *design-as-configuration* (e.g., as described in (Stefik 1995), CommonKads (Schreiber, et al. 1999), and elevator configuration (Schreiber and Birmingham 1996)) focus on constructing a satisfactory configuration of elements on the basis of a given set of requirements (also named: constraints). In most of these approaches no explicit manipulation of requirements is present, nor is a multi-levelled description of the elements taken into account. Models and theories on configuration-based design are relevant to the agent factory, in particular to the processes involved in combining conceptual and operational descriptions.

## 5.2 COMPARISON TO PREVIOUS RESEARCH

Previous research (Brazier, Jonker and Treur 2000; Brazier, Jonker, Treur and Wijngaards 2000; 2001) focussed on automated redesign of multi-agent systems at a conceptual level. The current research extends this work in three aspects.

The first distinction with previous work is that the agent factory no longer focuses on re-designing agents on the basis of first principles at a conceptual level. The agent factory uses building blocks to construct, and adapt, agents. Building blocks are reusable parts of agents, ranging from skeletons for larger parts of agents (i.e., templates) to specific functionality (i.e., components).

The second distinction with previous work is a broadening of the scope of the re-design process. The agent factory modifies not only the conceptual description of an agent, but also its detailed description (operational code). This necessitates knowledge about the relationship between the conceptual description and detailed description in a template or component, and knowledge related to practical implications of design choices.

The third distinction with previous work is that the client of the re-design process is not available for consultation during the re-design process, as it is the subject of the re-design process as well. The redesign process is based on the agent's self-awareness knowledge.

## 5.3 DISCUSSION

Automated design of adaptive software agents is a fascinating area of research. The interplay between client, artefact and designer plays an important role: adaptive artefacts are both client and subject of (re-)design. Artefacts are designed to be adaptive: they need to recognise needs for change. The agent factory adapts the agent according to their needs.

An agent needs self-awareness knowledge to determine needs for adaptation. The 'degree of' self-awareness of an agent determines the

'degree of' interpretative knowledge needed by an agent factory: the more imprecise an agent's needs for adaptation are, the more responsibility an agent factory has for interpretation of needs for adaptation.

This paper describes the relation between self-awareness knowledge and the knowledge available to an agent factory. Automated adaptation of agents (a re-design process) is not straightforward. Agents are designed at both a conceptual level and a detailed level: a configuration of conceptual building blocks and a configuration of detailed building blocks are needed.

Self-aware agents themselves decide which agent factory is to perform a re-design task. Such self-modifying agents (Brazier and Wijngaards 2001b) may evolve in ways that their designers could never have pre-conceived. This may be a testbed for creativity as both agents and agent factories have reflective capabilities, a pre-requisite for creativity.

The approach taken in the agent factory is similar, to some extent, to approaches such as IBROW (Motta, Fensel, Gaspari and Benjamins 1999). In IBROW semi-automatic configuration is supported of intelligent problem solvers. Their building blocks are 'reusable components', which are not statically configured, but dynamically 'linked' together by modelling each building block as a CORBA object. The CORBA-object provides a wrapper for the actual implementation of a reusable component. A Unified Problem-solving Method development Language UPML (Fensel, et al. 2002) has been proposed for the conceptual modelling of their building blocks. The agent factory differs in a number of aspects, which include: multiple conceptual and detailed languages, no pre-defined wrappers for detailed building blocks, agents consist of one (multi-threaded) process, and the process of reconfiguration is an automated (re-)design process.

Current research within the IIDS group focuses on the design and implementation of an Agentscape: an agent operating system together with a set of services. The Agent Factory is one of the services. A number of prototype agent factories have been designed, implemented and tested, providing insight in the functionality required from an agent operating system, and the functionality to be provided to mobile agents aware of their options for adaptation.

## Acknowledgements

The authors wish to thank the graduate students Hidde Boonstra, David Mocabach, Oscar Scholten and Sander van Splunter for their explorative work on the application of an agent factory for an information retrieving agent. This work was supported by NLnet Foundation, <http://www.nlnet.nl/>.

## References

- Barber, KS, McKay, RM, MacMahon, MT, Martin, CE, Lam, DN, Goel, A, Han, DC and Kim, J: 2001, Sensible agents: an implemented multi-agent system and testbed, in *Proceedings of the Fifth International Conference on Autonomous Agents (Agents-2001)*, ACM Press, New York, pp. 92-99.
- Biggerstaff, TJ and Perlis, AJ (eds): 1997, *Software Reusability. Volume 1, Concepts and models*, ACM Press, New York.
- Brazier, FMT, Jonker, CM and Treur, J: 2000, Compositional design and reuse of a generic agent model, *Applied Artificial Intelligence Journal* **14**: 491-538.
- Brazier, FMT, Jonker, CM, Treur, J and Wijngaards, NJE: 2000, Deliberate evolution in multi-agent systems, in J Gero (ed.), *Artificial Intelligence in Design'00*, Kluwer, Dordrecht, pp 633-650.
- Brazier, FMT, Jonker, CM, Treur, J and Wijngaards, NJE: 2001, Compositional design of a generic design agent, *Design Studies* **22**: 439-471.
- Brazier, FMT, Langen, PHG van and Treur, J: 1998, Strategic knowledge in compositional design models, in JS Gero and F Sudweeks (eds), *Artificial Intelligence in Design'98*, Kluwer, Dordrecht, pp. 129-147.
- Brazier, FMT, Langen, PHG van, Ruttakay, Zs and Treur, J: 1994, On formal specification of design tasks, in JS Gero and F Sudweeks (eds), *Artificial Intelligence in Design'94*, Kluwer, Dordrecht, pp. 535-552.
- Brazier, FMT, Splinter, S van and Wijngaards, NJE: 2001, Strategies for integrating multiple viewpoints and levels of detail, in J. S. Gero and K. Hori (eds), *Proceedings of the International Workshop on Strategic Knowledge and Concept Formation III*, Sydney: Key Centre of Design Computing and Cognition, University of Sydney. pp. 103-128.
- Brazier, FMT and Wijngaards, NJE: 2001a, Automated servicing of agents, *Journal of AISB*, special issue on Agent Technology **1**(1): 5-20.
- Brazier, FMT and Wijngaards, NJE: 2001b, Designing self-modifying agents, in JS Gero and ML Maher (eds), *Proceedings of the Fifth International Roundtable Conference Computational and Cognitive Models of Creative Design*, Sydney: Key Centre of Design Computing and Cognition, University of Sydney, pp. 93-112.
- Broekstra, J, Kampman, A and van Harmelen, F: 2001, Sesame: an architecture for storing and querying rdf data and schema information, in D Fensel, J Hendler, H Lieberman and W Wahlster (eds), *Semantics for the WWW*, MIT Press, Cambridge, MA, pp. 272-309.
- Chandrasekaran, B: 1986, Generic tasks in knowledge-based reasoning: high level building blocks for expert system design, *IEEE Expert* **1**(3): 23-30.
- Fensel, D, Motta, E, Benjamins, VR, Crubézy, M, Decker, S, Gaspari, M, Groenboom, R, Gross, W, van Harmelen, F, Musen, M, Plaza, E, Schreiber, ATh, Studer, R and Wielinga, BJ: 2002, The unified problem-solving method development language UPML, *Knowledge and Information Systems* (to appear).
- Gamma, E, Helm, R, Johnson, R and Vlissides, J: 1994, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison Wesley Longman, Reading, MA.
- Gray, R, Kotz, D, Cybenko, GE and Rus, D: 1997, Agent Tcl, in W Cockayne and M Zyda (eds), *Mobile Agents: Explanations and Examples*, Manning Publishing, Greenwich, CT pp. 58-95.

- Grefenstette, J: 1992, The evolution of strategies for multiagent environments, *Adaptive Behavior* 1(1): 65-90.
- Horrocks, I, van Harmelen, F, Patel-Schneider, P, Berners-Lee, T, Brickley, D, Connoly, D, Dean, M, Decker, S, Fensel, D, Hayes, P, Heflin, J, Hendler, J, Lassila, O, McGuinness, D and Stein, LA: 2001, DAML+OIL, <http://www.daml.org/2001/03/daml+oil-index.html>.
- Kolodner, JL: 1993, *Case-Based Reasoning*, Morgan Kaufmann, San Mateo, CA.
- Maher, ML. and Pu, P (eds): 1997, *Issues and Applications of Case-Based Reasoning to Design*, Lawrence Erlbaum, Hillsdale, NJ.
- Motta, E, Fensel, D, Gaspari, M and Benjamins, VR: 1999, Specifications of knowledge component reuse, *Proceedings of the 11th International Conference on Software Engineering and Knowledge Engineering (SEKE-99)*, Knowledge Systems Institute, Skokie, IL, pp. 17-19.
- Mozer, MC: 1999, An intelligent environment must be adaptive, *IEEE Intelligent Systems and their Applications* 14(2): 11-13.
- Nwana, H, Ndumu, D, Lyndon, L and Collis, J: 1999, ZEUS: A toolkit and approach for building distributed multi-agent system, *Proceedings of the Third International Conference on Autonomous Agents (Autonomous Agents'99)*, ACM Press, New York, pp. 360-361.
- Peña-Mora, F and Vadhavkar, S: 1996, Design rationale and design patterns in reusable software design, in JS Gero and F Sudweeks (eds), *Artificial Intelligence in Design'96*, Kluwer, Dordrecht, pp. 251-268.
- Reffat, RM and Gero, JS: 2000, Computational situated learning in design, in JS Gero (ed.), *Artificial Intelligence in Design '00*, Kluwer, pp. 589-610.
- Reticular Systems Inc: 1999, AgentBuilder: An integrated toolkit for constructing intelligent software agents, *White Paper*, <http://www.agentbuilder.com>, February 1999.
- Rus, D, Gray, R and Kotz, D: 1996, Autonomous and adaptive agents that gather information, in *AAAI'96 International Workshop on Intelligent Adaptive Agents*, AAAI Technical Report WS-96-04, pp. 107-116.
- Schreiber, G, Akkermans, H, Anjewierden, A, de Hoog, R, Shadbolt, N, Van de Velde, W and Wielinga, B: 1999, *Knowledge Engineering and Management, the CommonKADS Methodology*, MIT Press, Cambridge, MA.
- Schreiber, ATH and Birmingham, WP (eds): 1996, Special issue on Sisyphus-VT. *International Journal of Human-Computer Studies (IJHCS)* 44: 275-280.
- Sparling, M: 2000, Lessons learned through six years of component-based development, *Communications of the ACM* 43(10): 47-53.
- Stefik, M: 1995, *Introduction to Knowledge Systems*, Morgan Kaufmann Publishers, San Francisco, CA.
- Stumptner, M and Wotawa, F: 1998, Model-based reconfiguration, in JS Gero and F Sudweeks (eds), *Artificial Intelligence in Design'98*, Kluwer, Dordrecht, pp. 45-64.
- Suri, N, Bradshaw, JM, Breedy, MR, Groth, PT, Hill, GA, Jeffers, R, Mitrovich, TS, Pouliot, BR and Smith, DS: 2000, NOMADS: toward a strong and safe mobile agent system, in *Proceedings of the Fourth International Conference on Autonomous Agents*, New York, ACM Press, pp. 163-164.
- Tryllian: 2001, Agent development kit, *Technical White Paper*, Version 1.0, June 2001, [http://www.tryllian.nl/sub\\_downl/Technical%20white%20paper%20ADK%20v1.0.pdf](http://www.tryllian.nl/sub_downl/Technical%20white%20paper%20ADK%20v1.0.pdf)

- Wells, N and Wolfers, J: 2000, Finance with a personalized touch, *Communications of the ACM*, Special Issue on Personalization 43(8): 31-34.
- Williams, BC. and Nayak, PP: 1996, A model-based approach to reactive self-configuring systems, in *Proceedings of the AAAI'96 & IAAI'96*, AAAI Press / MIT Press, Cambridge, MA, Vol 2, pp. 971-978.

## AUTOMATED TOOLSET SELECTION FOR FEATURE MANUFACTURING

FARSAD BADJHOLI AND BURKHARD KITTL

*Technische Universität Wien  
Austria*

AND

MARKUS STUMPTNER  
*University of South Australia  
Australia*

**Abstract.** Optimal tool selection is a crucial task in the efficient employment of numerically controlled machines that is increasing in complexity and difficulty. We describe a constraint-based approach for the intelligent design of toolsets for work piece manufacturing on NC machines. Initial part designs can be produced and imported from a commercial CAD system. The toolset designer incorporates tool data from manufacturer catalogs, enhances the features with manufacturing information, and selects an optimal toolset that minimizes cost, refit cost, or number of tools as desired. The core of the implementation is an optimizing constraint algorithm that can be formally expressed in terms of generative constraints, a successful configuration formalism.

### 1. Introduction

The economical operation of numerically controlled machines is strongly influenced by the organizational framework. However, in spite of extensive progress in CAD/CAM coupling, the problem of technological support for the programmer has not been brought to a satisfactory solution to date. The experience of planning staff and methods engineers still plays a crucial role in selecting the machining steps for a given work piece geometry as well as the tools required for these steps and the suitable cutting parameters. At the same time the accelerating development of cutting materials and new tool geometries increases the complexity of this task. Providing an intelligent system that incorporates such knowledge potentially shortens the planning process and improves the result, by observing boundary conditions such as

surface quality and machining costs. Furthermore, the planning process becomes predictable since it no longer is solely dependent on individual experience and expertise (or lack of it) of the specific worker involved in the job; instead it provides company-wide guidelines. As a result, the systematic collection, structuring, and representation of machining knowledge would be a significant factor influencing the competitiveness of the production process.

When considering the development of a system that would provide practical tool selection support for the NC programmer, it became evident that such a selection could not occur simply based on the properties of the individual work steps, but that the tool requirements for the work piece as a whole would have to be taken into consideration, including such global aspects as the fixation of the workpiece. Furthermore, the optimization criteria could be based on a number of different factors. For example, in batch production, minimizing the main machining time was the crucial factor whereas in the production of individual pieces, the minimization of the number of different tools was the central point. These requirements were the basis for the development of a prototype tool selection system.

A further prerequisite for the development of the TOSS (TOol Selection System) prototype was a imitation of an executed selection process by a human expert. Figure 1 shows a selection process. It was also a integration with commercial CAD systems desired. First, such systems general provide the infrastructure for the actual part designs that are to be machined, second, such systems typically provide a feature-based representation of work piece geometry. Since the geometrical features present both a set of high level concepts that permit efficient data exchange between the CAD system and the tool selection system, , the ultimate criteria that determine the outcome of the tools selection process, and the ultimate criteria that determine the outcome of the tool selection process, the capability of the TOSS system solved two issues with one stroke. The system used for actual integration with the TOSS prototype was CATIA Version 5, which incorporates a dedicated NC module, thereby providing the desired high level interface, and also provides flexible interfaces to other data sources that were of relevance to the tools selection process, e.g., tool catalogs.

## 2. Forms of Tool-Selection Related Knowledge

The knowledge that needs to be represented about workpieces and features naturally falls into three separate categories. Geometric knowledge specifically describes shapes and is the information produced as output from the CAD system. Technological knowledge is production-related, and finally, tool specific knowledge is derived from manufacturer catalogs.

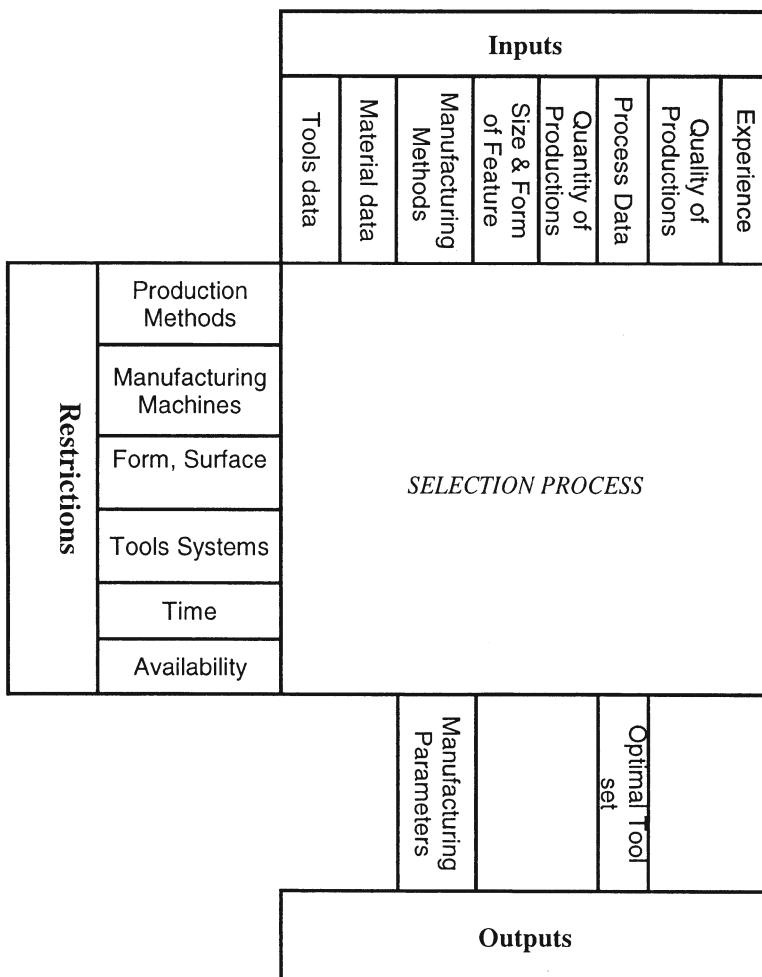


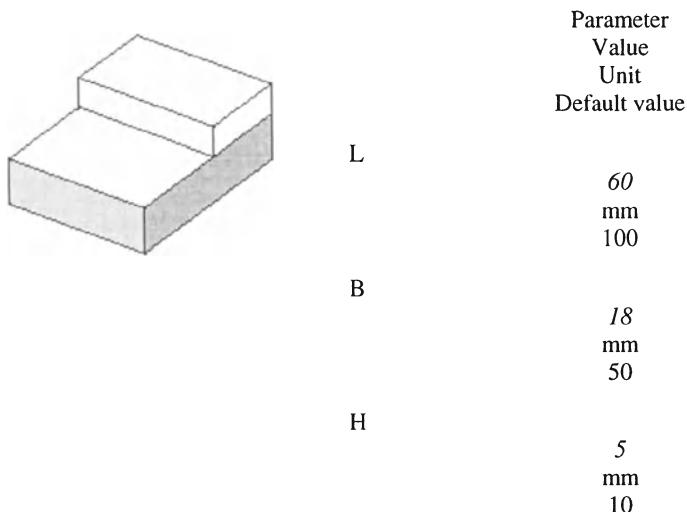
Figure 1. A tool selection process

## 2.1 GEOMETRY-RELATED KNOWLEDGE

Geometrical information represents the starting point of the selection process and is handed to the TOSS system in the form of a set of individual features, i.e., description of particular subshapes of the whole piece. Contrary to conventional 3D-Object representations in CAD systems, feature

descriptions in TOSS are syntactically extended to allow (in addition to the description of the feature shape) a semantics for the description of the technical meaning. They also permit the definition of more complex relationships between geometry and semantics (Ovtcharova 1997). Apart from geometrical parameters, they can be assigned explicit surfaces, manufacturing procedures, and other semantic information. For example, the upward gradient of the thread can be defined as an attribute of a tapped hole.

Since construction and production planning (or production preparation) focus on different tasks, in the following we differentiate between geometry features (which represent the object shape and the functions which can be fulfilled) and manufacturing features (which describe handling regulations for individual form elements). Figure 2 shows a geometrical feature, described by a list of parameters, each of which has a value, a measuring unit, and a default value specified.



*Figure 2. A step as a geometrical feature geometrical representation and stored information*

## 2.2 TECHNOLOGICAL KNOWLEDGE

The technological knowledge is modeled in the form of manufacturing features, which are assigned to the geometry features taken over by the CAD-System. Manufacturing features contain information about the manufacturing process to be applied, and thus indirectly specify the types of tool which can be used, and the crucial tool attributes for the geometry of the

work piece being processed. Apart from the geometrical description, technological information is also included in the manufacturing feature (for example the value "simple depth"). Contrary to the geometry data, which are taken over from the CAD system, these technological values (for example "Dr" or "Da" in the example below) are dependent on other choices (possibly mutually dependent) and calculated at the moment of assignment. This makes it possible for the system to optimize the manufacturing process including the possibility of switching between different types of manufacturing features that may represent the same geometrical feature. Figure 3 shows a manufacturing feature. In addition to the geometrical properties, this shows where the different parameters are defined (the purely geometrical parameters are listed as inherited from the geometrical model), and includes two technological parameters which refer to the vertical and horizontal shifts in succeeding passes of the milling tool that would be used to manufacture the step. In these parameters, the value field contains only a link to the constraint used for computation of the value (at the opportune time).

For the machining of any particular form item, it is in general possible to use multiple tool types. For example it is possible to manufacture the geometry feature "Hole" with a drill or with an end miller, Figure 4. This example shows that each pass of machining, depending on which of the set of different tool types is employed, automatically requires different manufacturing methods and this results in a different set of manufacturing features. Beyond that, also each modification in the flow of machining can result in the definition of new manufacturing features. For example, when using an end miller for machining a hole, two different methods can be used: circular milling or one-pass milling. In general a particular choice of method specifies that a particular set of parameters must be assigned values so that the machining program – the actual moves to be executed by the tool – can be uniquely determined.

| Descent                           | Parameter                        | Value                           | Unit                    | Default Values          |
|-----------------------------------|----------------------------------|---------------------------------|-------------------------|-------------------------|
| Inherited<br>L<br>60<br>mm<br>100 | Inherited<br>B<br>18<br>mm<br>50 | Inherited<br>H<br>5<br>mm<br>10 | Dr<br>Constr<br>Mm<br>5 | Da<br>Constr<br>Mm<br>5 |

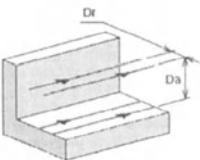


Figure 3. Geometrical representation and stored semantic information

To permit work piece-centered optimization of tool requirements, manufacturing features are organized in a hierarchical structure based on geometrical features, Figure 5.

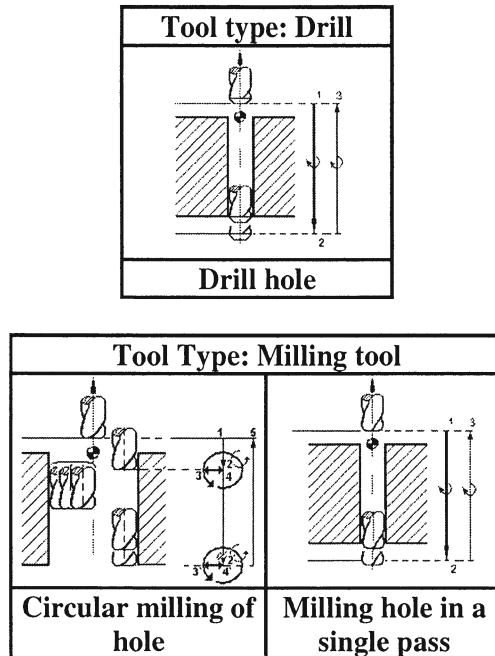


Figure 4. Manufacturing a hole with two different tool types.

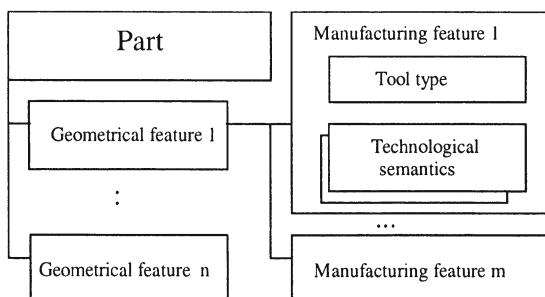


Figure 5. Hierarchical feature structure

Permanently assigning a manufacturing feature to a given three-dimensional geometry – the standard procedure followed by many CAD assistants – makes it very difficult to subsequently exchange this feature for

another variant. The present prototype, on the other hand, uses tables of preference criteria to assign the different manufacturing features to geometrical features.

As the development progressed, a simple atomic technological semantics structure proved insufficient for more complex machining tasks. Therefore an additional semantics level was defined in order to split manufacturing processes into individual process movements. For instance, technological information can be divided into "machining movements in the plane" and "machining movements at a right angle to the plane". This results in a flexible system for the management of a manufacturing feature's technological semantics. For example, it is then possible to define several feed motions for an end mill depending on the machining direction, or to assign information on the depth-setting movement.

### 2.3 TOOL-SPECIFIC KNOWLEDGE

Finally, in order to expect meaningful decisions by the selection of tools in the expert system, tool-specific knowledge must be represented as well. This information is not necessarily limited to the existing tools in the company. On the contrary, it also has to include information from the tool manufacturers. This can be realised with the integration of a neutral data format for tools such as STANDARD OpenBase. This tool neutral interface permits access of the tool catalog of various tool manufacturers.

Access to the existing tool databases is managed by a dedicated link mechanism. This mechanism permits the definition of aliases for parameter designations (field names) and provides unit conversion functions. The constraints are formulated uniformly but can still be applied regardless of the external database schema.

Additional categories of knowledge that are incorporated are information such as material data, cooling, lubricant, and machine parameters. Finally, company specific knowledge can be added, such as preferences for particular machining strategies that will lead to solutions that better fit local practices.

## 3. Knowledge Representation for Tool Selection

As noted by Darr et al. (1998), part selection is a fundamental problem in engineering design, in general describing a situation where off-the-shelf components are selected and assembled into some technical product. This general problem category is more commonly known as *configuration* (or configuration design) an active subarea of engineering design research, (Soininen 2001; Stumptner 2000).

The resemblance to the tool selection problem we are facing is obvious although in this particular case, the “artifact” to be assembled is not strictly physical (the same holds for many other configuration problems). The solution to the tool selection problem consists of both physical objects: work pieces and features with particular geometric and material characteristics; the choice of tools with particular parameter values selected from a manufacturer catalog or simply the list of locally available tools, but also a particular desired sequence of tool applications. This is not a full planning problem since choices such as the above will in general be highly constrained and come from a limited set of possibilities, but an explicit representation of and reasoning about such “process” entities is desirable. In addition, some attribute values will be computed rather than being selected from a fixed domain. Finally, a particular choice of procedure or manufacturing feature will influence the number of attributes and other entities in the problem.

A classic representation used for configuration problems with part selection characteristics is that of Constraint Satisfaction Problems (CSPs) (Mittal and Frayman 1989). CSPs are defined as consisting of a set of *variables*  $V$ , a set of *domains*  $D$  such that each  $v \in V$  has some  $d \in D$  assigned as domain, written  $\text{dom}(v)=d$ , and finally a set  $Z$  of constraints, where a constraint  $z$  refers to a particular set of variables  $v_1, \dots, v_n$  and restricts the set of values that can be assigned to these variables, i.e.,  $z \subseteq \text{dom}(v_1) \times \dots \times \text{dom}(v_n)$ . CSPs have the advantage of conceptual simplicity as well as a large amount of algorithms with well-known properties. However, the basic formalism is somewhat restrictive when dealing with domains where multiple entities of similar types exist (and may not be known to exist initially). Therefore we apply an extension that has been successfully used in the past.

### 3.1 GENERATIVE CONSTRAINTS

Generative Constraint Satisfaction Problems (GCSP), used in the COCOS/LAVA configuration tool (Fleischanderl et al. 1998) are described in Haselböck and Stumptner (1993) and Stumptner et al. (1998). They represent an extension of DCSP’s (Mittal and Falkenhainer 1990) that reifies complex objects in the context of CSPs by defining a separate set of *component variables*  $C$ . (Unlike the set of variables in a DCSP, the set of component variables in a GCSP is not predefined.) Each variable from  $C$  can be assigned a type  $\tau$  from  $T$ . For each  $\tau$ , there is a fixed set of *property variables*  $p$  such that if  $c$  from  $C$  is assigned  $\tau$  as type, then  $c.p$  is also a variable. In other words, by assigning values to types, the set of variables in the constraint network varies.

*Example:* Figure 2 shows a component (feature)  $c$  of type **Step**, i.e., the assignment  $c=\text{Step}$ . Because of this assignment, there exist property variables  $c.l$ ,  $c.b$ ,  $c.h$  describing the length, width, and height of the step feature represented by the component variable  $c$ . In this case, the domains of these three variables are numeric, i.e., belong to the set of *atomic* types, which means we refer to these three variables as *attribute variables*. The value assigned to variable  $c.l$ , for example, is 60.

If the domain of a property variable is  $C$ , the set of components, we refer to it as a *port* variable. In general, since the existence of property variable  $c.p$  depends on the type assigned to  $c$ , so does the domain of  $c.p$ .

The set  $C$  of components in a GCSP is variable; when a port variable requires a component of a particular type and none is available, then such a component is implicitly created (we then refer to it as *active*). Constraints are defined in a generic fashion for types of components, and when a component variable is assigned a type, the constraints associated with that type and its property variables are instantiated. (We do not discuss the more complex constraint types such as resource constraints here.) Like a normal CSP, a GCSP is *solved* when each variable is assigned a value and all constraints are satisfied. However, as with DCSPs, the set of active variables may grow during the search for a solution (and shrink again during backtracking).

For a more detailed discussion of GCSPs and their semantics, see Stumptner et al. (1998).

### 3.2 SOLVING GCSP'S

GCSP's can be solved using a backtracking constraint solver that creates components as needed during reasoning, and also observes the special semantics of property variables (which are, as mentioned above, created depending on the type assigned to a component). They can use adapted versions of the typical domain-independent propagation and filtering algorithms such as forward checking or various consistency algorithms (Tsang 1993; Bessiere, Freuder and Regin 1999).

As with other types of constraint problems, heuristic control knowledge can be easily integrated into the reasoning process by specifying orderings on the order in which variables are assigned, in which values from a domain are test, and in which different constraints are checked.

### 3.3 CONCEPT HIERARCHIES

Due to the fact that they effectively define an object-oriented representation language in CSP terms, GCSPs are both more powerful and more useful for

structured knowledge. In particular, they are naturally suited to expressing inheritance relationships between types, so that, e.g., the definition of property variables in the superclass is inherited by the subclass.

*Example:* In a component  $c$  describing the **MStep** manufacturing feature in Figure 3, the property variables  $c.l$ ,  $c.b$ , and  $c.h$  are inherited from the definition of the geometric **Step** feature.

The tool-specific knowledge necessary for the selection is organized in two concept hierarchies. The first hierarchy describes the tool types and defines the parameters that describe particular tools. It also contains the descriptions on how to access the manufacturer specific tool databases. The top level division in this hierarchy is between the three categories of drilling/sinking/filing tools, milling tools, and tap tools.

The second hierarchy is the hierarchy of the constraint types used to express the technological knowledge, which is subdivided along two different dimensions. This hierarchy can be expressed in GCSP terms by factoring out the behavior distinctions between constraints and representing them as components in the GCSP. In other words, control knowledge about particular types of constraints is stored via special components whose properties describe the type of constraint. In the current TOSS implementation, these are basically stored as a separate component hierarchy.

The first distinction in the component hierarchy is that into selection and calculation constraints. Selection constraints are those constraints which decide on the inclusion or exclusion of a tool in the final machining plan. The calculation constraints are used to calculate values for technological parameters, which are then available for further calculations, ranking of solutions, or direct incorporation into the final result (e.g. feed speed). The calculation constraints are in effect rules; they are assumed to be deterministic when all other values except for the computed one are known.

The second distinction is the one between obligatory and optional constraints. Obligatory constraints are constraints which are indispensable for the solubility of the decision algorithms. By the introduction of obligation parameters meaningful results are guaranteed. Optional constraints, if satisfied, will lead to a more accurate solution. Specifically, the set of possible values for particular positions in the hierarchy may be further constrained, but do not necessarily have to be unique. Constraints are subject to inheritance in the tool hierarchy just as parameters are, i.e., if a constraint is introduced at a particular node in the hierarchy, it is inherited by all lower levels. However, the expert has the possibility of overwriting the constraint on a subordinate level. A dedicated constraint editor supports knowledge base maintenance by permitting direct access to parameter values

and by executing consistency checks whenever new constraints are added to guarantee that the constraints are internally and mutually consistent.

### 3.4. OPTIMIZATION IN GCSPS

One crucial distinction to the traditional configuration-oriented use of CSP's is that the tool selection problem is heavily oriented towards finding optimized solutions. To accomodate optimization criteria requires a further extension of the GCSP formalism.

Formally, a Constraint Satisfaction Optimization Problem (Tsang 1993; Darr 1998) is defined as having a numerical optimality weight associated with each solution tuple, i.e., with each variable assignment that satisfies a particular constraint. Since we are interested in minimizing cost or time, solutions whose weight is minimal will be preferred in our case.

Optimization requires a value function which in our implementation can be provided by the user. In some case this utilizes explicit weight attributes as part of tool or feature descriptions. Since it is possible to include arbitrary attributes in a GCSP component, simply by extending the type definition to include the required property variables, this can be easily accommodated.

In general, solving a constraint optimization problem requires the computation of multiple solutions, with Branch and Bound methods used to expand only the more promising solutions (Tsang 1993). As more variables are assigned, the weight of a solution tuple will increase. The same can be done for GCSPs; notably the (potential) risk of unrestrained growth of a GCSP cannot occur in the optimization case since an infinitely large solution would automatically be inferior to a small solution with limited costs.

However, as also done by Darr (1998), in general heuristics will be used to keep the Branch-and-Bound method from deteriorating too much performance-wise. In our case, this was achieved via a set of quite strong heuristics – layering of constraints, and limitation of the size of locally examined solution subsets.

## 4. The Selection Process

In this section we will describe the general flow of the selection and optimization process and demonstrate the individual steps using a specific example part.

In a first step, the CAD/NC Assistant System detects a particular set of geometry features in a volume model. In the next step, the TOSS system adds the appropriate manufacturing features to the geometry features. Another possibility would be for a NC programmer to define the set of

manufacturing features by hand. Lastly, it is also possible to define the geometry features directly in the TOSS system regardless of which method is chosen to define the manufacturing features. However, the addition based on the features detected by the CAD/NC Assistant is the usual mode of operations.

Due to the complexity of the selection and optimization process, it was decided to split it into two phases early on, Figure 6.

**Phase I (Selection of suitable tools per feature):** This phase returns a weighted list of tools for each work piece. The list can either be used as input for Phase II or can be used for manual selection by the NC programmer.

**Phase II (Optimization per work piece, i.e., globally):** This Phase works on the basis of the lists produced by Phase I and computes a toolset that is optimized for the machining of the whole work piece.

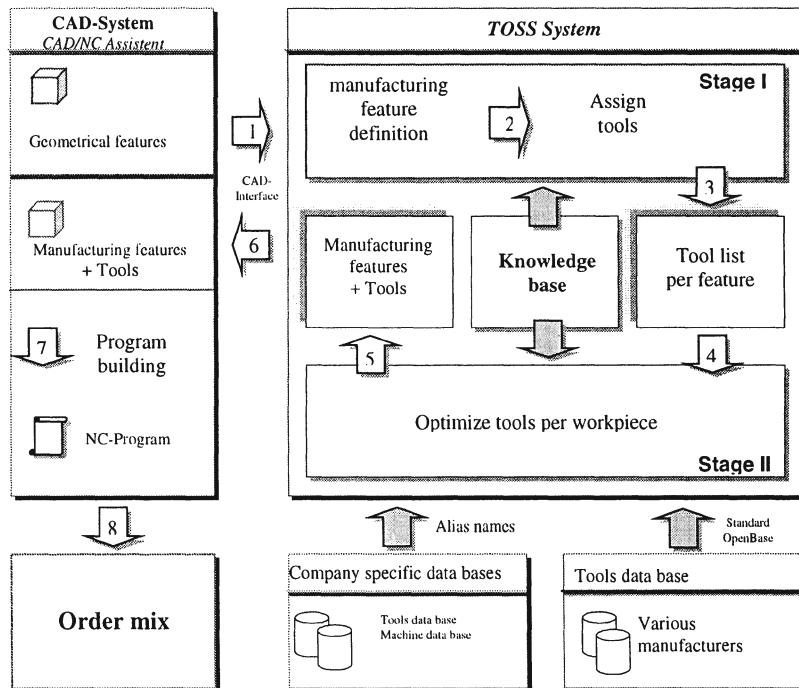


Figure 6. Flowchart of the selection and optimization process

#### 4.1 PHASE I

As mentioned above, manufacturing features incorporate the geometry provided by the CAD system, the machining strategy, and the assigned tool types. In this stage, the system selects the applicable tools for each feature from the the tool data base and sorts them according to their suitability. The computation of the suitability (i.e., the optimality value) occurs via freely programmable rules, i.e., formulas which compute the optimality (weight) parameter based on other properties of the tool and feature. The process can be considered to occur in four stages, which basically correspond to different groups of constraints being checked (and thereby applied to reduce the solution space):

1. **Selection according to tool type:** Tools that do not fit one of the tool types that can be used to manufacture that feature are rejected.
2. **Geometrical Selection:** This stage selects the geometrically compatible tools from the correct tool groups chosen in Step 1. The selection is based on constraints that are defined on the various tool attributes. Specifically, for a parameter constraints can restrict the reasonable minimum and maximum value (typically one constraint is responsible for each), and an optimal value can be defined within that interval. If defined, this value has priority and a tool with exact fit must be delivered if one exists.
3. **Technological Selection:** Continues from the state produced by geometrical selection and removes all tools that are not suitable from a machining stand point (i.e., which do not satisfy the technological constraints). In this stage, all technological parameters are computed as well.
4. **Sorting according to the optimization criterion:** Based on the optimization rules, the remaining tools are listed according to their suitability.

The final result of Phase I is therefore a list of tools for each individual manufacturing feature. These lists provide the domain values for further evaluation in Phase II.

#### 4.2 PHASE II

The goal of the second stage is optimization of the tool selection for the whole work piece. Although different optimization criteria can be defined, as discussed above, the usual effect will be that a reduction in the number of different tools is favored, e.g., to reduce the refit times when switching between tools, or to reduce the equipment costs. This stage again uses the general constraint optimization mechanism, again controlled by heuristics which preselect promising manufacturing features from the possible choices.

#### *4.2.1 Preselecting Manufacturing Features*

During Phase I, exactly one manufacturing feature is considered per geometry feature (basically to provide a preliminary ranking of tools),, regardless of whether the manufacturing features were assigned automatically or by the preparing worker. In Phase II, all remaining (i.e., not already deselected for technological reasons) manufacturing features are considered.

At the start of Phase II, the system therefore examines all manufacturing feature alternatives even if individual manufacturing features had been manually specified. Selections are ranked (a cutoff point can be specified if desired, e.g., at the level of the preselected features) and for all additionally selected manufacturing features, the full Phase I constraint processing is run (i.e., checking of geometrical and technological constraints and ranking).

Initial analysis showed that the experts in general assigned different weights to different manufacturing features. As a result, it was decided to use weights in two separate places, one for judging the relative suitability of the different manufacturing features (for the same geometrical feature) and one for judging the relative suitability of the different tool types for each class of manufacturing feature, Table 1.

TABLE 1. Segment from the geometrical feature list together with manufacturing features

| Geometrical feature | Manufacturing feature         | Weight | Tool Type         | Weight |
|---------------------|-------------------------------|--------|-------------------|--------|
| Step                | Mill step in one pass         | 35     | End Miller        | 40     |
| Step                | Mill step in one pass         | 35     | Cyl.Front Mill.   | 60     |
| Step                | Mill step in multiple passes  | 65     | End Miller        | 30     |
| Step                | \Mill step in multiple passes | 65     | Cyl.,Front Mill.. | 70     |

#### *4.2.2 Global optimization of tool requirements*

As stated above, the Phase II process has the full constraint optimization process executed, with the predefined optimization criteria used to rank solutions and point out the best combination, Figure 7.

An additional heuristic that can be used to speed processing with large tool databases is the assumption that in general only tools that are optimal for at least one feature *or* can be used for producing multiple geometry features will be found to participate in a globally optimal solution.

To use this procedure, an extensional constraint is defined which contains all tools for each manufacturing feature that survived Phase I. The constraint expresses all possible assignments of the tools to geometrical features that can be produced with them.

As described above, all tools which are not involved with a minimum number of features and are not optimal for at least one feature are pruned. The minimum number of features is chosen as a parameter (2 would be a typical value).

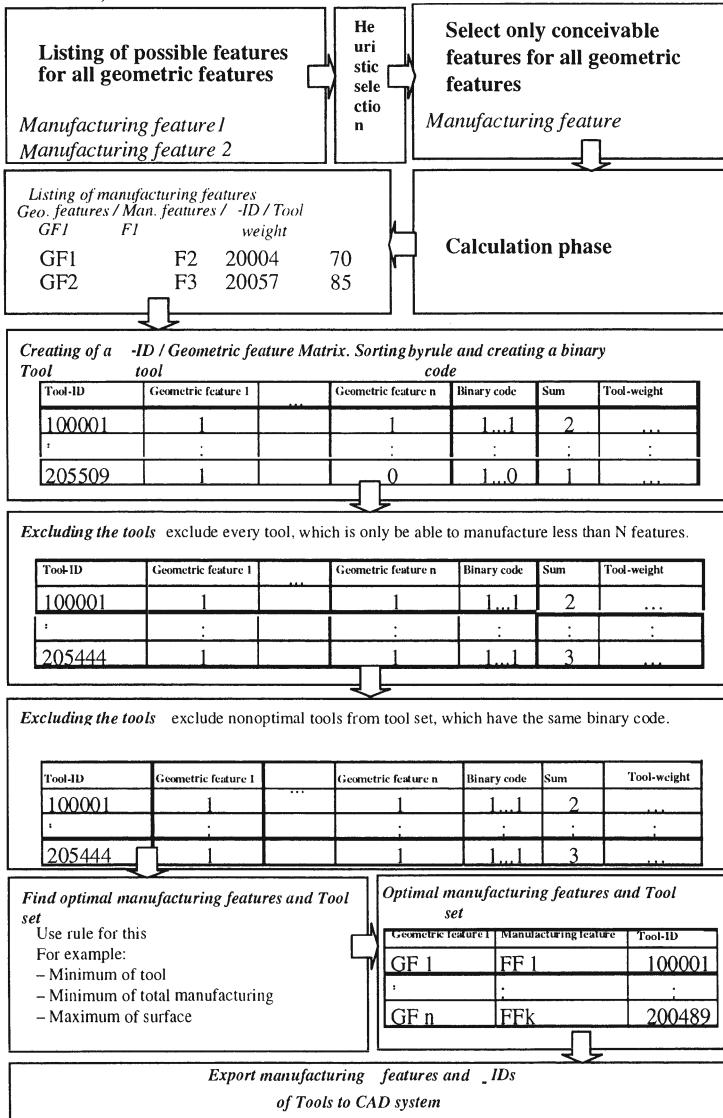


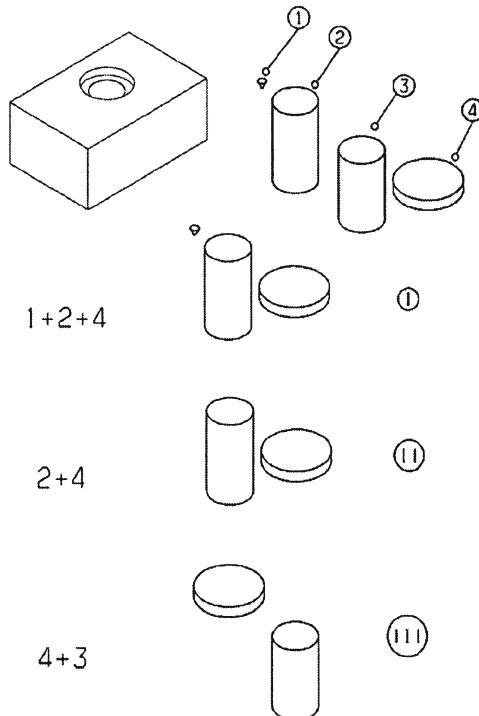
Figure 7. Flowchart of the global optimization of tool requirements (Phase II)

Interchangeable tools (i.e., those which work for the same features) are compared in terms of their utility value. From each group all but the tool with the highest utility are pruned. Of the remaining combinations of tools and features, the combination with the highest total utility is presented as best solution.

The utility formula can be specified by the user; it is possible to use different formulas for evaluating individual tools during the pruning above and tool combinations in the final step. The linear weighted utility function (Fishburn 1970) used by Darr et al. (1998) would be one possible choice.

#### *4.2.3 Allocation of new geometry feature versions*

A modification of the handling sequence of the geometrical features in a workshop can result in new geometrical features and this can result in new manufacturing features, new manufacturing methods and therefore new tool types. For example, if we have a counter bored hole, we can manufacture this geometrical feature in practice with three methods, Figure 8.



*Figure 8. Counter bored hole with its geometrical features*

Suppose that we have four geometrical features: a spot drill (#1), a long hole (#2), a short hole (#3) and a Counter hole (#4), Figure 8. In the first manufacturing method we order a spot drill (#1), a normal drill (#2) and milling operation (#4). In the second method we manufacture only the feature #2 (long hole) for example with a drill or mill tool and mill the feature #4. In the third manufacturing method we change the sequence of order and manufacture the geometrical features #4 at first and than drill the hole. Now we have a short hole (feature #3).

A administration of different versions of the geometrical features are difficult, because the expert system have to used fixed geometrical features, which are coming from CAD system. For the solution of this problem would be inserted a new level in the expert system TOSS. This new level assembles several geometrical features to a *geometrical feature family* together, Figure 9. This geometrical feature family describes all possible versions of the geometrical features with a unique processing order. If the expert system select a geometrical feature family then its geometrical features were be selected as input.

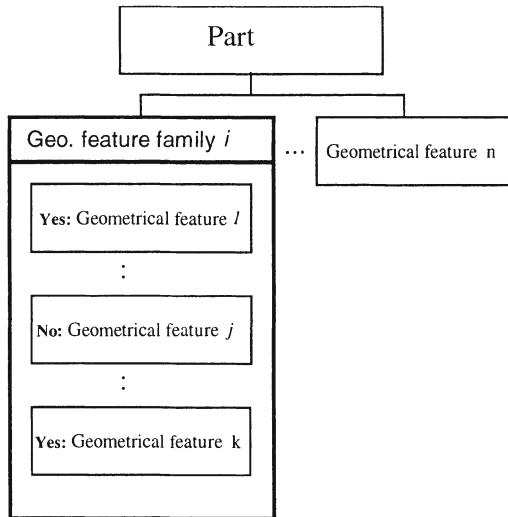


Figure 9. Geometrical feature family structure

## 5. An Example Run

This section presents a typical application session using slightly simplified example work piece that is to be subjected to drilling and milling operations.

The procedure followed by the TOSS system is contrasted with the reasoning process used by multiple experts.

Initially, some environmental parameters were chosen. The machine tool used was the EMCO VMC300, the material selected was categorized as "AlSi Cast alloys with below 10% Si" and the cooling agent chosen was "Oil Emulsion". The stock was a prismatic aluminium block with the dimensions 60 mm x 60 mm x 10 mm. The intended part required the machining of twelve features. The twelve features comprised eight different geometry features and six different manufacturing feature types, Figure 10 and Table 2.

TABLE 2. A geometrical feature family

| Geometrical feature family |                     |        |
|----------------------------|---------------------|--------|
| Variant Number             | Manufacturing order | Weight |
| 1                          | 1 & 2 & 4           | 40     |
| 2                          | 2 & 4               | 50     |
| 3                          | 4 & 3               | 10     |

| Feature | Number | Description                 |
|---------|--------|-----------------------------|
| 1       | 1      | Step L60 W18 H05            |
| 2       | 2      | Circular Pocket R20 H05     |
| 3       | 1      | Straight Groove L60 W09 H05 |
| 4       | 1      | Dead End Groove L10 R05     |
| 5       | 2      | Borehole Dm06 Dp10          |
| 6       | 1      | Dead End Groove L12 R2.5    |
| 7       | 2      | Borehole Dmx Dp05           |
| 8       | 2      | Borehole Dm10 Dp05          |

In this case, four experts were presented with the task and their choices recorded. The required answers were the definition of the handling strategy, the selection of the tool types and the determination of the optimal tool diameter  $D_{ideal}$  for the intended tool as well as the definition of the diameter limit values  $D_{min}$  and  $D_{max}$ . A further goal for the experts was to minimize the number of tools needed for machining the work piece (although this was not

an absolute requirement). The results produced by the experts are summarized in Table 3.

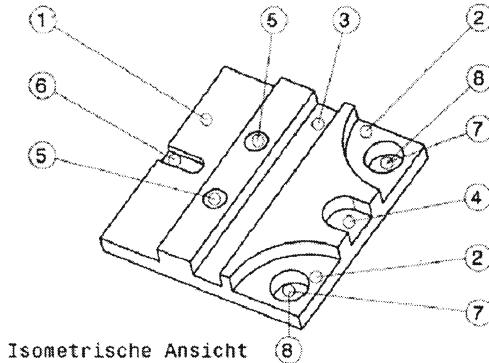


Figure 10. Example part with 8 geometrical features

TABLE 3. Tool Selection results produced by four experts (B: Drill, M: Mill-Dmin/Dideal/Dmax)

| Expert | Feature    |            |         |            |         |         |               |            | $\Sigma$ |
|--------|------------|------------|---------|------------|---------|---------|---------------|------------|----------|
|        | 1          | 2          | 3       | 4          | 5       | 6       | 7             | 8          |          |
| I      | M-20/30/30 | M-22/30/39 | M-6/9/9 | M-6/8/10   | B-6/6/6 | M-4/5/5 | -             | B-10/10/10 | 4M & 2 B |
| II     | M-22/40/40 | M-22/40/40 | M-5/5/5 | M-10/10/10 | B-6/6/6 | M-5/5/5 | -             | M-10/10/10 | 6M & 1 B |
| III    | M-20/20/40 | M-20/20/40 | M-8/8/8 | M-8/8/10   | B-6/6/6 | M-4/5/5 | B-9.8/9.8/9.8 | B-10/10/10 | 4M & 3 B |
| IV     | M-20/20/35 | M-20/40/40 | M-5/9/9 | M-5/10/10  | B-6/6/6 | M-5/5/5 | B-6/6/6       | B-10/10/10 | 5M & 3 B |

The table shows that the experts differ widely in the type of solutions produced. Expert I required six different tools when using the ideal values. Expert II and III needed seven tools and expert IV even needed eight tools.

The experts also reached different results as far as tool selection for each individual feature is concerned. This is well illustrated by feature 1, where all experts choose the same tool type but differ in terms of the upper limit and ideal value for the diameter. Two experts determine the ideal value rather at the lower limit, while the other experts see the ideal value as identical with the upper limit.

All four experts used global optimization process across features to reach a solution. Remarkable is that the tool minimization on only one tool (end miller 5 mm), which would be theoretically and technically, was not chosen by any expert.

The same project was specified for the TOSS system, by defining the geometry features via a CAD system. The allocation of manufacturing features and tool types were then executed automatically in TOSS. The starting situation before the optimization is shown in Table 4.

TABLE 4. Assigned manufacturing features and tool types

| Feature | Geometrical type        | Manufacturing feature description          | Tool type             | Weight |
|---------|-------------------------|--------------------------------------------|-----------------------|--------|
| 1       | Step                    | One-pass milling of step                   | end miller            | 50     |
| 2       | Quarter circular pocket | Rough-machining of quarter circular pocket | cylinder front miller | 100    |
| 3       | Groove                  | One-pass milling of groove                 | disk miller           | 80     |
| 4       | Dead end groove         | One-pass milling of dead end groove        | cylinder front miller | 80     |
| 5       | bore-hole               | Drilling                                   | spiral drill          | 90     |
| 6       | Dead end groove         | Milling of groove                          | end miller            | 80     |
| 7       | preparatory hole        | preparatory drilling                       | spiral drill          | 90     |
| 8       | bore-hole               | drilling                                   | spiral drill          | 90     |

TOSS produces a job list for each geometry feature that contains all manufacturing features selected beforehand and may also contain additional new manufacturing features, which are selected via a heuristic function by the system. For example, for the feature #3 “Groove” from Table 4, two further manufacturing features could be applicable, Table 5.

TABLE 5. Possible manufacturing features for the “groove” geometry feature

| Geometry feature | Description            | Tool type | weight |
|------------------|------------------------|-----------|--------|
| Groove           | One-pass milling       | Disk mill | 80     |
| Groove           | One-pass milling       | End mill  | 70     |
| Groove           | Back-and-forth milling | End mill  | 65     |

For each manufacturing feature in the job list, the appropriate tools were found in phase 1. TOSS subsequently identified the optimal global tool combination in phase 2, dependent on the the optimality measure. In this case the criterion was overall minimum tool cost.

The TOSS system calculated the following result at the end, Table 6.

TABLE 6: The result computed by TOSS

|        | Feature |      |     |     |     |     |     |      | $\Sigma$ |
|--------|---------|------|-----|-----|-----|-----|-----|------|----------|
|        | 1       | 2    | 3   | 4   | 5   | 6   | 7   | 8    |          |
| Tool   | M-20    | M-20 | M-5 | M-5 | B-6 | M-5 | B-6 | B-10 | 2M& 2 B  |
| Number | (1)     | (1)  | (2) | (2) | (3) | (2) | (3) | (4)  |          |

Total number of tools: 2 end milling tools (code M) and 2 spiral drills (code B)

The optimization process is best exemplified in the example by feature 1 and feature 2. For both, the expert system selected an end miller with 20 mm diameter, which is optimal for machining both features. The second optimization was made for features 3, 4 and 6. Total runtime was about a minute using a database of 13,000 tools.

For feature 3, the expert system selected another type of machining process than was determined in the start phase and exchanged the originally selected machining operation using a disk miller for the geometry feature "Groove" by another using an end miller. This end miller can also be used for the production of feature 4 and 6.

TOSS shows here a more accurate choice of tools. The system was able to execute a technically correct form of tool selection. It could not only optimize tools for the manufacturing features, it was specifically able to replace the original choices for a better final result than the human experts produced. A human expert is not able, or he doesn't have the time to himself calculate the complex technical parameters for each manufacturing feature and each tool. This means that the experts are only able to present approximately optimal tool sets based on their experience. The advantages of the expert system TOSS lie in the higher accuracy of its solution finding process. It compares the final results on base of the tool-IDs, and selects the optimal tool set within a short time from a very large number of tools.

## 6. Discussion

In terms of representation, this paper extends the GCSP-based configuration work of (Haselböck and Stumptner 1993; Stumptner, Friedrich and Haselböck 1998; Fleischanderl et.al 1998) by moving it into a new problem area. The dynamic properties of the GCSP are useful in modeling alternate feature arrangements with differing properties, and are extended since an optimization-oriented approach needed instead of the constraint-satisfaction-based earlier implementations.

In formal terms, the GCSP formalism (Haselböck and Stumptner 1993; Stumptner, Friedrich and Haselböck 1998) has a similar purpose as the MAD multiattribute CSP approach (Darr, Birmingham and Scala 1998). The GCSP approach provides a more direct modelling of objects and object properties, but is not as thoroughly analyzed in terms of formal properties compared to the backtracking-free subsets of the MAD approach. However, past industrial experience with the GCSP-based COCOS configuration tool indicates sufficient performance for practical application. Using effective heuristics, the performance of the TOSS system is suitable for interactive work. Multiattribute tuples are explicitly represented as entities in the MAD

approach, whereas they are composed from individual property variables in GCSPs. Since GCSPs generally assume complex interrelations between individual components this is not a direct drawback. As noted in (Darr, Birmingham and Scala 1998), the MAD approach does not explicitly manage which constraints are active at any given time whereas the goal of the GCSP is to provide a clear builtin activation semantics for components and constraints.

The TOSS system touches the area of intelligent design catalogs (Bradley, Agogino and Wood 1994). Their approach is more general, incorporating multimedia visualization and multilayered evaluation mechanisms to help users decide on the choice of near arbitrary parts from engineering catalogs. In our case, the application area is much more limited, however tool knowledge from commercial catalogs can be directly and automatically incorporated into the reasoning process, and TOSS produces solutions automatically.

## 7. Conclusion

This paper has described an automated tool selection system for numerically controlled machining, a crucial aspect of efficient production in contemporary machine shops. The constraint-based GCSP knowledge representation formalism, originally developed in a configuration context, was adapted to this new domain on the border between design and manufacturing by moving to an optimization-based view. The implementation uses a constraint propagator with special-purpose preprocessing stages and dedicated interfaces to commercial tool catalogs that permit the direct transfer of tool knowledge into the TOSS knowledge base. Experiments have shown that the system produces quality solutions that are comparable or better than those produced by human experts.

The insights gained in the context of the prototype development concerning software support of the tool selection on the basis of AI methods can be transferred directly to all machine cutting production of prismatic parts. The result shows that the methods of AI research can be used promisingly in the area of the tool selection in manufacturing.

## References

- Barker, V and O'Connor, DE: 1989, Expert systems for configuration at Digital: XCON and beyond, *Communications of the ACM* **32**(3): 298-318.
- Bessiere, C, Freuder, EC and Regin, C-J: 1999, Using constraint metaknowledge to reduce arc consistency computation, *Artificial Intelligence* **65**: 179-190.

- Bradley, S, Agogino, A and Wood, W: 1994, Intelligent engineering component catalogs, in JS Gero and F Sudweeks (eds), *Artificial Intelligence in Design'94*, Kluwer, Dordrecht, pp. 641-658.
- Darr, TP, Birmingham, WP and Scala, N: 1998, A MAD approach for solving part-selection problems, in JS Gero and F Sudweeks (eds), *Artificial Intelligence in Design'98*, Kluwer, Dordrecht, pp. 251-269.
- Fishburn, P.C.: 1970, *Utility Theory and Decision Making*, Wiley, New York.
- Fleischanderl, G, Friedrich, G, Haselböck, A, Schreiner, H and Stumptner, M: 1998, Configuring large-scale systems with generative constraint satisfaction, *IEEE Intelligent Systems* **13**(4): 59-68.
- Haselböck, A and Stumptner, M: 1993, A generative constraint formalism for configuration problems, *Proceedings AIIA '93*, Springer, Berlin.
- Heinrich, M and Jüngst, EW: 1991, A resource-based paradigm for the configuring of technical systems from modular components, *Proceedings of the 7<sup>th</sup> IEEE Conference on AI Applications (CAIA-91)*, pp. 257-264.
- Heinsohn, J and Socher-Ambrosius, R: 1999, *Wissensverarbeitung*, Spektrum Akademischer Verlag, Heidelberg-Berlin.,
- Jackson, P: 1987, *Expert Systems*, Addison-Wesley, Reading, MA.
- Mittal, S and Falkenhainer, B: 1990, Dynamic constraint satisfaction problems, *Proceedings 8th National Conference on Artificial Intelligence (AAAI-90)*, AAAI Press, pp. 25-32.
- Mittal, S and Frayman, F: 1989, Towards a generic model of configuration tasks, *Proceedings of the 11<sup>th</sup> International Joint Conference on AI (IJCAI-89)*, Morgan Kaufmann, San Mateo, CA, pp. 1395-1401
- Murtagh, N and Shimura, M: 1990, Parametric engineering design using constraint-based reasoning, *Proceedings 8th National Conference on Artificial Intelligence (AAAI-90)*, AAAI Press, pp. 505-510.
- Ovtcharova, J: 1997, Neue Perspektiven für die Feature-basierte Modellierung, *VDI-Z* **139**: 34-37. (in German).
- Rich, E: 1983, *Artificial Intelligence*, Mc Graw Hill, Singapore.
- Soininen, T (ed.):2001: *Working Notes, IJCAI Workshop on Configuration*, Seattle.
- Stumptner, M (ed.):2000: *Working Notes, ECAI Workshop on Configuration*, Berlin.
- Stumptner, M, Friedrich, G and Haselböck, A: 1998, Generative constraint-based configuration of large technical systems, *AI EDAM* **12**(4): 307-320.
- Stumptner, M, Haselböck, A, and Friedrich, G: 1994, COCOS - A tool for constraint-based, dynamic configuration, *Proceedings 10<sup>th</sup> IEEE Conference on AI Applications (CAIA-94)*, San Antonio, pp. 373-380.
- Tsang, E: 1993, *Foundations of Constraint Satisfaction*, Academic Press, London.
- Wright, JR, Weixelbaum, ES, Brown, K, Vesonder, G., Palmer, ST, Berman, JI and Moore, HG: 1993, A knowledge-based configurator that supports sales, engineering, and manufacturing at AT&T network systems, *Proceedings of the 5th Innovative Applications of AI Conference (IAAI-93)*, AAAI Press, 183-193.

## **COMPONENTS IN DESIGN AND DESIGN MODELS**

---

*Requirements specification and automated evaluation of dynamic properties of a component-based design*

Catholijn M Jonker, Jan Treur and Wouter C A Wijngaards

*Identifying component modules*

Robert I Whitfield, Joanne S Smith and Alex B Duffy

*Perspectors*

John Haymaker, Martin Fischer and John Kunz

*Product data exchange using ontologies*

Christel Dartigues and Parisa Ghodous

## REQUIREMENTS SPECIFICATION AND AUTOMATED EVALUATION OF DYNAMIC PROPERTIES OF A COMPONENT- BASED DESIGN

CATHOLIJN M JONKER, JAN TREUR AND WOUTER C A  
WIJNGAARDS  
*Vrije Universiteit Amsterdam*  
*The Netherlands*

**Abstract.** Within a design process, the evaluation of a candidate design solution against a set of requirements may be hard, especially when the requirements concern dynamic properties. For a component-based design, evaluation of the dynamics can be based on dynamic properties of the components, and the way in which they are connected. In this paper an automated approach to the evaluation of dynamic properties of a component-based design is presented. A declarative temporal modelling language to specify and analyse dynamic properties is offered. An executable subset of this language is defined, based on 'leads to' relations. If executable specifications in terms of leads to relations of dynamic properties of the (reusable) components within a component-based design are available, then automated support is offered for: (1) simulation of the overall dynamics based on the executable dynamic properties of the components, (2) evaluation of requirements in the form of required overall dynamic properties of a design against a number of execution traces of the design, and (3) proving properties of an overall component-based design from executable properties of its components. The paper presents methods, techniques and software for such support and illustrates these by an example from the application area of component-based software design.

### 1. Introduction

In some application areas of design, systems which have nontrivial dynamics are designed in a component-based manner. An example of such an application area is the design of component-based software for dynamic applications. In such application areas often components can be (re)used for which the dynamic properties are known. By composing a number of such components in a component-based design, the required overall dynamics is

obtained. If the dynamics required is not that simple, it is not straightforward how such dynamics relates to available reusable components and their dynamic properties. Therefore automated support for such design processes is desirable. Literature addressing automated support in the areas of model checking and verification in general can be found, for example, in (Henzinger et al. 1994; Bouajjani et al. 1996; Yovine 1997; Fisher 1994; Clarke et al. 2000; Manna and Pnueli 1995; Stirling 2001).

To support component-based design of dynamical systems, combining required nontrivial dynamics with the use of reusable components, a number of questions need to be addressed:

- How can dynamic properties of (reusable) components be specified?
- How can requirements on the dynamics of an overall design be specified?
- How can it be checked whether a given component-based design, with known dynamic properties of its components, fulfils a given requirement on its overall dynamics?
- To fulfill overall requirements on dynamics, how can proper reusable components be determined on the basis of their dynamic properties, and how can they be composed to obtain a component-based design fulfilling these overall requirements?

In this paper a rich temporal framework is used to address these questions (with an emphasis on the first three). The temporal framework provides a declarative (requirement) specification environment and is used for a variety of activities, such as temporal simulation, temporal model checking and temporal reasoning about dynamic properties. The application area used to illustrate the approach is the design of software, in particular a system of information agents.

Specification of dynamic properties of a component-based design has at least two different aspects of use. First, models for the dynamics can be specified to be used as a basis for *simulation*, also called executable models. These types of models can be used to perform (pseudo-)experiments on the basis of the design. Second, specification of dynamic properties of a system can be done in order to *analyse* its dynamics (i.e., to obtain automated support for requirements specification, verification and testing). These properties can play the role of requirements, and can be used, for example, in evaluation of sample behaviours of (realised or simulated) designs in the context of these requirements. These two different uses of specification of dynamic properties of a component-based design impose different desiderata on the languages in which these specifications are to be expressed.

A language for executable models should be formal, and as simple as possible, to avoid computational complexity. Expressivity can be limited. Software tools to support such a language serve as *simulation environment*. A language to analyse dynamic properties, on the other hand, should be sufficiently advanced to express various dynamic properties that are relevant. Expressivity should not be too limited; executability, however, is not required for such a language. What is important, though, is that properties specified in such a language can be checked for a given sample behaviour (e.g., a simulation run) without much work, preferably in an automated manner. Moreover, it is useful if a language to specify dynamic properties provides possibilities for further analysis of logical relationships between properties. For these reasons also a language to specify properties of the dynamics of a design should be formal, and at least partly supported by software tools (*analysis environment*).

In this paper two different temporal specification languages for dynamic properties are put forward and illustrated for an example application. In Sections 2 and 3 it is shown how the two languages (one aiming at analysis, the other one aiming at simulation) to model dynamics within a component-based design can be defined. An example component-based software design is used to illustrate the notions that are introduced. In Section 4 a model for the dynamics of this example design is presented. Section 5 addresses the use of the analysis environment for the example design. In Section 6 the use of the simulation environment is addressed. In Section 7 a discussion is included.

## 2. Specification of Dynamic Properties of a Component-Based Design

To specify dynamic properties of a design, the temporal trace language used in (Jonker and Treur 1998; Herlea et al. 1999) is adopted. This is a language in the family of languages to which also situation calculus (McCarthy and Hayes 1969; Reiter 2001), event calculus (Kowalski and Sergot 1986), and fluent calculus (Hölldobler and Thielscher 1990) belong. In Section 2.1 the Temporal Trace Language **TTL** is introduced. This language is more expressive than modal and temporal languages as described, for example, in (Henzinger et al. 1994; Bouajjani, Lakhnech and Yovine 1996; Yovine 1997; Fisher 1994; Clarke et al. 2000; Manna and Phueli 1995; Stirling 2001); see Section 6 for more discussion about expressivity. In Section 2.2 an example component-based software design (of a multi-agent system of information agents) is introduced to illustrate the language and its use.

## 2.1. THE TEMPORAL TRACE LANGUAGE TTL

An *ontology* is a specification (in order-sorted logic) of a vocabulary (also called a signature). A state for ontology  $\text{Ont}$  is an assignment of truth-values {true, false} to the set of ground atoms  $\text{At}(\text{Ont})$ . The *set of all possible states* for ontology  $\text{Ont}$  is denoted by  $\text{STATES}(\text{Ont})$ . The standard satisfaction relation  $\models$  between states and state properties is used:  $s \models p$  means that state property  $p$  holds in state  $s$ .

To describe behaviour, explicit reference is made to time in a formal manner. A fixed *time frame*  $\tau$  is assumed which is linearly ordered. Depending on the application, it may be dense (e.g., the real numbers), or discrete (e.g., the set of integers or natural numbers or a finite initial segment of the natural numbers), or any other form, as long as it has a linear ordering. A *trace*  $T$  over an ontology  $\text{Ont}$  and time frame  $\tau$  is a mapping  $T : \tau \rightarrow \text{STATES}(\text{Ont})$ , i.e., a sequence of states  $T_t (t \in \tau)$  in  $\text{STATES}(\text{Ont})$ . The set of all traces over ontology  $\text{Ont}$  is denoted by  $\text{TRACES}(\text{Ont})$ , i.e.,  $\text{TRACES}(\text{Ont}) = \text{STATES}(\text{Ont})^\tau$ .

States of a trace can be related to state properties via the formally defined satisfaction relation  $\models$  between states and formulae. Comparable to the approach in situation calculus, the sorted predicate logic *Temporal Trace Language* **TTL** is built on atoms referring to traces, time and state properties, such as

$$\text{state}(T, t, \text{output}(C)) \models p.$$

This expression denotes that state property  $p$  is true at the output of component  $C$  in the state of trace  $T$  at time point  $t$ . Here  $\models$  is a predicate symbol in the language (in infix notation), comparable to the *Holds*-predicate in situation calculus. Temporal formulae are built using the usual logical connectives and quantification (for example, over traces, time and state properties). The set  $\text{TFOR}(\text{Ont})$  is the set of all *temporal formulae* that only make use of ontology  $\text{Ont}$ . We allow additional language elements as abbreviations of formulae of the temporal trace language. Ontologies can be specific for a component.

## 2.2 AN EXAMPLE COMPONENT-BASED SOFTWARE DESIGN

For the example, consider a component-based design for a system of two software agents A and B (represented in the design by two components; see Figure 1) that participate in a small project: they each have to acquire some information (in the External World, represented by a third component in the design) and in cooperation they make up a concluding report on some topic. Each of the agents has access to useful sources of information, but this differs for the two agents. By co-operation they can benefit from the

exchange of information that is only accessible to the other agent. If both types of information are combined, the relevant conclusions can be drawn that would not have been achievable for each of the agents separately. A relevant property of such a component-based design is successfulness:

- What makes the difference for such a co-operation to succeed or to fail?
- Which dynamic properties of the components are relevant for being successful?
- How does successfulness relate to these dynamic properties?

For example, one of the agents, say A, may not be pro-active in its individual search for information. This might be compensated if the agent B is pro-active in requesting the other agent for information, but then at least A has to be reactive (and not entirely inactive) in information acquisition. Also other reasons for failure may exist. For example, one of the agents may not be willing to share its acquired information with the other agent. Yet another reason for failure may be that although both agents are active in searching and exchanging information, none of them is combining different types of information and deduce new conclusions. So, dynamic properties of the design as a whole depend in a crucial manner on the dynamic properties of the components, in this case the proper pro-activeness and reactivity properties that play a role in the dynamics of the cooperation process of the agents.

Summarizing, this example component-based design as depicted in Figure 1 is composed of three components: two information agents A and B and a component EW representing the External World. In this figure the boxes denote system components. During processing of the system, each component has at each point in time an input state and an output state. The arrows depict output-input connections: channels taking care that information available at an output state is also made available (possibly with some small delay) for the connected input state. Each of the agents is able to acquire partial information in the External World by initiated information acquisition, this is modelled by an arrow from the agent to the External World transferring information on what is to be acquired, and by an arrow back transferring information on the results of the acquisition.

Each agent's own observations are insufficient to draw conclusions of a desired type, but the combined information of both agents is sufficient: they have to co-operate to be able to draw conclusions. Therefore communication is required; each agent can communicate results of its own information acquisition and requests for information to the other agent: the arrows between the agents.

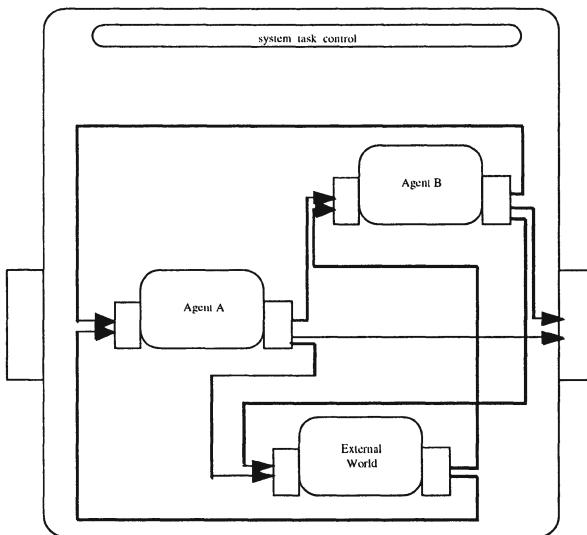


Figure 1. The example component-based design

#### *Assumptions on the notion of component-based design*

The above description gives an indication of the – very general – assumptions that are made on the notion of component-based design. This is a design structure based on *components* with at each point in time *input states* and *output states* that are *connected* by arrows to form the design. An arrow *modifies* the destination state at each point in time that the source state has changed, by inserting the changed source state in it.

For reasons of presentation, this by itself quite common situation for co-operative information agents is materialised in the following more concrete form. The world situation consists of an object that has to be classified. One agent can observe only the bottom view of the object, the other agent the side view. By exchanging and combining acquired information on the object they are able to classify the object. For example, if A observes a circle, and B observes a square, then, if A and B cooperate, together they can conclude that the object is a cylinder.

Communication from the agent A to B takes place in the following manner:

- the agent A generates at its output state a statement of the form:  
 $\text{communication\_from\_to}(<\text{type}>, <\text{atom}>, <\text{sign}>, \text{A}, \text{B})$
- this information is transferred to the input state of B using the arrow from A to B

In the example `<type>` can be filled with a label `request` or `world_info`, `<atom>` is an atom expressing information on the world, and `<sign>`, is one of `pos` or `neg`, to indicate truth or falsity. Instances of communication information that can be expressed are:

```
communication_from_to(request, view(B,circle), pos, A, B)
communication_from_to(world_info, view(B,circle), neg, B, A)
```

Here the object atom `view(B,circle)` expresses the world information that the view of the object visible for B is a circle. Interaction between an agent A and the External World takes place as follows:

- the agent A generates at its output state a statement of the form:

```
to_be_acquired_by(<atom>, A)
```

- the information is transferred to the input state of EW
- the External World EW generates at its output state a statement of the form:

```
acquired_information_for(<atom>, <sign>, A)
```

- the information is transferred to A

Instances of information on information acquisition for an agent A that can be expressed are:

```
to_be_acquired_by(view(A,circle), A)
acquired_information_for(view(A,circle), pos, A)
```

The output of an agent can include conclusions about the classification of the object of the form `conclusion(object_type(O))`; these are transferred to the output of the system as a whole.

In Table 1 an informal sketch of an example trace showing the dynamics of the component-based design is depicted. Here for simplicity the transfer of information between components is assumed to take no time (in contrast, in the simulation presented in Section 6, the transfer does take time). In this example trace, at time point 1 agent A takes two initiatives (see A's output state): (1) to acquire information and (2) to request information from B. These initiatives are immediately (same time point) received at the input of the External World component and at the input of B. As a result the External World provides the information (output at time point 2), and agent B starts an information acquisition process (output at time point 2). Immediately (at time point 2) the acquired information is received by A's input, and the information acquisition initiation of B is received at the External World component's input. As a result, the External World provides the information for B (time point 3), and this immediately is transferred to B's input. As a result, B communicates this information to A (time point 4). Finally, A, combining the gathered information, draws the conclusion.

TABLE 1. An example trace of a cooperative information gathering process

| Time point                     | 1                                                 | 2                           | 3                         | 4                               | 5                                                 |
|--------------------------------|---------------------------------------------------|-----------------------------|---------------------------|---------------------------------|---------------------------------------------------|
| Agent A input<br>output        | acquisition initiation by A;<br>request of A to B | information acquired by A   |                           | information acquired by B for A | conclusion combining the two types of information |
| Agent B input<br>output        | request of A to B                                 | acquisition initiation by B | information acquired by B | information acquired by B for A |                                                   |
| External World input<br>output | acquisition initiation by A                       | acquisition initiation by B | information acquired by A | information acquired by B       |                                                   |

The example trace discussed above shows a successful cooperation. In virtue of which dynamic properties of the components was this success possible? This question will be addressed in Section 3. In the remainder of this section, to illustrate the use of the temporal trace language, some dynamic properties for this component-based design as a whole are shown. Expressed both informally and formally, two of these example properties state the following:

- *Successfulness*

For any trace of the system, there exists a point in time such that in this trace at that point in time the system will provide a conclusion.

Using the Temporal Trace Language TTL, this is formally expressed by:

$$\forall \mathcal{T} : \text{TRACES} \exists t \exists O : \text{OBJECT} \\ \text{state}(\mathcal{T}, t, \text{output}(S)) \models \text{conclusion}(\text{object\_type}(O))$$

- *Cooperation necessity*

For any trace of the system and any point in time, if in this trace agent A provides the relevant conclusion, then at an earlier point in time agent B has communicated to agent A information acquired by B.

This is formally expressed by:

$$\begin{aligned}
 & \forall \mathcal{T} : \text{TRACES} \ \forall O1 : \text{OBJECT} \ \forall t \\
 & \text{state}(\mathcal{T}, t, \text{output}(A)) \models \text{conclusion}(A, \text{object\_type}(O1)) \Rightarrow \\
 & \exists t' < t \ \exists O2 : \text{OUTLINE} \\
 & \text{state}(\mathcal{T}, t, \text{output}(B)) \models \text{communication\_from\_to(world\_info, view}(B, O2), \text{pos}, B,
 \end{aligned}$$

A)

Other properties considered for this example are correctness and conservatism; see Section 5 for more details of these properties.

### 3. An Executable Language to Specify Dynamic Properties for Simulation

To obtain an executable language, in comparison with the temporal trace language discussed above strong constraints are imposed on what can be expressed. These constraints define a temporal language within the paradigm of executable temporal logic; cf. (Barringer et al. 1996). Roughly spoken, in this executable language it can only be expressed that if a certain state property holds for a certain time interval, then this *leads to* the situation that after some delay another state property should hold for a certain time interval. The use of real-valued parameters for time durations and delays is an extension, compared to the languages described in (Barringer et al. 1996). This specific temporal relationship  $\bullet \rightarrow$  (*leads to*) is definable within the temporal trace language TTL. This definition is expressed in two parts, the forward in time part and the backward in time part. Time intervals are denoted by  $[x, y]$  (from and including  $x$ , to but not including  $y$ ) and  $[x, y]$  (the same, but includes the  $y$  value).

#### Definition (The leads to relationship $\bullet \rightarrow$ )

Let  $\alpha$  and  $\beta$  be state properties, and let  $P1$  and  $P2$  refer to parts of the design (e.g., input or output of particular components). Then  $\beta$  *follows*  $\alpha$ , denoted by  $P1:\alpha \rightarrow_{e, f, g, h} P2:\beta$ , with time delay interval  $[e, f]$  and duration parameters  $g$  and  $h$  if (see also Figure 2):

$\forall \mathcal{T} : \text{TRACES} \ \forall t1 :$

$$\begin{aligned}
 & [\forall t \in [t1 - g, t1] : \text{state}(\mathcal{T}, t, P1) \models \alpha \Rightarrow \\
 & \exists \lambda \in [e, f] \ \forall t \in [t1 + \lambda, t1 + \lambda + h] : \text{state}(\mathcal{T}, t, P2) \models \beta]
 \end{aligned}$$

Conversely, the state property  $\beta$  *originates from* state property  $\alpha$ , denoted by  $P1:\alpha \leftarrow_{e, f, g, h} P2:\beta$ , with time delay in  $[e, f]$  and duration parameters  $g$  and  $h$  if

$\forall \mathcal{T} : \text{TRACES} \ \forall t2 :$

$$\begin{aligned}
 & [\forall t \in [t2, t2 + h] : \text{state}(\mathcal{T}, t, P2) \models \beta \Rightarrow \\
 & \exists \lambda \in [e, f] \ \forall t \in [t2 - \lambda - g, t2 - \lambda] : \text{state}(\mathcal{T}, t, P1) \models \alpha]
 \end{aligned}$$

If both  $P1:\alpha \xrightarrow{e,f,g,h} P2:\beta$ , and  $P1:\alpha \bullet \xrightarrow{e,f,g,h} P2:\beta$  hold, this is called a *leads to* relation and denoted by  $P1:\alpha \bullet \xrightarrow{e,f,g,h} P2:\beta$ . Sometimes also conjunctions or negations on one of the sides (or both) of the arrow are used.

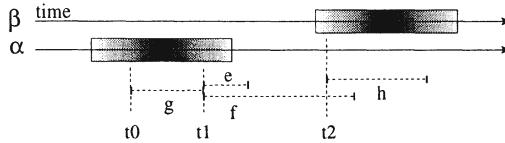


Figure 2. The time relationships between the parameters

Notice that a special case is when the state property  $\alpha$  is taken the trivial property true that is always (at every time point in every trace) true. In this case a leads to relation specifies that the property  $\beta$  will be *initiated* or *realized*. Examples of leads to relations in the context of the example component-based design as described in Section 2.2 are the following. Here the dynamic properties are formulated for a given agent A with respect to the other agent B; similar properties will hold for agent A's cooperation partner B.

#### *Information acquisition reactive*

A request to an agent A by another agent B for information that can be acquired by A leads to acquisition of this information by agent A. Formally expressed:

$\forall O : \text{OUTLINE}$

$\text{input}(A):\text{communication\_from\_to(request, view(A, O), pos, B, A)} \bullet \xrightarrow{5,5,10,10} \text{output}(A):\text{to\_be\_acquired(view(A, O), A)}$

#### *Request pro-active*

Agent A initiates communication to agent B of a request for information that B is able to acquire. Formally expressed:

$\forall O : \text{OUTLINE}$

$\text{true} \bullet \xrightarrow{5,5,10,10} \text{output}(A):\text{communication\_from\_to(request, view(B, O), pos, A, B)}$

More examples can be found in Section 4.

The definition of the relationships as given above can be applied to situations where the sources hold for longer than the minimum interval length  $g$ . The result for a longer duration of  $\alpha$  for  $P1:\alpha \bullet \xrightarrow{e,f,g,h} P2:\beta$  is depicted in Figure 3. The additional duration that the source holds, is also added to the duration that the result will hold, provided that the condition  $e + h \geq f$  holds. This is because the definition can be applied at each subinterval of  $\alpha$ , resulting in many overlapping intervals of  $\beta$ . The end result is that the

additional duration also extends the duration that the resulting notion  $\beta$  holds.

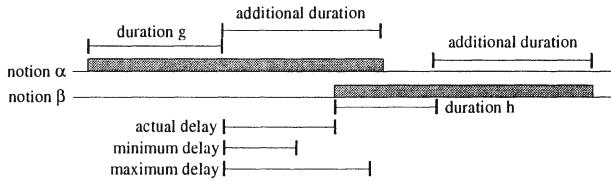


Figure 3. Temporal relationships for longer durations

#### 4. An Executable Model for the Example Component-Based Design

To obtain an executable model for the dynamics of a component-based design, two types of dynamic properties in leads to format are used: properties for the dynamics of each of the components, and properties that define the dynamics of the connections between components.

##### 4.1 DYNAMIC PROPERTIES OF THE COMPONENTS

For the example design, the components have dynamic properties as listed below. This does not mean that every possible component has all these properties. Rather, each specific candidate for a component is characterised by a subset of these properties. First the possible properties for the agents are discussed. They are expressed for agent A (with respect to the other agent B and the External World).

###### **Information acquisition reactive**

A request to an agent A by another agent B for information that can be acquired by A leads to acquisition of this information by agent A. Formally expressed as:

$\forall O : \text{OUTLINE}$

input(A) : communication\_from\_to(request, view(A, O), pos, B, A)  $\bullet\rightarrow_{5,5,10,10}$   
 output(A) : to\_be\_acquired(view(A, O), A)

###### **Information acquisition pro-active**

Agent A initiates the acquisition of information it is able to acquire. Formally expressed as:

$\forall O : \text{OUTLINE}$

true  $\bullet\rightarrow_{5,5,10,10}$  output(A) : to\_be\_acquired(view(A, O), A)

### **Request pro-active**

Agent A initiates communication to agent B of a request for information that B is able to acquire. Formally expressed as:

$\forall O : \text{OUTLINE}$

true  $\bullet\rightarrow_{5,5,10,10}$  output(A) : communication\_from\_to(request, view(B, O), pos, A, B)

### **Information provision reactive**

A request to an agent A by another agent B for information that can be acquired by A, together with receiving this acquired information by agent A leads to communication of this information to agent B. Formally expressed as:

$\forall O : \text{OUTLINE}$

input(A):communication\_from\_to(request, view(A, O), pos, B, A)  $\wedge$   
 input(A) : acquired\_information\_for(view(A, O), pos, A)  $\bullet\rightarrow_{5,5,10,10}$   
 output(A) : communication\_from\_to(world\_info, view(A, O), pos, A, B)

### **Information provision proactive**

Receiving acquired information by agent A leads to communication of this information to agent B. Formally expressed:

$\forall O : \text{OUTLINE}$

input(A) : acquired\_information\_for(view(A, O), pos, A)  $\bullet\rightarrow_{5,5,10,10}$   
 output(A) : communication\_from\_to(world\_info, view(A, O), pos, A, B)

### **Conclusion proactive**

Receiving acquired information by agent A and receiving by communication information acquired by agent B leads to combining this information in drawing a conclusion. Which specific conclusion is to be drawn depends on the input information. Therefore, this property consists of a set of specific properties (instances), for example in formal form:

input(A) : acquired\_information\_for(view(A, square), pos, A)  $\wedge$   
 input(A) : communication\_from\_to(world\_info, view(B, square), pos, B, A)  $\bullet\rightarrow_{5,5,10,10}$   
 output(A) : conclusion(object\_type(cube))

The last property is a property of the External World component.

### **Information acquisition effectiveness**

This is a property of the External World component that guarantees that every information acquisition initiative results in the acquired information. Formally expressed:

$\forall X : \text{AGENT} \forall O : \text{OUTLINE} \forall S : \text{SIGN}$

input(EW) : to\_be\_acquired(view(X, O), X)  $\wedge$   
 output(EW) : true\_in\_world(view(X, O), S)  $\bullet\rightarrow_{5,5,10,10}$   
 output(EW) : acquired\_information\_for(view(X, O), S, X)

## 4.2 DYNAMIC PROPERTIES FOR CONNECTIONS BETWEEN COMPONENTS

In addition to dynamic properties of components, connection properties are used to express the dynamics of the connections between the components in the design. These have a common pattern, instantiated for the particular components and the state properties involved at the connected output and input states. A possibility is to take zero time delay for connections. However, to be more general, nonzero delays in connections are allowed as well.

### Connection of one agent to the other

$\forall A : \text{AGENT} \forall B : \text{AGENT} \forall C : \text{COMMUNICATION\_TYPE} \forall I : \text{INFO}$   
 $[A \neq B] \Rightarrow$

output(A) : communication\_from\_to(C, I, S, A, B)  $\bullet\rightarrow_{5,5,10,10}$   
input(B) : communication\_from\_to(C, I, S, A, B)

### Connection of an agent to the External World

$\forall A : \text{AGENT} \forall I : \text{INFO}$   
output(A) : to\_be\_acquired\_by(I, A)  $\bullet\rightarrow_{5,5,10,10}$   
input(EW) : to\_be\_acquired\_by(I, A)

### Connection of the External World to an agent

$\forall A : \text{AGENT} \forall I : \text{INFO} \forall S : \text{SIGN}$   
output(EW) : acquired\_information\_for(I, S, A)  $\bullet\rightarrow_{5,5,10,10}$   
input(A) : acquired\_information\_for(I, S, A)

### Connection of an agent to the system output state

$\forall A : \text{AGENT} \forall O : \text{OBJECT}$   
output(A) : conclusion(object\_type(O))  $\bullet\rightarrow_{5,5,10,10}$   
output(S) : conclusion(object\_type(O))

## 5. Analysis Environment

Apart from an editor to specify dynamic properties, the analysis environment includes two parts; all these tools assume a finite time frame:

1. a tool that, given a set of traces (e.g., generated by simulation based on the component properties or testing of a prototype realisation), checks any dynamic property of a component-based design expressed in the Temporal Trace Language TTL. In additional tool has been developed that, given a trace checks for any property expressed in terms of  $\bullet\rightarrow$  where exactly in the trace this property fails.
2. a tool that, given an executable specification, for any dynamic property in leads to format proves or disproves whether it is entailed by the dynamic properties in leads to format of the components.

### 5.1. CHECKING A DYNAMIC PROPERTY AGAINST A SET OF TRACES

This application assumes that at a certain point in time within a design process both a set of requirements for dynamic properties of a design as a whole, and a generated candidate for the design are available. To check whether a required dynamic property for the design (expressed by a temporal formula in TTL with a universally quantified variable  $\mathcal{T}$  for a trace) is fulfilled in a given trace or set of traces, a software environment based on some Prolog and C programmes (of about 4000 lines) has been developed. Within this program temporal formulae from TTL are represented by nested term structures based on the logical connectives. Specification of properties is supported by an editor in which properties in terms of TTL can be expressed more or less as written below.

#### **GR1 Successfulness**

For any trace of the system, there exists a point in time such that in this trace at that point in time the system will provide the relevant conclusion.

Using the Temporal Trace Language TTL, this is formally expressed by:

$$\forall \mathcal{T} : \text{TRACES } \exists O : \text{OBJECT } \exists t \\ \text{state}(\mathcal{T}, t, \text{output}(S)) \models \text{conclusion}(\text{object\_type}(O))$$

This can also be expressed in simple format as follows:

$$\exists O : \text{OBJECT } \text{true} \rightarrow_{0,1000,10,10} \text{output}(S) : \text{conclusion}(\text{object\_type}(O))$$

#### **GR2 Correctness**

For any trace of the system and any point in time, if in this trace the system provides information at  $t$ , then this information is correct. Formally expressed in TTL format by:

$$\forall \mathcal{T} : \text{TRACES } \forall O : \text{OBJECT } \forall t \\ \text{state}(\mathcal{T}, t, \text{output}(S)) \models \text{conclusion}(\text{object\_type}(O)) \Rightarrow \\ \text{state}(\mathcal{T}, t, \text{output}(EW)) \models \text{true\_in\_world}(\text{object\_type}(O), pos)$$

This can also be expressed in simple format as follows:

$$\forall O : \text{OBJECT} \\ \text{output}(W) : \text{true\_in\_world}(\text{object\_type}(O), pos) \bullet_{0,1000,10,10} \\ \text{output}(S) : \text{conclusion}(\text{object\_type}(O))$$

#### **GR3 Conservatism**

For any trace of the system and any point in time, if in this trace the system provides a conclusion at  $t$ , then for all points in time  $t'$  later than  $t$  in this trace at  $t'$  the system provides the same conclusion. Formally expressed in TTL format by:

$$\forall \mathcal{T} : \text{TRACES } \forall O : \text{OBJECT } \forall t \\ \text{state}(\mathcal{T}, t, \text{output}(S)) \models \text{conclusion}(\text{object\_type}(O)) \Rightarrow \\ \forall t' > t \text{ state}(\mathcal{T}, t', \text{output}(S)) \models \text{conclusion}(\text{object\_type}(O))$$

This can also be expressed in simple format as follows:

$$\forall O : \text{OBJECT} \quad \text{output}(S) : \text{conclusion}(\text{object\_type}(O)) \implies_{1,1,1,1} \text{output}(S) : \text{conclusion}(\text{object\_type}(O))$$

#### GR4 Cooperation necessity

For any trace of the system and any point in time, if in this trace agent A provides a conclusion, then at an earlier point in time agent B has communicated to agent A information acquired by B. Formally expressed in TTL format by:

$$\begin{aligned} \forall T : \text{TRACES} \quad \forall O_1 : \text{OBJECT} \quad \forall t \\ \text{state}(T, t, \text{output}(A)) \models \text{conclusion}(\text{object\_type}(O_1)) \Rightarrow \\ \exists t' < t \quad \exists O_2 : \text{OUTLINE} \\ \text{state}(T, t', \text{output}(B)) \models \text{communication\_from\_to(world\_info, view}(B, O_2), \text{pos}, B, A) \end{aligned}$$

This can also be expressed in simple format as follows:

$$\begin{aligned} \forall O_1 : \text{OBJECT} \quad \forall O_2 : \text{OUTLINE} \quad \forall A : \text{AGENT} \quad \forall B : \text{AGENT} \\ [A \neq B] \Rightarrow \end{aligned}$$

$$\begin{aligned} \text{output}(B) : \text{communication\_from\_to(world\_info, view}(B, O_2), \text{pos}, B, A) \bullet_{0,1000,10,10} \\ \text{output}(A) : \text{conclusion}(\text{object\_type}(O_1)) \end{aligned}$$

As an example, the specified successfulness property is represented as a nested term structure internally within the software environment:

$$\text{forall}(M, \text{ex}(T_2, \text{INF}), \text{holds}(\text{state}(M, T_2, \text{output}(S)), \text{communication\_from\_to}(\text{INF}, S:\text{SYSTEM}, U:\text{USER}), \text{true}))$$

Traces are represented by sets of Prolog facts of the form  
 $\text{holds}(\text{state}(m1, t(2), \text{input}(\text{component})), a), \text{true}.$

where m1 is the trace name, t(2) time point 2, and a is a state formula in the ontology of the component's input. It is indicated that state formula a is true in the component's input state within the design at time point T2. The Prolog programme for temporal formula checking uses Prolog rules such as

$$\begin{aligned} \text{sat}(\text{and}(F, G)) &\dashv \text{sat}(F), \text{sat}(G). \\ \text{sat}(\text{not}(\text{and}(F, G))) &\dashv \text{sat}(\text{or}(\text{not}(F), \text{not}(G))). \\ \text{sat}(\text{or}(F, G)) &\dashv \text{sat}(F). \\ \text{sat}(\text{or}(F, G)) &\dashv \text{sat}(G). \\ \text{sat}(\text{not}(\text{or}(F, G))) &\dashv \text{sat}(\text{and}(\text{not}(F), \text{not}(G))). \end{aligned}$$

that reduce the satisfaction of the temporal formula finally to the satisfaction of atomic state formulae at certain time points, which can be read from the trace representation.

Another part of the program takes a trace of a component-based design as input, and specifications of dynamic properties (assumed in executable format) and generates an interpretation of what happens in this trace: it provides a check where exactly these dynamic properties actually hold in that trace.

The program marks any deficiencies in the trace compared with what should be there due to the temporal relationships. If a relationship does not hold completely, this is marked by the program. The program produces

yellow marks for unexpected events. At these moments, the event is not produced by any temporal relationship; the event cannot be explained. The red marks indicate that an event has not happened, that should have happened. In addition to checking whether the rules hold, the checker produces an informal reading of the trace. The reading is automatically generated, using a simple substitution, from the information in the given trace.

This environment can be used to compare any trace (for example obtained by prototyping or simulation) to the possible dynamics of a component-based design. For example, it can be found that a certain part within the realisation does not fulfill the relevant component's specified dynamic properties. As an example, in Figure 4 a trace is considered where agent B is expected to have a certain combination of properties, under which the information acquisition reactivity property but actually does not have this property (nor the information acquisition pro-activeness property). No conclusion is reached by the system. What is the cause of this failure? By checking all expected properties of B (and of the other components) it turns out that indeed the information acquisition reactivity property fails, whereas the other expected properties do not fail. This pinpoints the cause of the failure. The result of the check is (taken from the larger picture in the interface) as depicted in Figure 5. It shows in red which and when outputs were expected from agent B that actually lack in the trace.

For all traces, the successfulness property GR1, the correctness property GR2, the conservatism property GR3, and the cooperation necessity property GR4 as specified in simple form above can be checked automatically, and actually have been checked for a number of traces. As mentioned, to use this software environment, both a specification of a relevant property for the overall design and a set of traces of the considered design are needed. The relevant dynamic property is assumed to result from a requirements analysis during the design process.

The set of traces against which the requirement is checked, can be provided in two manners. The first manner to obtain this set of traces is by simulation of the candidate component-based design. Here it is assumed that the (reusable) components used in the component-based design all have associated dynamic properties that hold for them. Moreover, it is assumed that these dynamic properties of the components can be expressed in executable format, based on the relation  $\bullet \rightarrow\!\!\!$ . If these assumptions are fulfilled, simulation can be performed based on the simulation software environment described in Section 6. The second manner to obtain a set of traces is by prototyping. This assumes that realisations of the components used are available, and that they can easily be configured to obtain a

prototype realisation of the whole design. This prototype realisation is used to do some test sessions, resulting in a set of traces that can be checked automatically as described above.

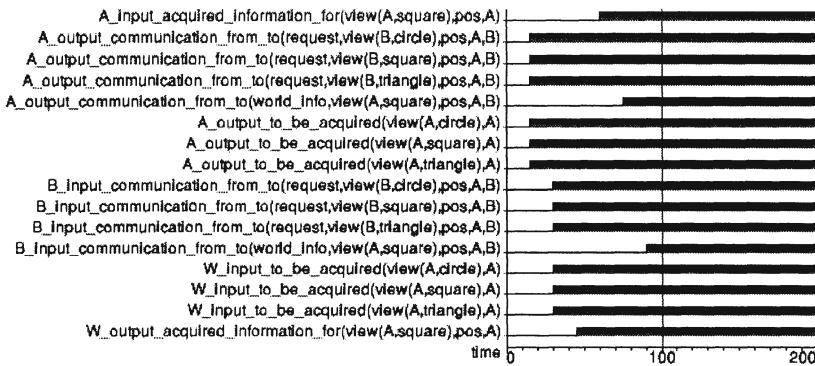


Figure 4. An unsuccessful trace

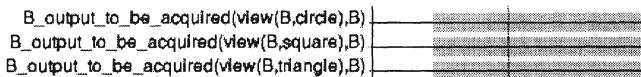


Figure 5. Pinpointing the cause of failure

## 5.2 PROVING A $\bullet \rightarrow\!\!\! \rightarrow$ PROPERTY FROM AN EXECUTABLE SPECIFICATION

Another software tool (about 300 lines of code in Prolog) addresses the proving of dynamic properties (expressed in terms of  $\rightarrow\!\!\! \rightarrow$ ,  $\bullet\!\!\! \rightarrow$  or  $\bullet \rightarrow\!\!\! \rightarrow$ ) of a component-based design as a whole from an (executable) specification of the dynamic properties of the components without involving specific traces. This dedicated prover exploits the executable nature of the specification and the ‘past/current implies future’ (cf. (Barringer et al. 1996)) nature of the property to keep complexity limited.

Using this prover, dynamic properties of the component-based design can be checked that hold for all traces, without generating them all (or a sample set of them) by subsequent simulation or prototyping as is needed in the approach described in Section 5.1. The efficiency of finding such a proof strongly depends on the complexity of the specifications of the dynamic properties for the different components. For example, given the executable specification of the software agents, the correctness property GR1, the correctness property GR2, the conservatism property GR3, and the

cooperatioon necessity property GR4 defined above can be (and actually have been) proven from suitable sets of properties of the components.

Also properties can be disproven. For example, for specifications of sets of dynamic properties of the agent components that do not guarantee information exchange between the agents, the property that ‘succesfulness of information acquisition of the two agents implies successfulness’ can be (and actually has been) proven not to hold. The prover comes up with a trace that is a counter example against this property: Similarly, for agents that are not conclusion pro-active, the property that ‘if each of the agents provides the other agent with the information it acquired implies successfulness’ has been disproven.

The efficiency of the prover is reasonable. The price that is paid to keep complexity limited is that only properties of the overall design can be proven that can be written in one of the formats  $\rightarrow$ ,  $\bullet\rightarrow$  or  $\bullet\rightarrow\rightarrow$ .

## 6. Simulation Environment

Based on the declarative executable temporal language, a simulation environment has been created to enable the simulation of a component-based design based on specification (in executable format) of dynamic properties of its component, to be used for testing and evaluation purposes; similar to, e.g., (Al-Asaad and Hayes 1995). In this section, first an example of an executable model is discussed. Then the simulation software is introduced. Finally, some of the results of the experiments with the example executable model are discussed. Input for the simulation environment is a set of executable temporal formulae expressed in terms of the leads to relation  $\bullet\rightarrow$  (i.e., in the format defined in Section 3). A software environment has been developed which implements the temporal formalisation of the dynamics as specified by an executable model. First the approach is introduced, then the program will be briefly reviewed, after which some of the results are discussed.

### 6.1 THE SIMULATION SOFTWARE

The simulation determines the consequences of the temporal relationships forwards in time. Remember that  $\alpha$  leads to  $\beta$ , is denoted by  $P1:\alpha \bullet\rightarrow_{e, f, g, h} P2:\beta$ , where the time delay  $\lambda$  is taken from the interval  $[e, f]$ . The duration parameter  $g$  denotes the time span that  $\alpha$  must minimally hold, and  $h$  denotes the duration parameter that  $\beta$  must minimally hold. In order to make simulation efficient, long intervals of results are derived when starting from long intervals. By applying additional conditions (i.e.,  $e+h \geq f$ ), the derivation of longer intervals becomes possible, see Section 3, Figure 3. The

logical relationships thus avoid unnecessary work for the simulation software.

The delay value  $\lambda$  can either be chosen randomly within the interval  $[e, f]$  each time a relationship is used in simulation, or the  $\lambda$  can be a fixed value in this interval. Selecting either a random or fixed  $\lambda$  enables thorough investigation of the consequences of a particular model.

Following the paradigm of executable temporal logic, cf. (Barringer et al. 1996), but extended to real-time intervals, a 8000 line simulation program was written in C++ to automatically generate the consequences of the leads to relationships within the executable specification of dynamic properties of the components within a component-based design. The program is a special purpose tool to derive the results forwards in time, as in executable temporal logic. After a short look at the method of forward derivation, the specification of the derivation rules is presented.

In order to derive the consequences of the temporal relationships within a specific interval of time, a cycle is performed, starting at time 0. For the set of rules the earliest starting time of the antecedent for each rule, for which the consequent does not already hold, is computed. A rule with earliest start time of the antecedent is chosen. This rule is then fired at that time, adding the consequent to the trace. The cycle is restarted, only looking for antecedents at or after the fire time point, as effects are assumed to occur simultaneously or after their causes. This continues until no more rules can be fired, or the fire time is at or after the end time of the simulation interval.

The program reads a specification of temporal rules from a plain text file. The maximum time for derivation is also specified in that file, the interval  $[0, \text{MaxTime}]$ . In order to specify facts about the environment, (periodic) intervals can be given. The functions `not()`, `and()`, and `or()` can be used to make more complex properties from atoms. The properties have `and`, `or` and `not` given in prefix ordering for the program (instead of infix), i.e., a function is given before its arguments, e.g., `and(a, b)` instead of `(a and b)`. The `+(` and `+`) brackets perform concatenation of their contents, in order to construct identifiers from variables and strings. The  $\bullet\rightarrowtail$  relation is specified using `LeadsTo`, followed by the  $e$ ,  $f$ ,  $g$  and  $h$  values. Note that the program does not use the  $\bullet\leftarrow$  (originates from) part of the relation as only forward derivation is performed. First the timing is given, then the variables are quantified. A restriction is often put on the variables. Then the antecedent and consequent are given. For clarity, tokens are displayed boldface, values and identifiers are not. For example, the information acquisition reactivity property is expressed as follows; here the keywords of the language are in bold:

```
RuleVarLeadsTo delay 5 5 10 10
Var ForAll A : AGENT
```

```

ForAll B : AGENT NotEqual B : A
ForAll O : OUTLINE
EndVar
+(A _input_ communication_from_to(request , view(A , O) , pos , B , A) +) o-->
+(A _output_to_be_acquired(view(A , O) , A) +)

```

## 6.2 SOME SIMULATION RESULTS

Figure 6 shows some of the results of simulation (picture generated by the tools) with the example component-based design. Time is on the horizontal axis. The properties are listed on the vertical axis. The  $\lambda$  is fixed at 0.5. A dark box on top of the line indicates the property is true during that time period, and a single line indicates that the property is false during that time period. The first line, for example, contains the property that A has as input that what he has acquired is a shape that is no circle. This property is false most of the time, but true from approximately time point 60 on.

Figure 6 shows that the first event is that agent A takes the initiative (around time point 15) to acquire information by itself, and to request B for information. This information is received by B and the External World component around time point 30. The External World provides the information (it is a square) around time point 45; this acquired information is received by A around time point 65. In the meantime, around time point 45 B starts to acquire its own information, which reaches the External World around time point 60. Around time point 75 the External World provides this acquired information for B (it is a circle), which is received by B around time point 90. Agent B draws the conclusion around time point 105, and also communicates its acquired information to A around this timepoint 105; A receives it around time point 120, and draws a conclusion around time point 130. Figure 7 shows another simulation trace generated by the tools.

## 7. Discussion

In this paper the use of specified properties of the dynamics of a component-based design in the context of evaluation of the design is addressed. A declarative temporal trace language TTL is offered to specify dynamic properties, and a declarative executable language is defined as a basis for simulation and more efficient analysis. This executable language belongs to the paradigm of executable temporal languages, cf. (Barringer et al. 1996), but adds the use of real-time intervals. Models can be specified in a declarative manner based on a temporal ‘leads to’ relation which is parameterized by four real valued parameters for time durations and delays. Within the simulation environment such models can be executed.

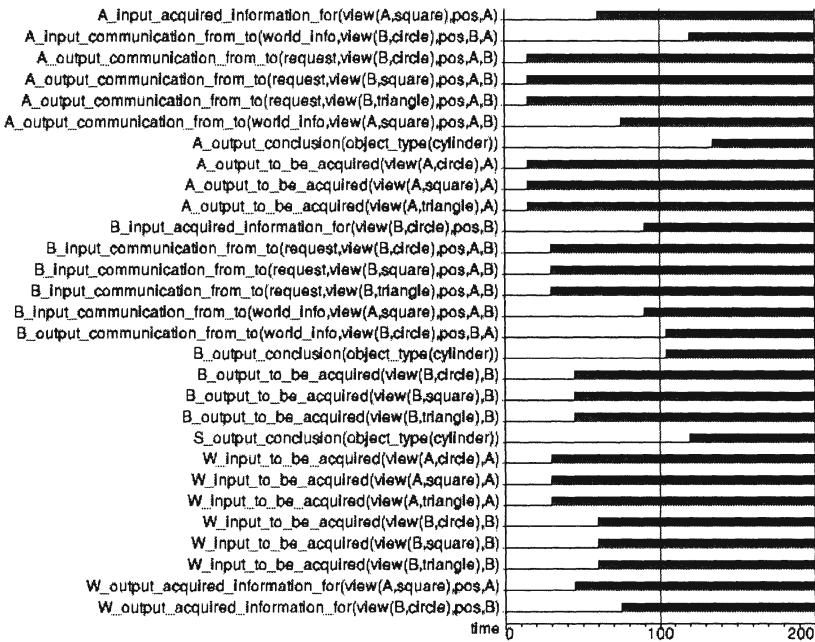


Figure 6. A is fully reactive and proactive; B is reactive, but proactive in making conclusions.

To specify dynamic properties in a more expressive manner, the language TTL is used: a Temporal Trace Language that belongs to the family of languages to which also situation calculus (McCarthy and Hayes 1969; Reiter 2001), event calculus (Kowalski and Sergot 1986), and fluent calculus (Hölldobler and Thielscher 1990) belong. The executable language for simulations is definable within this much more expressive language. Supporting tools have been implemented that consist of:

- A software environment for simulation of a component-based design on the basis of a specification of dynamic properties of the components and their connections in executable ‘leads to’ format
- A software environment for evaluation of dynamic properties against traces for a component-based design, both for properties in TTL format and in ‘leads to’ format
- A software environment to automatically prove (or disprove) ‘leads to’ properties of the overall design from ‘leads to’ properties of its components and their connections

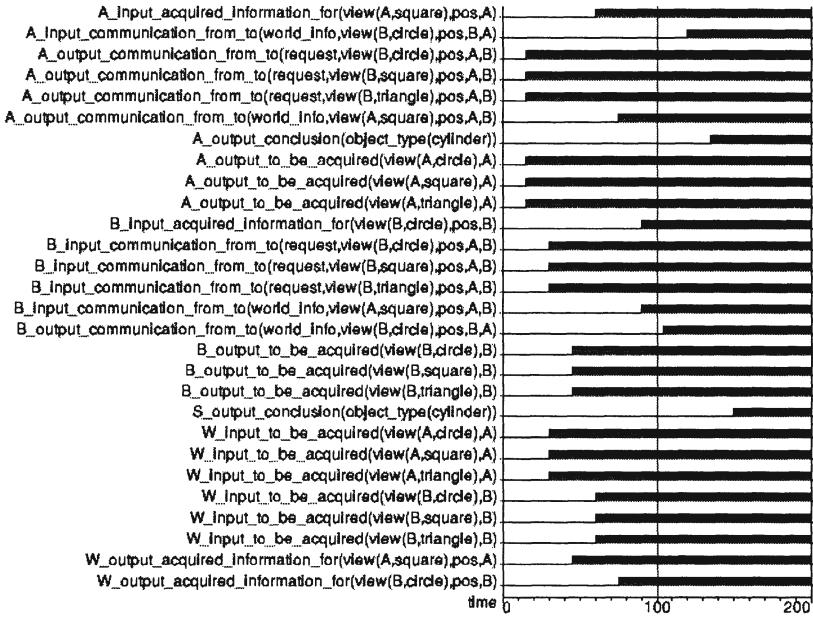


Figure 7. A is fully proactive and reactive, B is reactive only.

The temporal trace language TTL used in our approach is much more expressive than standard or extended modal temporal logics as described, for example, in (Henzinger et al. 1994; Bouajjani et al. 1996; Yovine 1997; Fisher 1994; Clarke et al. 2000; Manna and Pnueli 1995; Stirling 2001), in a number of respects. In the first place, it has *order-sorted predicate logic* expressivity, whereas most standard temporal logics are propositional. Secondly, the explicit reference to *time points and time durations* offers the possibility of modelling the dynamics of real-time phenomena. These first two points apply only partially to logics where it is possible to have real numbers for time and arithmetical operations and order relations to express constraints between time points, as in (Henzinger et al. 1994; Bouajjani et al. 1996; Yovine 1997).

Third, the possibility to quantify over traces allows for specification of *more complex dynamics*. As within most temporal logics, reactivity and pro-activeness properties can be specified. In addition, in our language also properties expressing different types of adaptive behaviour can be expressed. For example an adaptive property such as ‘exercise improves skill’, or ‘the better the experiences, the higher the trust’ (trust monotonicity) which both are a relative property in the sense that it involves the comparison of two alternatives for the history. This type of property can

be expressed in our language, whereas in standard forms of temporal logic different alternative histories cannot be compared. The same difference applies to situation calculus, event calculus, fluent calculus, and the languages in (Henzinger, et al. 1994; Bouajjani, et al. 1996; Yovine 1997).

Fourth, in TTL it is possible to define *local languages for parts* of a system. For example, the distinctions between components, and between input and output languages are crucial, and are supported by the language, which also entails the possibility to quantify over system parts and changing system parts over time; for example, this allows for specification of system configuration modification over time; cf. (Dastani, et al. 2002)

The approach proposed suggests that (documentation) libraries of reusable components should as much as possible include specifications of dynamic properties in the simpler ‘leads to’ format. If these properties can be taken from such a library, and requirements on the dynamics of the design as a whole are formally specified as indicated, then the support as described can work quite well. If, however, no specifications of the dynamic properties of the reusable components are known, then as part of the design process these properties and their specifications have to be identified first.

## References

- Al-Asaad, H and Hayes, JP: 1995, Design verification via simulation and automatic test pattern generation, *International Conference on Computer-Aided Design*, IEEE Society Press, Los Alamitos, CA, pp. 174-180.
- Barringer, H, Fisher, M, Gabbay, D, Owens, R and Reynolds, M: 1996, *The Imperative Future: Principles of Executable Temporal Logic*, Research Studies Press Ltd. and John Wiley and Sons.
- Bouajjani, A, Lakhnech, Y and Yovine, S: 1996, Model checking for extended timed temporal logic, *Proceedings of the 4th International Symposium Formal Techniques in Real-Time and Fault-Tolerant Systems, FTRTFT'96*, Springer-Verlag, Berlin, pp. 306-326.
- Brazier, FMT, Jonker, CM and Treur, J: 1998, Principles of compositional multi-agent system development, in J Cuena (ed.), *Proceedings of the 15th IFIP World Computer Congress, WCC'98, Conference on Information Technology and Knowledge Systems, IT&KNOWS'98*, pp. 347-360. To be published by IOS Press, 2002.
- Clarke, EM, Grumberg, O and Peled, DA: 2000, *Model Checking*, MIT Press, Cambridge, MA.
- Dastani, M, Jonker, CM and Treur, J: To appear , A requirement specification language for configuration dynamics of multi-agent systems, in M Wooldridge, G Weiss and P Ciancarini, (eds), *Proceedings of the 2nd International Workshop on Agent-Oriented Software Engineering, AOSE'01*. Lecture Notes in Computer Science, vol. 2222. Springer Verlag.
- Fisher, M: 1994, A survey of concurrent METATEM — the language and its applications, in DM Gabbay and HJ Ohlbach (eds), *Temporal Logic — Proceedings of the First International Conference*, Lecture Notes in AI, vol. 827, pp. 480-505.

- Henzinger, T, Nicollin, X, Sifakis, J and Yovine, S: 1994, Symbolic model checking for real-time systems, *Information and Computation* 111(2): 193—244.
- Herlea, DE, Jonker, CM, Treur, J and Wijngaards, NJE: 1999, Specification of behavioural requirements within compositional multi-agent system design, in FJ Garijo and M Boman (eds), *Multi-Agent System Engineering, Proceedings of the 9th European Workshop on Modelling Autonomous Agents in a Multi-Agent World, MAAMAW'99*. Lecture Notes in AI, vol. 1647, Springer Verlag, Berlin, pp. 8-27.
- Hölldobler, S and Thielscher, M: 1990, A new deductive approach to planning, *New Generation Computing* 8: 225-244.
- Jonker, CM and Treur, J: 1998, Compositional verification of multi-agent systems: a formal analysis of pro-activeness and reactivity, in W.P. de Roever, H. Langmaack, A. Pnueli (eds), *Proceedings of the International Workshop on Compositionality, COMPOS'97*. Lecture Notes in Computer Science, vol. 1536, Springer Verlag, Berlin, pp. 350-380, Extended version in *International Journal of Cooperative Information Systems*. In press, 2002.
- Kowalski, R and Sergot, M: 1986, A logic-based calculus of events, *New Generation Computing* 4: 67-95.
- Manna, Z and Pnueli, A: 1995, *Temporal Verification of Reactive Systems: Safety*. Springer Verlag, Berlin.
- McCarthy, J and Hayes, P: 1969, Some philosophical problems from the standpoint of artificial intelligence, *Machine Intelligence* 4: 463--502.
- Reiter, R: 2001, *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*, MIT Press, Cambridge, MA.
- Roever, WP de, Langmaack, H and Pnueli, A, (eds): 1998, *Proceedings of the International Workshop on Compositionality, COMPOS'97*. Lecture Notes in Computer Science, vol. 1536, Springer Verlag, Berlin.
- Stirling, C: 2001, *Modal and Temporal Properties of Processes*, Springer Verlag, Berlin.
- Yovine, S: 1997, Kronos: A verification tool for real-time systems, *International Journal of Software Tools for Technology Transfer* 1: 123-133.

## IDENTIFYING COMPONENT MODULES

ROBERT I WHITFIELD, JOANNE S SMITH AND ALEX B  
DUFFY

*University of Strathclyde  
UK*

**Abstract.** A computer-based system for modelling component dependencies and identifying component modules is presented. A variation of the Dependency Structure Matrix (DSM) representation was used to model component dependencies. The system utilises a two-stage approach towards facilitating the identification of a hierarchical modular structure. The first stage calculates a value for a clustering criterion that may be used to group component dependencies together. A Genetic Algorithm is described to optimise the order of the components within the DSM with the focus of minimising the value of the clustering criterion to identify the most significant component groupings (modules) within the product structure. The second stage utilises a 'Module Strength Indicator' (MSI) function to determine a value representative of the degree of modularity of the component groupings. The application of this function to the DSM produces a 'Module Structure Matrix' (MSM) depicting the relative modularity of available component groupings within it. The approach enabled the identification of hierarchical modularity in the product structure without the requirement for any additional domain specific knowledge within the system. The system supports design by providing mechanisms to explicitly represent and utilise component and dependency knowledge to facilitate the non-trivial task of determining near-optimal component modules and representing product modularity.

### 1. Introduction

Various product-structuring principles such as product architectures (Erens et al. 1996) and product platforms (Elgard et al. 1998) have been prescribed to better support the documentation, rationalisation and re-use of components in product structures. Researchers have increasingly sought to improve the means by which the identification of such structures is determined (Otto 2001; Gonzalez-Zugasti et al. 2000; Jiao et al. 1999;

Zamirowski et al. 1999). As such, the principle of modular design has gained increasing prominence as a potential means to facilitate improved product architectures/platform design and re-use support (Smith et al. 2001a; Smith et al. 2001b).

Modular design involves the creation of product variants based on the configuration of a defined set of modules. Modules are commonly described as groups of 'functionally' or 'structurally' independent components (Sosale et al. 1997). The principal aim is to create variety, reduce complexity and maximise kinship in designs and across product families. The major benefits of a modular design include: efficient upgrades; reduced complexity; reduced costs; rapid product development, and, improved design knowledge structuring (Miller 1999; Muffato 1999; O'Grady et al. 1998). Despite the existing evidence regarding its benefits and the increased understanding of the capabilities of modularity as a design tool, 'little work has been done on these research issues' (O'Grady et al. 1998) and 'modularity has been treated in the literature in an abstract form' (Huang et al. 1998). That is, there is little aid in the form of tools, techniques and methodologies for practicing designers, and consequently, a need exists for approaches 'to determine modules, represent modularity, and optimise modular design' (Huang et al. 1998).

A number of algorithms exist that may be applied to the optimisation of modular design problems including simulated annealing (Kirkpatrick et al. 1983), genetic algorithms (GA) (Holland 1962) and Tabu search (Glover 1993). The optimisation algorithms tend to have a number of parameters that affect their performance and are intrinsically linked to the problem domain, for example, the annealing schedule for simulated annealing.

This paper demonstrates the application of a GA to component structure optimisation where the objective was to determine the optimum modular configuration for the design artefact. Modularisation research is discussed within Section 2 in order to identify the requirements for a modular identification system. The Dependency Structure Matrix (DSM) – Steward (1981) was used as the dependency modelling technique due to its generic applicability, ease of representation within a computer-based system, and, its quantifiable nature. The DSM modelling technique and system are described within Section 3. A multi-criteria GA was developed and adapted for application to this particular type of problem and is described within Section 4. Section 5 defines the '*Module Strength Indicator*' (*MSI*) function that is used to facilitate module identification and production of a '*Module Structure Matrix*' (*MSM*). Two different design artefacts were used to test the performance and effectiveness of both the modularisation approach and of the optimisation algorithm. These artefacts, and the results of the

optimisation are described within Section 6. Finally, conclusions are made within Section 7.

## 2. Modularisation Research

The main characteristics that determine modularity have been defined as the degree of interaction/dependency between components both: within a module, and, of different modules (Ulrich et al. 1991; Kusiak et al. 1996).

The criteria for the optimal modular product structure can thus, be defined as the clustering of components such that the degree of interaction/dependency is:

- Maximised internally within groups (modules).
- Minimised externally between groups (modules).

The challenge for modular design research is to identify this optimal modular structure. Firstly, there is the requirement for the adequate modelling of component and interaction knowledge to support its analysis. Secondly, the resulting model must be optimised with respect to the above criteria. Finally, modules must be identified based on the results of the optimisation process.

A number of researchers have addressed the problem of modularising product designs. The modelling requirements are generally fulfilled using either interaction graphs (Kusiak et al. 1996) or matrix techniques (Blackenfelt 2001; Jarventausta et al. 2001; Salhieh et al. 1999; Sosale et al. 1997). Optimisation criteria and techniques vary between applications. For example, Salhieh and Kamrani (1999) utilised a P-median model to optimise the sum of component similarities, Jarventausta and Pulkkinen (2001) the Cluthill-McGee algorithm and Sosale et al. (1997) used a simulated annealing algorithm. Alternatively Blackenfelt (2001) based component groupings on their ability to fulfil the criteria of a series of strategic module drivers such as 'styling' or 'the make or buy decision'. Despite the availability of varying methods to both model and optimise component and interaction knowledge, there are disparities with respect to the methods for the identification of modules within these optimised models. For example, in Jarventausta and Pulkkinen (2001), and Salhieh and Kamrani (1999) modules are identified manually by perusing an optimised matrix. In Sosale et al. (1997) the results of the optimisation are presented in a list of components each with a value indicating the module number to which the component is assigned. However, it is not clear from their published works (Sosale et al. 1997; Gu et al. 1997) how these modules are extracted from the optimised matrix. The rationale given for their module groupings would suggest that it is subject to manual interpretation of the matrix.

Due to the complex nature of the module identification problem it may not always be appropriate, or possible, to manually identify modular component groupings within graphs or matrices, as in Jarventausta and Pulkkinen (2001). This is especially pertinent as the design space becomes more highly constrained i.e. when there are a large number of inter-dependencies between components. In such cases, the clustered matrix/graph may be densely populated and on first perusal yield no significant modules. Therefore, an alternative analysis of the optimised model would be required to facilitate module identification. Such analysis would conceivably require the utilisation of further resources such as domain specific knowledge. This would result in additional time and effort on the part of the human designer and/or the development of computational support systems. Further, when modules are readily identifiable, whether manually or automatically, it may not always be appropriate to return a list of definitive modules as suggested by Sosale et al. (1997). For example, what is modular from one perspective (e.g. assembly) may not be modular from another (e.g. maintenance) (Miller 1998). Thus, we see that the modular design problem is further complicated by the fact that differing modular configurations support different perspectives of the problem. In addition, the modular design may also exist over different hierarchical levels of the product structure. Thus, the inherent hierarchical modularity is not exposed when the outcome of the identification phase is presented as a list (Sosale et al. 1997) or as definitive module boundaries (Salhieh et al. 1999).

Given the above, the following must be determined in order to develop a means to facilitate improved module identification:

- Inherent modularity.
- Potentially differing modular configurations.
- Differing hierarchical levels of modularity in the product structure.

Our aim is to facilitate module identification with respect to the above modular requirements. The means of doing so is to combine the strengths of both the designer, in terms of their domain and problem specific knowledge, and of computer support, in terms of its advanced capabilities for rapid and precise analysis and calculation.

### 3. Dependency Structure Matrix

The Dependency Structure Matrix (DSM), also known as the Design Structure Matrix, has been extensively used to represent concepts such as: tasks, resources and parameters, as well as the inter-concept dependencies. The DSM is generic in nature, but due to its compactness, easily quantifiable nature, and ability to represent most design activity

relationships, has seen considerable use in the analysis and management of the product development process (Coates et al. 2000; Eppinger et al. 1994; Kusiak et al. 1990; Steward 1981). More recently however, the DSM has been applied to model various design product concepts and their dependencies, from functions (Jarventausta et al. 2001) to parts (Salhieh et al. 1999).

The DSM consists of a list of concepts (activities, functions, parts) that are represented in the same order in both the row and column of the matrix. The matrix part represents the dependencies between the concepts. Steward (1981) originally represented the dependencies in a binary form: 0 to indicate no dependency, and, 1 to indicate a dependency, however, the modelling technique has evolved to reflect a measure of the degree of dependency, termed its weight.

A DSM modelling and analysis system was constructed with the focus of providing mechanisms to enable the optimisation of the order of components with respect to a pre-determined clustering criterion, Figure 1.

The system allows the creation of a matrix containing any number of components with the matrix changing size automatically as components are added or removed. The dependencies between components are currently limited to representing a single type, e.g. a geometrical perspective, however, work is underway to enable the representation of multi-dimensional dependencies. Selecting a cell within the matrix will change the state of the dependency from either independent or dependent. The user may also change the weight of the dependency, which is reflected by its colour.

The order of the components within the matrix may be managed manually by dragging either of the rows or columns into a new position. The value for the clustering criterion is simultaneously re-calculated, assisting the user in the determination of an improved modular structure. Alternatively, the product structure may be optimised using one of the optimisation algorithms available within the optimisation module. The system can simultaneously manage the optimisation of multiple design artefacts although this will obviously take longer on a computer with a single processor.

#### 4. Modular Structure Optimisation

A number of optimisation algorithms such as hill-climbing and simulated annealing were developed and tested within the optimisation of the order of components within the DSM (Whitfield 2001b). The difficulty in optimising the DSM lies in the number of combinations of possible component orders. For example, a matrix containing 30 components has  $6.652 \times 10^{32}$  possible

combinations. An exhaustive search for this type of problem is clearly inappropriate. There is also a requirement for the optimisation technique to be able to deal with discrete, multi-modal, noisy and multi-criteria solution spaces that are common within the DSM (Todd 1997).

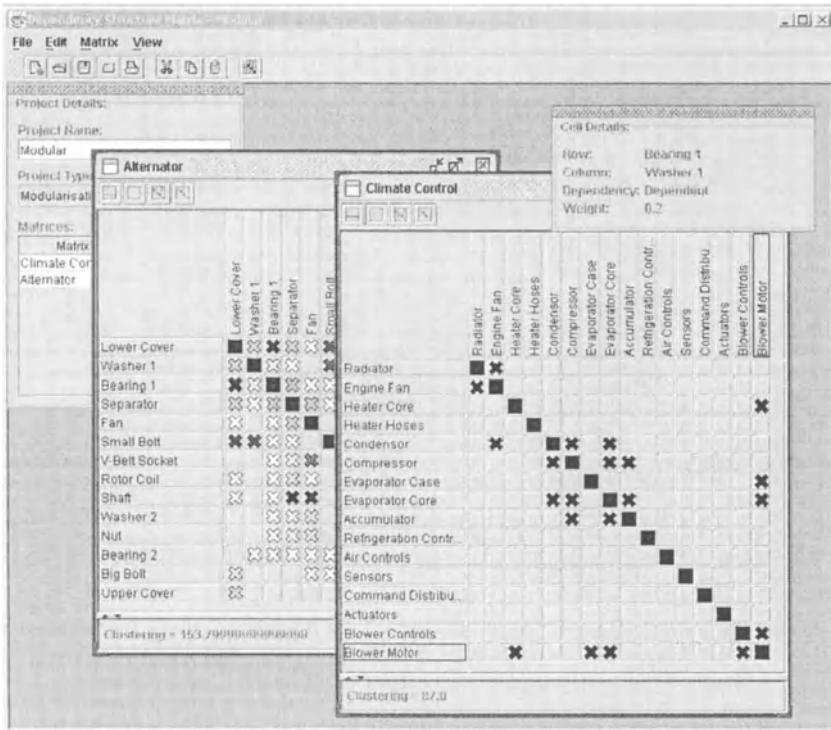


Figure 1. Dependency Structure Matrix system

The general procedure for Genetic Algorithms developed by Goldberg (1989a), has been used to enable the evolution of optimal modular structures. The objective of the GA in this particular application is to minimise the value for the clustering criterion, (equation 1). Additional clustering criteria will be included to investigate other modular performance characteristics. The algorithm is illustrated within Figure 2.

The system's GA is generic in nature using object-oriented techniques and allows the encoding of a sequence of any type of concept. Within this application, the chromosome is initially encoded as a random order of components. This randomisation attempts to ensure that the chromosome represents a unique point in the solution space, such that the group of

chromosomes are randomly distributed throughout, Figure 3. In the case of the DSM problem, the group of chromosomes, or population, represent orders of components that generally have a poor initial clustering criterion. The chromosomes are then evaluated within the DSM with respect to the criteria.

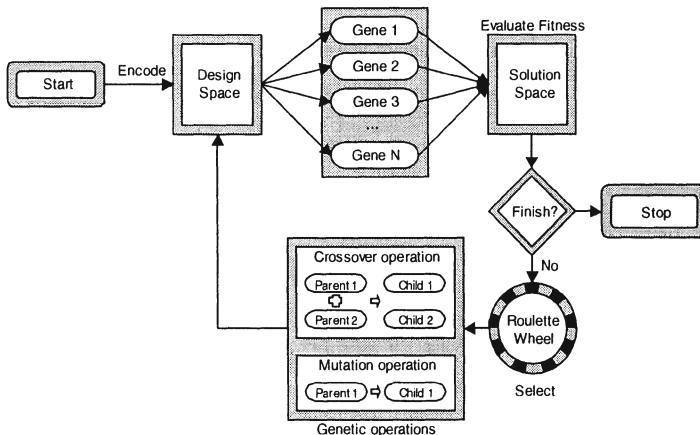


Figure 2. A general structure for genetic algorithms

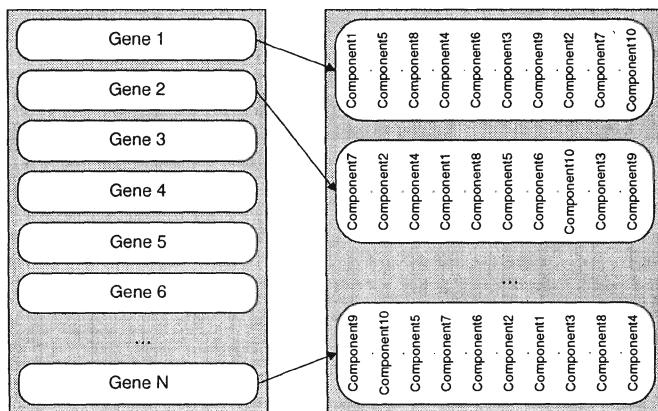


Figure 3. Genetic representation of component order

A Roulette-wheel type selection procedure is used where each chromosome is given a portion of the wheel that is proportional to its

performance (Goldberg 1989a). Chromosomes with higher performance characteristics therefore have a greater chance of surviving, although it is possible for lower performance chromosomes to be passed through to the next generation.

Crossover and mutation operations are then performed upon the selected chromosomes to produce the next generation. Two parent chromosomes are selected at random and removed from the population. The two parents are then crossed based upon a probability of crossover to produce two children containing genetic information from both parents. Mutation works in a similar manner on a single chromosome to produce a small change in the parent. The crossover and mutation operations encoded within the GA are listed within Tables 1 and 2.

TABLE 1. Crossover operators encoded within GA

| Initial | Description                           | Reference                   |
|---------|---------------------------------------|-----------------------------|
| 1PX     | One Point Crossover                   | Murata & Ishibuchi (1994)   |
| 2PEX    | Two Point End Crossover               | Murata & Ishibuchi (1994)   |
| 2PCX    | Two Point Centre Crossover            | Murata & Ishibuchi (1994)   |
| 2PECX   | Two Point End/Centre Crossover        | Murata & Ishibuchi (1994)   |
| PBX     | Position Based Crossover              | Syswerda (1991)             |
| IPX     | Independent Position Crossover        | Murata & Ishibuchi (1994)   |
| PMX     | Partially Mapped Crossover            | Goldberg & Lingle (1985)    |
| OX      | Ordered Crossover                     | Davis (1985)                |
| CX      | Cycle Crossover                       | Oliver et al. (1987)        |
| ERX     | Edge Recombination Crossover          | Whitley et al. (1989)       |
| EERX    | Enhanced Edge Recombination Crossover | Starkweather et al. (1991)  |
| SCX     | Subtour Chunks Crossover              | Greffenstette et al. (1985) |
| AEX     | Alternating Edges Crossover           | Greffenstette et al. (1985) |
| IX      | Inversion Crossover                   | Goldberg (1989b)            |

The new population is then re-evaluated with respect to the performance criteria. A check is made to determine whether the GA has completed a pre-determined number of generations, finishing if it has, otherwise repeating this evaluation, selection, crossover and mutation processes.

TABLE 2. Mutation operators encoded within GA

| Initial | Description                   | Reference                 |
|---------|-------------------------------|---------------------------|
| 2ORS    | Two Operation Random Swap     | Murata & Ishibuchi (1994) |
| 2OAS    | Two Operation Adjacent Swap   | Murata & Ishibuchi (1994) |
| 3ORS    | Three Operation Random Swap   | Murata & Ishibuchi (1994) |
| 3OAS    | Three Operation Adjacent Swap | Murata & Ishibuchi (1994) |
| SOM     | Shift Operation Mutation      | Murata & Ishibuchi (1994) |

The parameters for the genetic algorithm and the optimisation criteria may be selected using the optimiser dialog shown within Figure 4. The population size, number of generations, crossover probability and mutation probability may be entered within the text fields. An indicator displays the genetic algorithm's progress through the evaluation of the populations.

A previous investigation attempted to determine the parameters for generic combinatorial optimisation using GAs (Todd 1997), however research has demonstrated that the parameters for the GA are intrinsically tied to the domain (Whitfield et al. 2001a).

Checking each combination of crossover and mutation operators and probabilities identified suitable GA parameters for this particular domain. The results indicated that the two point end crossover and two operation adjacent swap mutation operators with 80% and 20% probabilities respectively was the most successful set-up for the GA. The population size and generation count were both set at 100.



Figure 4. Optimiser dialog

Criteria may be selected for the basis of optimisation using the Criteria Set-up area of the optimisation dialog. The optimisation may be either single criterion or multi-criteria where the individual objectives are minimisation, maximisation, target value, or any combination.

After completion of the optimisation, a list of optimal component orders is displayed within the solution table. The list displays the component order, the values for the criteria selected, the fitness and the rank for each of the solutions. For multi-criteria problems, the genetic algorithm will produce a

number of solutions that represent the trade-off between the criteria. Selecting one of the optimum solutions within the solution table will display the component ordering within the DSM.

The clustering criterion is represented as equation (1). The DSM model may however have any number of criteria included to increase the diversity of the optimisation. Equation 1 represents the summation of the dependencies both above and below the leading-diagonal multiplied by their distance from the leading-diagonal on the basis of their weight. The focus of minimising the clustering criterion is therefore to get the dependencies as close to the leading diagonal as possible grouping component dependencies together. Priority is automatically given towards higher weighted dependencies.

$$\text{Clustering Criterion} = \sum_{i=1}^N \sum_{j=1}^N |(j-i)| \times w_{i,j} \quad (1)$$

Where:  $N$  is the number of components in the DSM,  
 $i$  and  $j$  are the row and column indices, and  
 $w_{i,j}$  are the dependency weights.

The criteria values are automatically re-calculated and updated when new components are added to the matrix, or when component dependencies are changed, and are displayed within the criteria area.

## 5. Module Identification

A function was derived to assign a '*Module Strength Indicator*' (*MSI*) to all potential modules within the design artefact. The MSI function consists of two parts, equations 2 and 3 below. Equation 2 provides the designer with the mean value of the internal dependencies within the module determined by the indices and represents the strength of the internal dependencies of a component grouping. Equation 3 determines a mean value for the external dependencies within the module and represents the strength of the external dependencies of the component grouping. The focus of the MSI function is towards identifying modules that have a maximum number of internal dependencies and a minimum number of external dependencies, Figure 5.

Subtracting equation 3 from equation 2 can derive the relative modularity of a clustered artefact, with respect to its components' internal and external dependencies. The MSI function, equation 4, therefore provides a modularity metric directly related to the overall modularity characteristics of the design artefact.

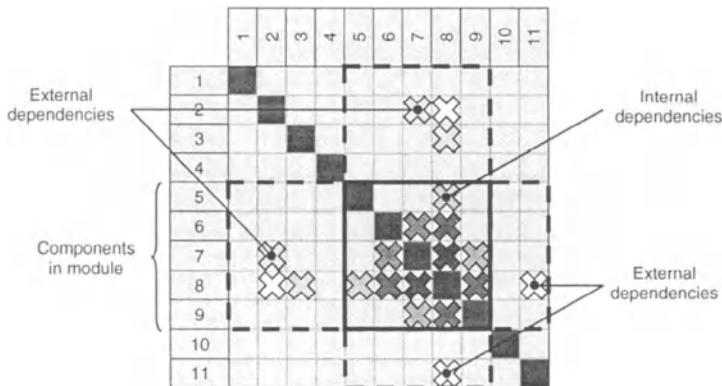


Figure 5. Internal and external dependencies of module

$$MSI_i = \frac{\sum_{i=n_1}^{n_2} \sum_{j=n_1}^{n_2} w_{i,j}}{(n_2 - n_1)^2 - (n_2 - n_1)} \quad (2)$$

$$MSI_e = \frac{\sum_{i=0}^{n_1} \sum_{j=n_1}^{n_2} w_{i,j} + w_{j,i}}{2 \times (n_1 \times (n_2 - n_1))} + \frac{\sum_{i=n_2}^N \sum_{j=n_1}^{n_2} w_{i,j} + w_{j,i}}{2 \times ((N - n_2) \times (n_2 - n_1))} \quad (3)$$

$$MSI = MSI_i - MSI_e \quad (4)$$

Where;  
 $n_1$  = index of first component in module, and  
 $n_2$  = index of last component in module.

Given a maximum dependency weight within the DSM of 1.0, the maximum MSI value that can be returned from equation 4 is 1.0. This represents the strongest possible module solution, i.e. a module consisting entirely of maximum weight internal dependencies (1.0 from equation 2), with no related external dependencies (0.0 from equation 3). The system provides the functionality to allow the designer to specify module identification criteria. The designer can choose to: exclude external dependencies (use only equation 2); normalise the MSI results, and, assign

colours to incremental value ranges of the MSI. The system then traverses through the DSM calculating an MSI value for all possible modules.

The MSI technique results in an alternative representation of the DSM. The resulting MSM depicts, as different coloured cells, the relative modularity of all available component modules within it. Thus, the MSM exposes the boundaries of any existing modular structure based on the given dependencies. The MSM identifies inherent hierarchical modularity within highly constrained problems with densely populated dependency matrices that otherwise may not have been readily apparent. Further, as the coloured groupings represent different strengths of modularity the MSM reveals the inherent hierarchical modularity within the structure.

The MSM provides an indication of the existing modularity within the design artefact. The designer is free to adapt the module configuration within the boundaries of the inherent modularity identified. Designers can define an appropriate modular structure from the resulting MSM based on their specific design requirements and domain knowledge.

## 6. Case Studies

The module identification approach is demonstrated within two case studies. The studies, taken from published literature represent: a climate control system (Pimmler et al. 1994), and, an alternator (Sosale et al. 1997). The climate control example depicts the components and their dependencies from a material perspective. The alternator case depicts dependencies based on the re-use and recycling life-phase of the artefact's components.

### 6.1 CLIMATE CONTROL SYSTEM

Figure 6 presents a DSM for a climate control system. The system's components are represented by the same arbitrary structure in both the column and rows. The crosses in the central part of the matrix represent the dependencies between the components, which in this case, are all equally weighted. The current value for the clustering criterion is shown in the left-hand bottom corner of the box. The value for the clustering criterion for this structure is 87.0.

Figure 7 shows an optimised structure for the climate control DSM. We can see that the clustering criterion has been reduced by approximately 65% (from 87 to 31). It is also apparent that the dependencies within the matrix are now closer together and are clustered into a number of groupings around the diagonal, indicating potential modules.

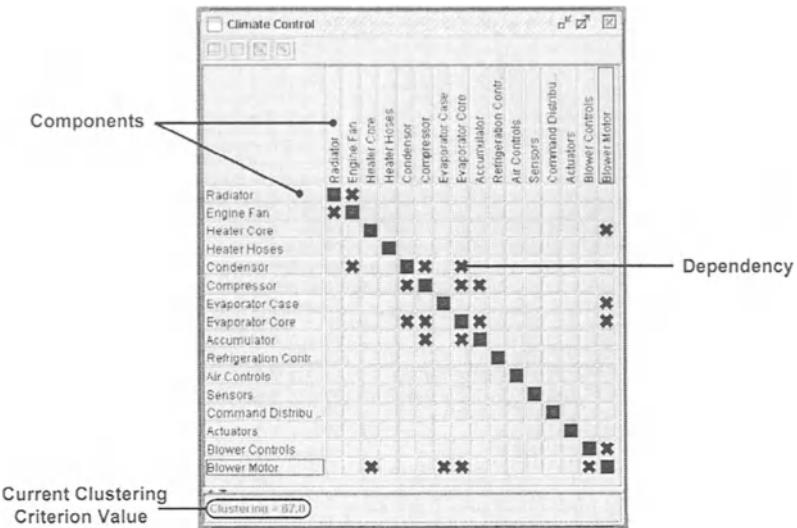


Figure 6. Climate control system DSM



Figure 7. Optimised climate control system DSM

However, the potential modular configurations and the applicable hierarchical modularity within the given solution are not apparent from the optimised DSM. The MSI function was applied to highlight modular configurations and hierarchical modularity. The resulting MSM is presented in Figure 8. This matrix was obtained through the application of equation 4 to the DSM within Figure 7 with normalised results. The matrix provides a mechanism from which to identify potential modules of varying modular strength represented by differing colours. The results indicate that there are a number of modular configurations and that a hierarchical modularity exists within the structure.

Table 3a catalogues all available modules within the structure at the differing module strengths highlighted by the MSM. An example of a hierarchical modular configuration based on the available module catalogue is given in Table 3b.

## 6.2 ALTERNATOR

The DSM for an Alternator is presented in Figure 9. Again, the components are represented in an arbitrary structure and the crosses represent the dependencies between the components.

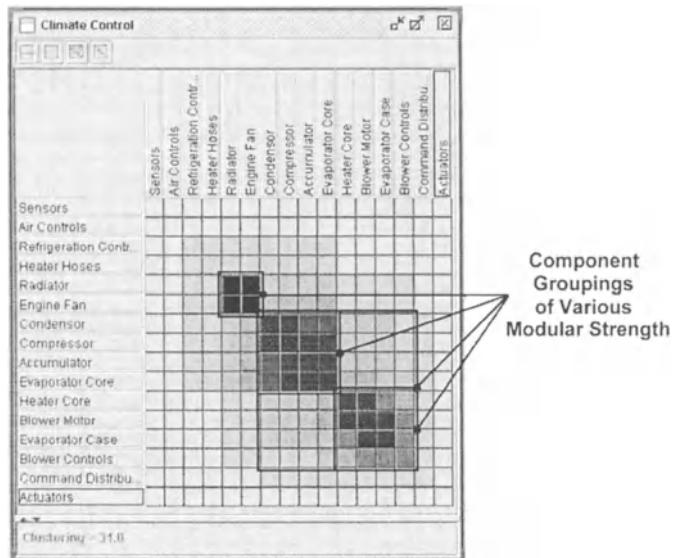


Figure 8. Climate control system MSM

TABLE 3a. Module catalogue for climate control system

| Modular Strength | Module          | Components                                                                                  |
|------------------|-----------------|---------------------------------------------------------------------------------------------|
| 1.0              | M <sub>1</sub>  | Radiator, Engine Fan                                                                        |
| 0.9              | M <sub>2</sub>  | Condenser, Compressor                                                                       |
|                  | M <sub>3</sub>  | Compressor, Accumulator, Evaporator Core                                                    |
|                  | M <sub>4</sub>  | Heater Core, Blower Motor                                                                   |
|                  | M <sub>5</sub>  | Blower Motor, Evaporator Case                                                               |
| 0.8              | M <sub>6</sub>  | Components within M <sub>2</sub> and M <sub>3</sub>                                         |
| 0.7              | M <sub>7</sub>  | Components within M <sub>4</sub> and M <sub>5</sub>                                         |
|                  | M <sub>8</sub>  | Components within M <sub>5</sub> and Blower Controls                                        |
| 0.6              | M <sub>9</sub>  | Components within M <sub>8</sub> and Heater Core                                            |
|                  | M <sub>10</sub> | Components within M <sub>4</sub> and Evaporator Core                                        |
|                  | M <sub>11</sub> | Components within M <sub>6</sub> and Engine Fan                                             |
| 0.5              | M <sub>12</sub> | Components within M <sub>11</sub> and Radiator                                              |
|                  | M <sub>13</sub> | Components within M <sub>6</sub> and M <sub>7</sub>                                         |
|                  | M <sub>14</sub> | Components within M <sub>7</sub> , Evaporator Core and Blower Controls                      |
| 0.4              | M <sub>15</sub> | Components within M <sub>1</sub> , M <sub>6</sub> , Refrigeration Controls and Heater Hoses |
|                  | M <sub>16</sub> | Components within M <sub>1</sub> and M <sub>13</sub> and Blower Controls                    |
|                  | M <sub>17</sub> | Components within M <sub>13</sub> , Blower Controls and Command Distributor                 |
| 0.2              | M <sub>18</sub> | All                                                                                         |

TABLE 3b. Hierarchical module configuration example

| Configuration | Module          | Components                                                                                                       |
|---------------|-----------------|------------------------------------------------------------------------------------------------------------------|
| Level i       | M <sub>1</sub>  | Engine Fan, Radiator                                                                                             |
|               | M <sub>6</sub>  | Condenser, Compressor, Accumulator, Evaporator Core.                                                             |
|               | M <sub>7</sub>  | Heater Core, Evaporator Case, Blower Motor                                                                       |
| Level ii      | M <sub>13</sub> | M <sub>6</sub> and M <sub>7</sub>                                                                                |
| Level iii     | M <sub>16</sub> | M <sub>1</sub> , M <sub>13</sub> and Blower Controls                                                             |
| Level iv      | M <sub>18</sub> | M <sub>16</sub> , Sensors, Air Controls, Refrigeration Controls, Heater Hoses, Command Distributor and Actuators |

The weightings were calculated as the sum of a number of different life-phase aspects such as: maintenance, upgrading, and reconfiguration. The alternator case therefore represents a more highly constrained problem than that of the climate control system, as indicated by the densely populated matrix. The value of the clustering criterion for this structure is 153.8.

Figure 10 represents the optimised DSM for the alternator problem with a clustering criterion value of 117.8, corresponding to a reduction of approximately 23%. Without further analysis, it is difficult to clearly

identify any significant component groupings within the optimised DSM. The MSI function is applied and the resulting MSM is shown in Figure 11.



Figure 9. Alternator DSM



Figure 10. Optimised alternator DSM

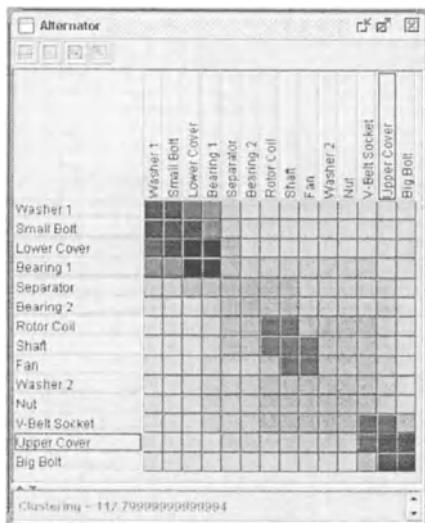


Figure 11. Alternator MSM

Again the results are normalised and are based on the application of equation 4. It is apparent from Figure 11 that there are a number of strong module groupings and hierarchical levels within the artefact.

Table 4 catalogues all available modules within the structure at the differing module strengths highlighted by the MSM. Such modularity is not immediately evident from the dependencies displayed within the optimised DSM, Figure 10.

As with the Climate Control case the designer can utilise this module catalogue and the MSM to identify a number of differing modular configurations within the product structure.

The cases presented cover the ‘structural’ viewpoint of design i.e. parts and components. However, the design activity involves the evolution of more abstract knowledge, from viewpoints such as function and working principle (means), into this physical structure. Andreasen’s Domain Theory (Andreasen 1995; 1996), Erens’ Product Model (Erens 1995) and Zhang’s Multi-Viewpoint Evolutionary Approach (Zhang 1998) all provide examples of the differing ‘views’ that can be taken of a design activity and how ‘mapping’ between these supports the evolution of the design. Therefore, a structuring principle that supports the activity of design is required to define structures within each viewpoint and depict the interactions between these viewpoints. The approach and system are being developed to support new design. As such the system can be applied to design viewpoints other than

structural, i.e. functional, and working principles (means). Further 'between viewpoint' mapping mechanisms are currently under investigation to allow the principles of modularity to be applied throughout the evolution of a new design.

TABLE 4. Module catalogue for alternator

| <b>Modular Strength</b> | <b>Module</b>   | <b>Components</b>                                                                   |
|-------------------------|-----------------|-------------------------------------------------------------------------------------|
| 1.0                     | M <sub>1</sub>  | Lower Cover, Bearing 1                                                              |
| 0.9                     | M <sub>2</sub>  | Upper Cover, Big Bolt                                                               |
|                         | M <sub>3</sub>  | Small Bolt, Washer 1                                                                |
|                         | M <sub>4</sub>  | Small Bolt, Lower Cover                                                             |
| 0.7                     | M <sub>5</sub>  | Components within M <sub>3</sub> and M <sub>4</sub>                                 |
|                         | M <sub>6</sub>  | Rotor Coil, Shaft                                                                   |
|                         | M <sub>7</sub>  | Shaft, Fan                                                                          |
|                         | M <sub>8</sub>  | V Belt Socket, Upper Cover                                                          |
| 0.6                     | M <sub>9</sub>  | Components within M <sub>1</sub> and M <sub>5</sub>                                 |
| 0.5                     | M <sub>10</sub> | Components within M <sub>1</sub> and Separator                                      |
|                         | M <sub>11</sub> | Components within M <sub>6</sub> , Separator and Bearing 2                          |
|                         | M <sub>12</sub> | Components within M <sub>6</sub> , M <sub>7</sub> , Washer 2, Nut and V Belt Socket |
|                         | M <sub>13</sub> | Components within M <sub>2</sub> and M <sub>8</sub>                                 |
| 0.4                     | M <sub>14</sub> | Components within M <sub>9</sub> and Separator                                      |
|                         | M <sub>15</sub> | Components within M <sub>11</sub> , M <sub>12</sub> and Bearing 1                   |
|                         | M <sub>16</sub> | Components within M <sub>11</sub> , M <sub>12</sub> and Upper Cover                 |
| 0.3                     | M <sub>17</sub> | All                                                                                 |

Similarly, the system currently represents dependencies for a single 'perspective' of any viewpoint such as the physical relationships of components. Work is underway to extend the functionality to enable the representation of multiple perspectives within a single matrix, for example, the representation of dependencies from energy, material, and information perspectives. This functionality provides the means to simultaneously consider more than one perspective in the determination of inherent hierarchical modularity.

## 7. Conclusions

A generic system for representing the component structure and inter-component dependencies for design artefacts has been presented. The system evaluates a clustering criterion using the dependencies for any given structure of components. The system also incorporates a genetic algorithm to optimise the clustering objective through the re-ordering of the components.

A ‘Module Strength Indicator’ (MSI) was derived based on the criteria for an optimal modular structure, which is defined as the maximisation of internal dependencies and the minimisation of external dependencies. The application of the MSI to the DSM resulted with a MSM, which represents inherent hierarchical modularity within the product structure.

The system’s functionality was demonstrated through two case studies and illustrated the ability to facilitate the identification of near-optimal component modules without the need for any other domain or artefact specific knowledge. The results of both case studies identified similar module configurations as those defined in the original publications. Minor differences between modules identified were due to the application of domain specific knowledge by the authors of the original research. Inclusion of this knowledge within the original matrices would have influenced the results accordingly.

Another significant contribution of this approach is the identification of inherent modular hierarchy within the product structure, which was not evident from the module identification/definition results from previous case study work.

### Acknowledgements

The authors gratefully acknowledge the support given by the Engineering and Physical Science Research Council who provided the grant RES/4741/0929 and postgraduate studentship no.98319349.

### References

- Andreasen, MM, Duffy, A and Mortensen, NH: 1995, Relation types in machine systems, *WDK Workshop on Product Structuring*, Delft University of Technology, Delft.
- Andreasen, MM, Hansen, CT and Mortensen, NH: 1996, The structuring of products and product programmes, *2nd WDK Workshop on Product Structuring*, Delft University of Technology, Delft, pp. 1-27.
- Blackenfelt, M: 2001, Modularisation by relational matrices – a method for consideration of strategic and functional aspects, in A Riiithuhta and A Pulkkinen (eds), *Design for Configuration – A Debate Based on the 5<sup>th</sup> WDK Workshop on Product Structuring*, Springer, Berlin, pp. 134-152.
- Coates, G, Duffy, AHB, Hills, W and Whitfield, RI: 2000, A generic coordination approach applied to a manufacturing environment, *Journal of Materials Processing Technology*, **107**, 404-411.
- Davis, L: 1985, Job shop scheduling and genetic algorithms, *Proceedings of the First International Conference on Genetic Algorithms and their Applications*, Lawrence Erlbaum, Hillsdale, NJ, pp. 136-140.
- Elgard, P and Miller, TD: 1998, Designing product families, *Design for Integration in Manufacturing in Proceedings of the 13th IPS Research Seminar*, Aalborg University, Fuglsoe, pp. 1-19.

- Eppinger, SD, Whitney, DE, Smith, RP and Gebala, DA: 1994, A model-based method for organizing tasks in product development, *Research in Engineering Design* 6: 1-13.
- Erens, F and Vershulst, K: 1995, Managing system design, *WDK Workshop on Product Structuring*, Delft University of Technology, Delft, pp. 1-14.
- Erens, F and Vershulst, K: 1996, Architectures for product families, *2<sup>nd</sup> WDK Workshop on Product Structuring*, Delft University of Technology, Delft.
- Gershenson, JK, Jagannath Prasad, G and Allamneni, S: 1999, Modular product design: a life-cycle View, *Journal of Integrated Design and Process Science* 3(4): 13-26.
- Glover, F and Laguna, M: 1993, Tabu search, in CR Reeves (ed.), *Modern Heuristic Techniques for Combinatorial Problems*, Blackwell Scientific Publications, Oxford, pp. 70-150.
- Goldberg, DE and Lingle, R: 1985, Alleles, Loci and the travelling salesman problem, *Proceedings of the First International Conference on Genetic Algorithms and Their Applications*, Lawrence Erlbaum Associates, Hillsdale, NJ, pp. 154-159.
- Goldberg, DE: 1989a, *Genetic Algorithms in Search, Optimisation and Machine Learning*, Addison-Wesley, MA.
- Goldberg, DE: 1989b, Sizing populations for serial and parallel genetic algorithms, *Proceedings of the Third International Conference on Genetic Algorithms*, Morgan Kaufmann, San Mateo, CA, pp. 70-79.
- Gonzalz-Zugasti, JP and Otto, KN: 2000, Modular platform based product family design, *Proceedings of the Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, ASME, pp. 13-26.
- Greffenstette, JJ, Gopal, R, Rosmaita, B and VanGucht, .: 1985, Genetic algorithms for the travelling salesman problem, *Proceedings of the First International Conference on Genetic Algorithms and Their Applications*, Lawrence Erlbaum, Hillsdale, NJ, pp. 160-168.
- Gu, P, Hashemian, M and Sosale, S: 1997, An integrated modular design methodology for life-cycle engineering, *Manufacturing Technology CIRP Annals* 46(I): 71-74.
- Holland, JH: 1962, Outline for a logical theory of adaptive systems, *Joint Association of Computing Machinery* 9: 297-314.
- Huang, C and Kusiak, A: 1998, Modularity in design of products and systems, *Transactions on Systems, Man and Cybernetics Part A – Systems and Humans* 28(1): 66-77.
- Järventausta, S and Pulkkinen, A: 2001, Enhancing product modularisation with multiple views of decomposition and clustering, in A Riiiahuhta and A Pulkkinen (eds), *Design for Configuration – A Debate Based on the 5<sup>th</sup> WDK Workshop on Product Structuring*, Springer, Berlin, pp. 153-168.
- Jiao, J and Tseng, MM: 1999, A methodology for developing product family architecture for mass customisation, *Journal of Intelligent Manufacturing* 12: 3-20.
- Kirkpatrick, S, Gelatt, CD and Vecchi, M: 1983, Optimisation by simulated annealing, *Science* 220(4598): 671-680.
- Kusiak, A and Park, K: 1990, Concurrent engineering: decomposition and scheduling of design activities, *International Journal of Production Research* 28(10): 1883-1900.
- Kusiak, A and Huang, C: 1996, Development of modular products, *IEEE Transactions on Components, Packaging and Manufacturing Technology*, Part A 19(4).
- Miller, TD and Elgard, P: 1998, Defining modules, modularity and modularisation, *Design for Integration in Manufacturing*, *Proceedings of the 13th IPS Research Seminar*, Fuglsoe, Aalborg University, pp. 1-19.

- Miller, TD: 1999, Modular engineering of production plants, in U Lindemann, H Birkhofer, H Meerkamm and S Vajna (eds), *Proceedings of the International Conference on Engineering Design*, Schriftenreihe WDK 26, Munich.
- Minsk ML: 1990, Process models for cultural integration, *Journal of Culture* **11**(4): 49-58.
- Muffato, M: 1999, Introducing a platform strategy in product development, *International Journal of Production Economics* **60-61**: 145-153.
- Murata, T and Ishibuchi, H: 1994, Performance evaluation of genetic algorithms for flow shop scheduling problems, *Proceedings of the First IEEE Conference on Evolutionary Computation* **2**: 812-817.
- O'Grady, P and Liang: 1998, An object oriented approach to design with modules, *Computer Integrated Manufacturing Systems* **11**(4): 267-283.
- Oliver IM, Smith, CJ and Holland JRC: 1987, A study of permutation crossovers on the travelling salesman problem, *Proceedings of the Second International Conference on Genetic Algorithms and their Applications*, Morgan Kaufmann, San Mateo, CA, pp. 225-230.
- Otto, K: 2001, A process for modularising product families, in S Culley, A Duffy, C McMahon and K Wallace (eds), *Proceedings of the International Conference on Engineering Design*, pp. 523-529.
- Pimpler, TU and Eppinger, SD: 1994, Integration analysis of product decompositions, *Proceedings of the ASME Sixth International Conference on Design*, pp. 343-351.
- Salheih, SM and Kamrani, AK: 1999, Macro level product development using design for modularity, *Robotics and Computer Integrated-Manufacturing* **15**: 319-329.
- Smith, JS and Duffy AHB: 2001a, Modularity in support of design for re-use, *Proceedings of the International Conference of Engineering Design, ICED*.
- Smith, JS and Duffy AHB: 2001b, Product structuring for design re-use, in A Riiihuhta, and A Pulkkinen (eds), *Design for Configuration – a debate based on the 5<sup>th</sup> WDK Workshop on Product Structuring*, Springer, Berlin, pp.83-100,
- Sosale, S, Hashemian, M and Gu, P: 1997, Product modularisation for re-use and recycling, *Concurrent Product Design and Environmentally Conscious Manufacturing*, DE **94**, ASME, pp. 195-206.
- Starkweather, T, McDaniel, S, Mathias, K, Whitley, D and Whitley, C: 1991, A comparison of genetic sequencing operators, *Proceedings of the Fourth International Conference on Genetic Algorithms*, Morgan Kaufmann, San Mateo, CA, pp. 69-76.
- Steward, DV: 1981, The design structure system: a method for managing the design of complex systems, *IEEE Trans. Engineering Management* **28**: 71-74.
- Suh, NP: 1990, *The Principles of Design*, Oxford University Press, Oxford.
- Syswerda, G: 1991, Scheduling optimisation using genetic algorithms, in L Davis (ed.), *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, pp. 332-349.
- Theory and Methodology, Minneapolis, MN, Sept: 1994. Also, MIT, Sloan School of Management, Cambridge, MA, Working Paper no. 3690-94-MS.
- Todd, D: 1997, *Multiple Criteria Genetic Algorithms in Engineering Design and Operation*, Ph.D. Thesis, Engineering Design Centre, University of Newcastle upon Tyne.
- Ulrich, K and Tung, K: 1991, Fundamentals of product modularity, *Issues in Design/Manufacture Integration*, DE **39**: 73-79.
- Whitfield, RI, Duffy, AHB, Coates, G and Hills W: 2001a, Efficient process optimisation, submitted to *Journal of Concurrent Engineering Research and Applications*.
- Whitfield, RI: 2001b, *Effective Combinatorial Optimisation*, CAD Centre internal report, CADC/01-06/R/03, DMEM, University of Strathclyde.

- Whitley, D, Starkweather, T and Fuquay, D: 1989, Scheduling problems and the travelling salesman: the genetic edge recombination operator, *Proceedings of the Third International Conference on Genetic Algorithms*, Morgan Kaufmann, San Mateo, CA, pp.133-140.
- Zamirowski, EJ and Otto, KN: 1999, Identifying product portfolio architecture modularity using function and variety heuristics, *Proceedings of the 11<sup>th</sup> International Conference on Design Theory and Methodology*, ASME, Las Vegas, NV, DETC99/DTM-8760.
- Zhang, Y: 1998, *Computer-Based Modelling and Management for Current Working Knowledge Evolution Support*, Doctoral Thesis, DMEM, University of Strathclyde, Glasgow.

## PERSPECTORS

*Inferring spatial relations from building product models*

JOHN HAYMAKER, MARTIN FISCHER AND JOHN KUNZ  
*Stanford University*  
USA

**Abstract.** Building product models are beginning to find their way into AEC practice. They are proving useful for coordinating large multidisciplinary design and construction teams. In an evolving design and construction planning process, building components are added, modified, or deleted from the product model, causing important spatial relationships to emerge. In addition, new criteria can emerge throughout this design and planning process, causing particular spatial configurations of building components to become of interest. Current practice for building product models relies heavily on explicit representation or on visual inspection to determine conditions of interest. As models become larger, and more complex, visual inspection and/or explicit representation become prohibitively difficult. Formalized automated reasoning mechanisms are needed to complement formalized representation strategies. This paper presents two cases from the Walt Disney Concert Hall, a project using a detailed 3-D building product model, where current practice failed to provide the team with required information. Practitioners need a user-customizable tool that can analyze the building product model and consistently and rapidly identify instances of these spatial conditions based on the current state of the model. We formalize reasoning mechanisms called perspectors that analyze a building product model and add objects, attributes, or relationships, based on the analysis. We use these perspectors to infer implicit spatial relationships between building components, making these relationships explicit.

### **1. Introduction: The Need for Automated Reasoning Mechanisms for Building Product Models**

The immediate purpose of this research is to enable queries for emergent spatial configurations of building components in building product models. When collaborating to improve a design and its construction plan, design

and construction professionals routinely modify, add, or delete design information. Consequently, new important spatial configurations between building components emerge. Today, practitioners must identify these emergent spatial configurations in a slow, manual, and error-prone process that hinders the flexible reuse of product models across disciplines and through the life of a project. If product models supported queries about the implicit spatial configuration of building components, professionals could rapidly construct consistent and up-to-date project representations for their work, and product models created by one discipline could be more readily used for the work of other disciplines.

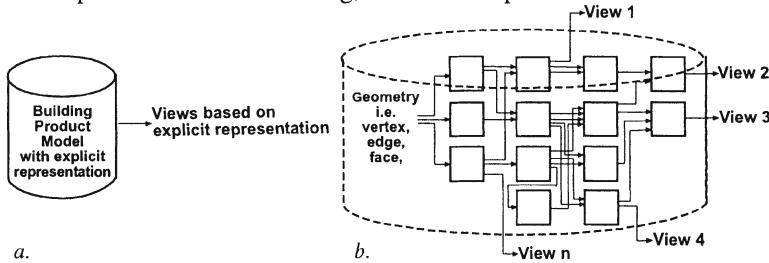
The issue of emergent spatial relationships between building components is symptomatic of a more fundamental issue in building product modeling: Balancing the need for discipline specific representations of a project with the need to integrate these representations with the work of other disciplines. Current practice and research attempts to address these issues in one of two ways: Some attempt the explicit representation of all domain's concerns in one integrated model, while others propose the use of multiple, domain specific models. This paper illustrates that both approaches do not adequately support a design and construction process where design data and criteria are incrementally added, modified, and deleted. Actions of one discipline often have emergent effects on the concerns of other disciplines that remain undetected. In addition, as the design evolves, new criteria often emerge that the existing data representation is unable to support.

This paper therefore argues that a formalized approach that augments representational approaches with automated reasoning about these representations is the best approach to adequately handle emergent conditions and criteria. We propose mechanisms, called 'Perspectors'<sup>\*1</sup>, that modularize the process of constructing domain specific representations of a building product model. Given an existing representation of a building product model, each perspector adds, modifies, or deletes objects, attributes or relationships to this representation, shown schematically in Figure 1. This representation modification can involve explicit representation. For example the 'Beam Perspector' could annotate product model geometry as a 'beam' through user selection of which geometry constitutes a beam. Or the view modification can involve automated reasoning, algorithmically inferring from the geometry which shapes satisfy the beam criteria. The net result of this 'Beam Perspector' is a classification of which geometry constitutes a

---

The term Perspector combines the words :  
'Prospector' – referring to the mechanisms that search through a database, and  
'Perspective' – implying a certain semantic bias to that search.

beam. Users can connect Perspectors into larger structures to create more complex views, for example finding beams in the product model that have a particular spatial relationship to slabs. Perspectors should simplify the act of representation, while increasing the flexibility and reusability of building product models by allowing users to construct representations of the project, that incorporate modular reasoning, suitable to a particular user's criteria.

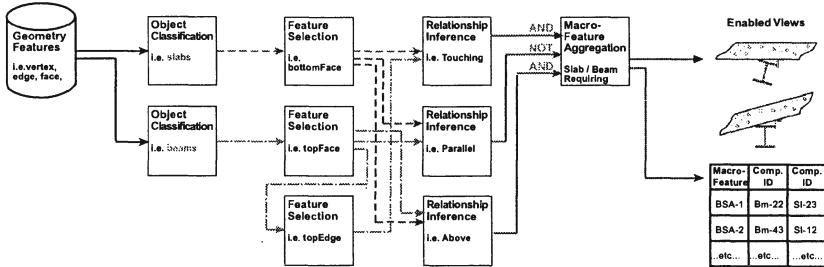


*Figure 1.* A simplified depiction of the role of ‘Perspectors’: mechanisms that take in data and add objects, attributes, and relations (o,a,r), based on a particular perspective. a. shows a product model containing more explicit data, with no perspectors, capable of producing only predefined views; b. shows geometry augmented with perspectors, capable of producing representations of implicit conditions in the product model .

In this paper, we use perspectors to infer complex spatial relationships between building components. We specify the required representation, consisting of a set of relevant geometric features of building components, and a set of spatial relations between these features. Macro-features aggregate spatial relations between features to form a representation of the condition between building components that is of interest. For reasoning, we use a series of perspectors that formalize the transformation of a product model without certain explicit spatial relationships, into a product model with spatial relationships.

With this framework, a user will be able to specify the particular spatial relationships (i.e. above) between particular features (i.e. top face) that are of interest, aggregate these selected relationships between features into a macro-feature (i.e. slabs that have a particular spatial configuration with their supporting beam), and direct the computer to search the database for all sets of components that implicitly satisfy the macro-feature requirements. The system can then explicitly create these spatial relationships in the product model, and present the instances of the specified criteria to the user with different types of views, such as a tabular list, or a 2D or 3D representation, suited to the task at hand. Figure 2 gives a diagrammatic preview of our system in the context of a test case, presented in Section 2,

involving a particular set of spatial relationships of interest between a slab and a beam. The system is explained in Section 4



*Figure 2.* A series of Perspectors (denoted by a box) work together to transform a building product model with implicit spatial relations into a building product model with explicit spatial relations. Output of one perspector serves as the input to the next perspector.

Design and construction planning involves the incorporation and negotiation of multiple domains criteria. Potential solutions satisfying the criteria of one domain often have emergent effects on the criteria of other domains. This research claims that formalized, automated reasoning is needed to manage the propagation of effects between domains. We investigate perspectors in general, and automated spatial reasoning in particular, as a step towards a formalized, general approach to reasoning about these product models. Perspectors enable the automated construction of domain specific representations from a shared geometric product model. If successful, perspectors would:

- Simplify modeling by allowing practitioners to model fewer explicit semantic concepts.
- Increase the usefulness of the models as a collaborative tool by allowing disciplines not doing the modeling to construct useful queries.
- Modularize the construction of queries, allowing incremental construction, modification and reuse of both representation and reasoning components, on an evolving product model.
- Enable ‘live views’ – views are automatically updated when data changes
- Enhance the power of the product model from a static project documentation tool, into an active project-reasoning tool.

### 1.1 OUTLINE OF PAPER

We first present the case for automated spatial reasoning in the context of the Walt Disney Concert Hall (WDCH), designed by Frank O. Gehry and Associates (FOG/A), and constructed by M. A. Mortenson Company (M.A.M.) , in Los Angeles, California. The WDCH's expected completion is 2003. The WDCH project uses 'state of the art' 3-D building product models as an integral coordination tool for design and construction. We present two examples from the WDCH where spatial reasoning mechanisms would have helped the practitioners perform their design and planning tasks.

We then present the current state of research and practice relevant to our goal of inferring spatial relationships of building components from a building product model. We discuss, in light of the WDCH example, why existing *a priori* and *a posteriori* classification approaches alone will not satisfy our requirements for quickly and accurately detecting spatial relationships of interest. We describe the fundamental points of departure, in the domains of product modeling and spatial reasoning that provide the necessary formalisms to augment existing product modeling approaches to achieve our requirements.

Next, we present our system of perspectors that construct spatial relationships between building components. Our representation consists of a set of geometric features of building components, and a set of spatial relationships between these features. These spatial relationships are aggregated to form a macro-feature describing the spatial configuration between building components. For reasoning, we propose a series of perspectors that help to transform a product model without certain explicit spatial relationships, into a product model with spatial relationships.

The final section of this paper discusses future work in this research, and speculates on potential academic and practical contributions and implications of automated spatial reasoning and perspectors for building product models. There has been significant development towards a formalized approach to representing product models for the AEC industry. We investigate perspectors in general, and automated spatial reasoning in particular, as a step towards a formalized, general approach to reasoning about these product models.

## 2. Test Case: The Walt Disney Concert Hall

The WDCH is to be the new 2290-seat home for the Los Angeles Philharmonic Orchestra. The Architects, FOG/A, are recognized internationally for their daring designs, their innovative and advanced use of 3-D modeling to represent their designs, and the integral role these models

play in the construction process. To our knowledge, the WDCH represents some of the industry's best practices in terms of the collaboration of multiple disciplines using large amounts of 3-D electronic data. This project, therefore, serves as an ideal test case to explore how designers and construction planners from different disciplines interact with these models to support their work. These best practices introduce new challenges and opportunities in the way professionals interact with the building product models.

Consistent with The Center for Integrated Facility Engineering (CIFE) research practice, the first author spent several months on the job site working along side the construction planners and architects. During this period, while working on the issues of the day and constructing a series of 4-D models, we monitored the current practice looking for real world problems with fundamental theoretical shortcomings that academic research could help resolve.

The 3-D building product model for WDCH is constructed using CATIA V4, a powerful 3-D modeling program common in the automotive and aviation industries. The scope of this paper does not cover the specific process for constructing and sharing this model, for this we refer the reader to Haymaker and Fischer (2001).

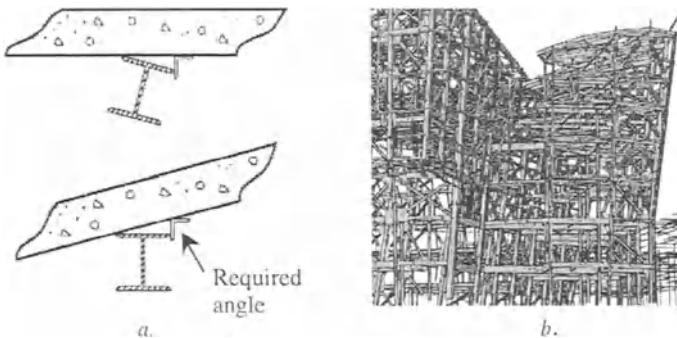
In this section we present two situations from the WDCH project where the current 3-D building product models failed to adequately support practitioners in their design and construction-planning tasks. In the first situation, a slab and beam meet at unique angles, requiring the addition of miscellaneous steel components. In this situation, the designer required a local view of each location where this condition occurred. In the second situation, ceiling panel supports meet the ceiling panels in a number of ways. In this case, the designer requires a more global view, to understand how these conditions are distributed throughout the ceiling system.

In these situations, if a user had a tool that enabled them to formally define conditions of interest conceptually, allowing the tool to find the instances of these conditions rapidly and consistently, the user could have found appropriate solutions faster and more inexpensively.

## 2.1 A SLAB AND BEAM CONNECTION REQUIRING ATTENTION

Emergent spatial conditions of interest can occur when two components are positioned proximately in space. The WDCH has numerous conditions where the steel beams and concrete slabs meet at unusual angles. These occur mainly where a floor slab meets a slanted wall, or where a slab slopes, such as at a roof, or theater seating. In these conditions, as shown in Figure 3, a steel angle is needed to close the gap to assure the slab is properly

supported. The architect and structural engineer do not locate these angles, they are left for the contractor to locate, size, detail and install.



*Figure 3a.* A slab slopes in relation to the beam, requiring an angle for support.  
*b.* A portion of the complex steel model for the WDCH.

Finding these situations required M.A.M. to use a highly skilled and expensive CATIA engineer to manually navigate through the complex 3-D model, Figure 3b, and visually locate and annotate each individual condition. This process is unsatisfactory because:

- The task consumed three full weeks of the engineer's time and was difficult and un-gratifying for the engineer and expensive for M.A.M.
- There is great likelihood of missed conditions that could result in costly delays in the field.
- New conditions could emerge due to design changes after the inspection took place, leaving these conditions undetected.
- New criteria could emerge, for example certain slab types may no longer require the added support due to a metal decking detail, requiring the search to be repeated, taking into account this new criteria.

Practitioners need a user-customizable tool that can analyze the building product model and consistently and rapidly identify instances of these spatial conditions based on the current state of the model.

## 2.2 EDGE OR CANTILEVER SUPPORT CONDITIONS OF CEILING PANELS

The concert hall ceiling is an architectural and acoustical structure hanging from the WDCH roof trusses. The ceiling integrates lighting, mechanical, A/V systems, and a concrete acoustical fill into a curvilinear wooden ceiling finish, shown in Figures 4a and 4b, with roof trusses and ductwork also shown. The design is daring and complicated.



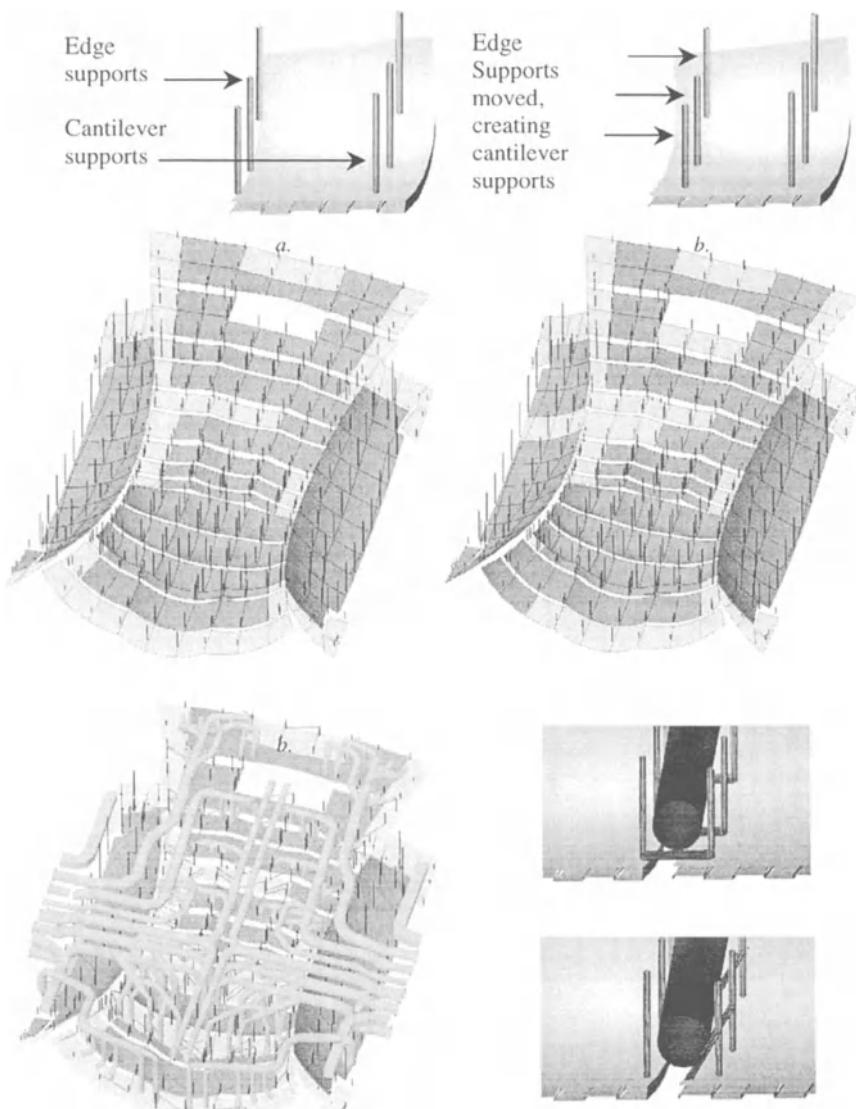
a.



*Figure 4a.* The WDCH ceiling from below (image from a FOG/A physical model) and *b.* from above including roof framing and ductwork (image from the 3-D product model)

FOG/A is proactive in integrating the knowledge of contractors and subcontractors into the design process to build these complex designs. They develop the building product model to the point where it communicates architectural intent, along with some schemes for how it might be built, then forward these to the contractors for their input. The ultimate ceiling framing strategy, whether it becomes a pre-cast concrete system, a light gage steel framing solution, a structural steel framing solution, or some combination of these, was left to the contractor to determine. The design team predetermined the ceiling support drop locations to allow mechanical and other design work to proceed, yet these drop locations could change during design development to suit architectural, mechanical, structural or acoustical demands, Figure 5b. The scope of this paper is not to tell the entire story of the ceiling analysis, but to present a situation during this process where the spatial reasoning mechanisms we propose would have been useful.

The ceiling system was panelized to aid constructability, and these ceiling panels were to be fabricated elsewhere, and trucked to the site for installation. Some of the panels had cantilevered support conditions, while others had edge support conditions, Figures 5a and 5b. While considering the structural solution for the panels, the design team wanted to know how many of these cantilevered conditions occur, how severe the cantilevers are, what the distribution pattern of this condition is throughout the entire ceiling, Figures 5c and 5d, and how this distribution of cantilever conditions interface's with other building components, such as ducts, in the design, Figure 5e, to help decide how best to resolve the cantilever condition, Figures 5f and 5g).



*Figure 5.* Moving supports (a and b) can have emergent and global effects. A practitioner wants to keep track of which panels contain cantilevered supports (c, and, after some supports are moved, d). They also want to understand these cantilever conditions in the context of the ducts (f) so they can choose the best way to support the cantilever (f and g).

While it would be possible to visually inspect the model, determine where the condition occurs, and annotate the building product model accordingly to construct a view such as Figure 5c, this approach contains difficulties:

- Panel supports could be moved, creating the need for repeated inspection, or missed conditions.
- The criterion for what a cantilever is differs depending on the structural solution chosen. The light gage framing method can span less distance than other solutions, before a cantilevered detail is required.
- The visual inspection and annotation process is laborious, error prone, and restricts a more fluid ‘what-if’ design process that would allow the designer to explore many alternatives.

### 2.3 LIMITATIONS OF CURRENT PRACTICE

What is needed for the two cases presented in this section is a methodology, previewed in Figure 2, for defining the spatial condition of interest conceptually, and allowing the computer to do the difficult, repetitive, and error prone task of finding all of the instances matching this spatial condition in the building product model. In this way, if something in the design changes, the computer can update the views automatically. Similarly, if the concept changes, for example, if the criterion for cantilever changes slightly, only the concept needs to be altered, and the views can be updated automatically. In the next section we explain why existing strategies of *a priori* and *a posteriori* classification of data, while a valid starting point, do not adequately address our requirements for a user-customizable tool that can analyze the building product model and consistently and rapidly identify instances of these spatial conditions in the model at any point in time. We identify relevant research that serves as a point of departure towards meeting these requirements

### 3. Point of Departure and Limitations of Current Research

This research seeks to augment existing product modeling approaches with spatial reasoning mechanisms, implemented using our concept of *perspectors*, to create explicit spatial relationships. Figure 6 presents a specification for spatial reasoning *perspectors* as an IDEF diagram. We detail the system in Section 4, but provide the diagram and brief explanation here to set up our discussion of points of departure. This section describes points of departure in the representation of product models from the AEC and mechanical engineering domains in Section 3.1, reasoning about these product models in Section 3.2, and spatial reasoning from the domain of cognitive psychology in Section 3.3.

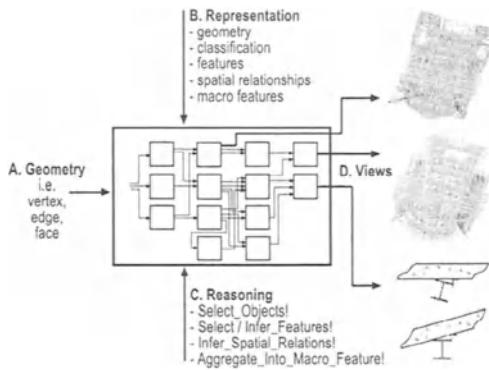


Figure 6. An IDEF diagram describing spatial reasoning perspectors

Figure 6 can be understood as follows:

- A. At the root of a series of perspectors is the 3D project geometry. We explore a geometry consisting of vertices, edges, and faces here, although perspectors could be coded to expect other geometrical representations.
- B. Perspectors expect a particular ontology, beginning with a geometrical representation that the spatial reasoning mechanisms are prepared to analyze. Perspectors then incrementally construct semantics, using the ontology of classification, features, spatial relationships and macro-features.
- C. Perspectors implement reasoning algorithms to analyze the building product model to create explicit macro-features that are used to create views (D). Through analysis of the test cases at the WDCH, we have identified the listed reasoning algorithms (which are each implemented as individual perspectors) as significant steps in inferring the macro-features.
- D. The output of these perspectors is an augmented instance of a building product model containing the macro-feature of interest. This augmented representation can be used to generate views.

### 3.1 PRODUCT MODELS

A number of researchers have worked in the domain of product modeling, defining the relevant objects, attributes and relationships in a building product model. Some of these product-modeling approaches investigate a central shared model incorporating all project views (IFC, 2001) while others propose multiple, domain specific models with integrity relationships between the models (Turk 2001, Rosenman and Gero 1996). Some (Bjork

1987, IFC 2001) explore a *semantically explicit* approach providing specific objects (i.e., 'beam'), attributes (i.e., 'W16 X 50'), and relationships (i.e., 'supports'). Others approach the problem *syntactically*, providing more abstract structures such as objects, attributes and relationships that can be extended to create a particular ontology (Phan 1993, Stouffs, 1997). Some of these approaches can be used *a Priori*, or at design time (IFC, 2001, Gielingh, 1988). Others are intended to be used *a Posteriori*, or during design inspection. (Clayton, 1999, Hakim and Garrett, 1997). We will now discuss the benefits and drawbacks (Zamanian and Pittman, 1999) of each of these approaches, followed by a discussion of our system of perspectors in this context, as a system that maximizes the benefit of each of these approaches, mitigating the drawbacks, and incorporating reasoning as part of the representational approach.

### *3.1.1 Integrated model vs. multiple domain specific models*

A major problem on large, multidisciplinary projects is the coordination and communication between the different disciplines. One approach to addressing these issues is through construction of an integrated product model that contains all of the objects, attributes, and relationships relevant to all of the disciplines involved. As building product models become larger and more complex, and as greater numbers of disciplines seek to collaborate using these models, strategies relying on explicit representation of all disciplines concerns in a shared model break down. The multidisciplinary concerns are too numerous and difficult to represent explicitly by the disciplines doing the modeling, maintaining integrity of relationships in the product model during design collaboration is too complex, and the models are too large to ask each individual discipline to visually inspect the models continuously throughout the process for emergent effects.

Other research explores the use of multiple, domain specific models, similar to the traditional paper-based approach, allowing representations suited to individual domains concerns (for example, the structural engineer may represent a beam as a line annotated as a W16 X 50, and type of connection to other columns and beams explicitly represented. The steel detailer may represent this beam in full detail, with holes, bolts, and connection plates represented explicitly. The architect may want to represent how this beam intersects a non load bearing, fire-rated wall, and is therefore interested in the profile, etc.) Integration of these views is either performed in coordination meetings (similar to the traditional approach) or through complex integration rules (such as the beam in the structural engineer's view is the same as the beam in the architect's view) allowing changes in one domain's view to be propagated to the other domains

representations. When these integration rules are used, the difficulties described in integrated models, explained above, are again encountered.

This paper advocates an integrated geometric product model, using a mixture of shared and domain specific semantic representations to annotate and transform this geometry to achieve domain specific views. We extend other similar approaches by incorporating modular reasoning mechanisms into the representational structure.

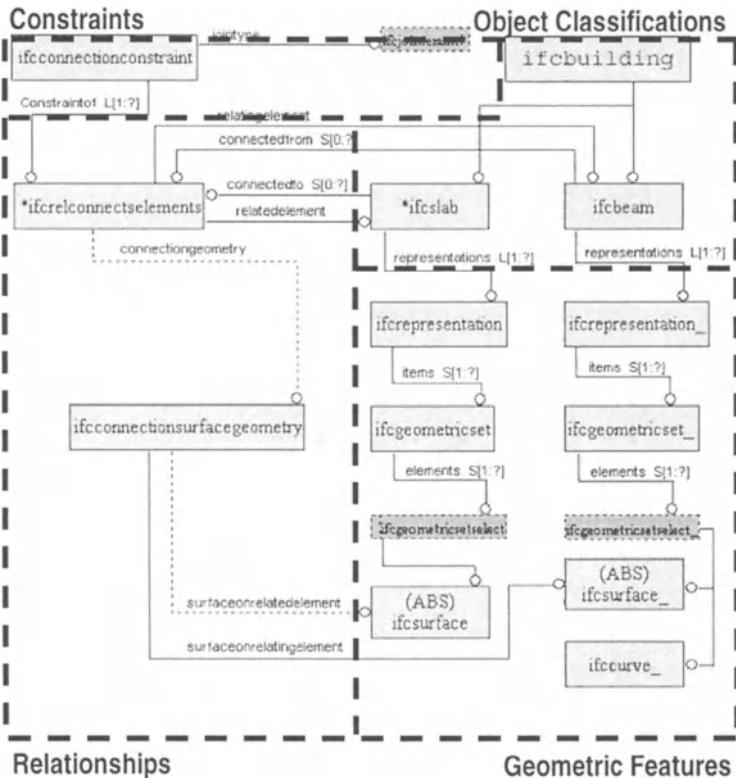
### *3.1.2 Syntactic vs. semantically explicit. approaches*

Existing product modeling approaches contain a wealth of formalisms to represent the syntactics and semantics of real world concepts of interest. Syntactic approaches formalize many of the concepts such as object classification, features, and relationships that we require to represent macro-features between components, Figures 2 and 7. These syntactic approaches provide a framework with which to define semantic concepts, allowing great flexibility in the types of concepts they can describe. However, syntactic approaches do not explicitly formalize specific semantic concepts, making sharing data across projects, and writing general analysis software for the industry difficult. Semantic approaches explicitly define objects (i.e., 'beams'), attributes, (i.e., 'steel grade'), and relationships, (i.e., 'supports'), with which to construct a specific project's product models. These approaches generally enable more semantic data sharing between applications, but the encoded semantics are often too brittle to adequately describe many real world situations.

Figure 7 gives an example of how the Industry Foundation Classes (IFC's) (IAI, 2001), the proposed standard from the Industry Alliance for Interoperability and a semantic approach, can be used to model the slab and beam example from Section 2. Figure 7 identifies notions of classification, geometric features, constraints and relationships, coinciding closely with the steps, specified in Figure 2, that we use to construct spatial relationships. The annotation suggests that, for our goal of automatically inferring spatial relationships from these building product models, the IFC's and other product models contain much of the formalisms required to adequately represent the existence of a relationship between components. However, these representational approaches must be extended. Current product modeling approaches lack a representation of certain features, of the specific spatial relationships between features, and the ability to aggregate these relationships into a descriptive macro-feature.

The framework of perspectors is essentially a syntactic approach, but can be used to encode semantic concepts. we use the IFC as a point of departure, using their formalism where it is available to describe many semantic

concepts such as ‘beam’. The emphasis on the syntactic approach allows for many project specific concepts to be built on top of these shared industry concepts. In addition, by formalizing reasoning as part of this syntactic approach, we simplify the application of user defined semantics to the specific project data.



*Figure 7.* A (simplified) representation of the slab-beam example using the IFC ontology, represented in Express-G. Concepts not explicit in the IFC begin with a capital letter.

### 3.1.3 *A Priori* vs. *A Posteriori* application of schema

Some research intends for the application of semantic schemas to the design at design time. Some CAD programs (IAI 2001) allow you to create IFC objects such as beams, slabs, and spaces, etc, explicitly during design. Other

approaches acknowledge that disciplines other than the ones doing the modeling will want to construct different semantics onto the data, allowing users to semantically label the design as they inspect it.

Attempts to explicitly represent conditions as they are created (i.e., when components are positioned), which we term ‘*a priori*’ because they are classified before the view is required, are difficult because:

- The discipline generating the change often does not understand the implications on other domains, or does not understand that the configuration has significance until later in the design process.
- Such exhaustive data entry interrupts the natural way designers work.
- There is great potential for errors and omissions.
- Concepts often emerge during the design and planning process that cannot be predicted.

Asking each discipline to inspect and classify the data after it has been created (i.e., when the view is required), which we term ‘*a posteriori*’, is also not practical because:

- Data is often updated incrementally, requiring the users to repeat the review task incrementally.
- Visually inspecting a database with thousands of components is prohibitively time consuming.
- There is great potential for errors and omissions.

The approach of *perspectors* allows for both strategies. A software package could construct a beam *perspector* as it generates the data, or it could allow a user to create this *perspector* when inspecting the geometry. The system of *perspectors* extends both of these approaches by offering a third alternative, representing the concept of beam algorithmically, and allowing the computer to infer which geometry satisfies the beam criteria.

### 3.2 MODEL BASED REASONING

Existing Product modeling approaches also lack the ability to dynamically update to changes in the design, or to address evolving concerns of those using the model. Some approaches (IFC 2001; Eastman 2001) define constraints between objects that include tests for validity of these constraints. However, these constraints are applied in an *a priori* fashion as part of the object definition, not allowing for a *a posteriori* automated analysis of the implicit relations between building components. As the WDCH test case shows, practitioners need a mechanism that defines a spatial configuration between components conceptually at any time, and look for

instances in the building product model that satisfy this conceptual spatial relationship of interest.

Existing approaches do not, however, propose a formal way for reasoning about the model. Our goal is to develop and test a theory for perspectors that can be used with any of these product-modeling approaches. We seek a formalized reasoning model to complement representation models, providing the added flexibility to query the model for objects, attributes, or relationships that are not explicitly represented in the model. In this paper we explore perspectors in the context of spatial reasoning.

In the mechanical engineering domain, Rosen and Dixon (1994) recognize the need for the inference of implicit relationships in the product model. He proposed a general process that, through what he calls a process of filtration, annotation, and aggregation, could infer these types of relationships. While these insights for the need and general process are valuable, they did not develop or formalize the notion of annotation (analogous to our requirement for inference of spatial relationships).

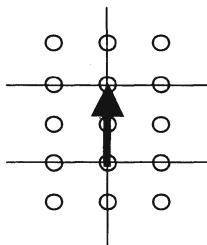
Researchers at the Center for Integrated Facility Engineering (CIFE) and elsewhere have worked in the area of model-based reasoning with models incorporating geometry. Most of this research has been done in the context of a single domain, and on domain specific models. Among some of the CIFE projects, Darwiche (1988) and Aalami (1998) perform model-based reasoning to produce a construction schedule; Akinci (2000) analyzes a 4-D model to infer time-space conflicts for workspace; Akbas (2001) analyzes project geometry with productivity constraints to determine daily work zones; Fischer (1993) analyzes product models for constructability concerns; Han (2000) analyzes an IFC based product model for handicapped accessibility; Korman (2001) performs MEP coordination, and Staub-French (2000) works on the automation of cost analysis. Outside of CIFE others have created similar model-based reasoning systems. Among these Dym et al (1988), and Amor (1992) perform automated Architectural code checking. Others work to perform a series of tasks around an integrated product model Aouad (1997) and Laitinen (1998).

Perspectors, as a mechanism that analyzes a building product model and adds, modifies or deletes objects, attributes or relationships based on the contents of the model are an approach towards a generalized framework for model-based reasoning. Perspectors enable the transformation of an integrated geometric product model into domain specific representations that allow each of the system described above to perform their analysis.

### 3.3 SPATIAL REASONING

To address the two required extensions to the product-modeling domain, representing the spatial constraints between components, and inferring these constraints from a building product model, we turn to the domain of qualitative spatial reasoning. In this section we briefly identify relevant research in this domain, which help to represent and infer spatial constraints between geometric features. We describe the formalisms of orientation and topological relations, conceptual neighborhoods of these relations, and frames of reference.

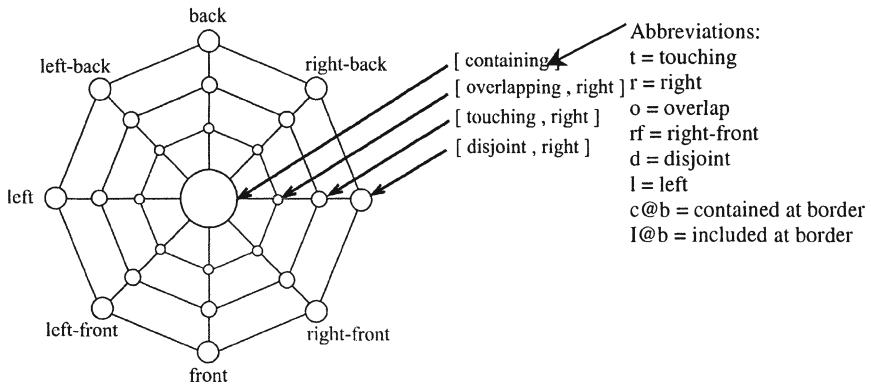
Zimmerman and Freksa (1996) identify the qualitative relationships between a path and a point as consisting of the 15 possibilities shown in Figure 8. Hernandez extends this idea to create a hierarchical ordering, showing that ‘in front of’, can be further decomposed to ‘front-right’, ‘front’, and ‘front-left’, etc. We need to extend these notions to 3-D, however, the insight that the relationships between two components can be classified into a finite and hierarchically structured representation is valuable. For example, a practitioner may look for any slabs that are above the beam (with orientation of ‘above’ implied by gravity), or he may want to specify that all light switches be placed to the right of the doors (in reference to the exiting direction). We discuss orientation in greater detail below.



*Figure 8.* Freksa’s relationships between a path and a point in 2D. We extend these relations to 3-D and apply the hierarchical ordering proposed by Hernandez.

Both Egenhofer (1991) and Hernandez (1994) discuss the notion of topological-orientation pairs to further describe the spatial constraints between objects. In addition to the orientation relations discussed above, one can refer to two objects as ‘disjoint’, ‘touching’, ‘overlapping’, ‘contained-within’, etc. It is therefore possible to further constrain the relationships between two objects, or features of objects, as a pair of orientation and topological constraints. One can therefore say something is [above, touching]. For example, we may ask for the slabs that are above and touching the beams. These researchers also identified the existence of

conceptual neighborhoods, where certain orientation-topological pairs are neighbors of other orientation-topological pairs. It should therefore be possible to identify neighborhoods of relations that are of interest (i.e., [front,touching] → [right,disjoint] ), rather than just an individual relationship. Figure 9 shows a partial conceptual neighborhood diagram of orientation-topological pairs, as formalized by Hernandez.



*Figure 9.* Hernandez' conceptual neighborhood of orientation-topological pairs

Orientations occur within a frame of reference. If one says ‘the light switch is to the right of the door’, this occurs within one of three frames of reference:

- Intrinsic: where orientation is given by an inherent property of the object (i.e. right, in reference to the doors exiting direction)
- Extrinsic: where external factors impose orientation on the reference object (i.e., right in reference to the rooms orientation)
- Deictic: where orientation is imposed by the point of view from which the object is seen. (i.e., right from where the observer is standing)

These formalisms from the qualitative spatial reasoning domain provide a framework to represent the individual spatial constraints between geometric features that can then be aggregated to form a descriptive spatial relationship between building components. We now combine these contributions from the product modeling and spatial reasoning domains to specify a system that can achieve our requirements for a fast, error-free, and automated process of detecting spatial relationships of interest in building product models.

#### 4. Using Perspectors to Infer Spatial Relationships

Motivated by the requirements for a consistent, rapid, on demand, and up-to-date process for inferring implicit spatial relationships between components in a building product model, two limitations in current product modeling research have been uncovered.

First, there is no formalism to represent spatial relationships, and aggregations of spatial relationships between geometric features of building components to describe spatial configurations of building components. Formalisms from the product-modeling domain appear sufficient for representing geometric features (as shown in Figure 7, although these need to be extended to include semantic modifiers such as ‘top face’), and formalisms from the spatial reasoning domain appear to be an excellent starting point for representing the relationships between the features. A representation structure to aggregate the relationships into a descriptive macro-feature is still needed. We currently use a simple aggregation. For example:

$$\begin{aligned} & (\text{slab.bottomFace } \textit{Above} \text{ beam.topFace}, \text{ AND} \\ & \quad \text{slab.bottomFace } \textit{Touching} \text{ beam.topEdge} \text{ AND NOT} \\ & \quad \text{slab.bottomFace } \textit{Parallel} \text{ beam.topFace}) \end{aligned}$$

The second limitation in current product modeling research is the lack of a formal way to automatically detect instances of these spatial relationships in the building product model. We formalize the notion of perspector, as a mechanism that analyzes a product model to produce new objects, attributes, and relationships for that product model as a necessary building block for constructing complex reasoning structures. Figure 2 shows a series of perspectors, which, as an aggregate describe the skewed spatial relationship between the slab and the beam. Figure 10 re-represents this process, showing how an instance of slab and beam are processed through the perspectors to check for, and represent the concept

Analysis of a number of WDCH test cases suggests that this process: *Classification → Feature Selection → Relationship Inference → Macro-Feature Aggregation* can be used to describe many different spatial relations between components.

Figure 10 can be understood as follows:

1. The ‘classification’ perspectors, search the data for ‘slabs’ and ‘beams’. These could be more specific classifications (i.e., ‘cast-in-place’ concrete slabs) or more general classifications (i.e., ‘structural members’). .
2. The ‘Feature Selection’ perspector accesses the component’s available features. In our system we explore faces, edges, and points, but additional semantics can be applied to the feature selection. For

example, the user may be interested in the ‘top face’, or the ‘web’. These can be either explicitly represented in the database, or a perspector could be written to infer which is the relevant feature.

3. A feature on one component is compared to a feature on another component through a relationship perspector. For example, the user states that the ‘top edge of the ‘beam’ should be ‘touching’ the ‘bottom face’ of the ‘slab’. These relations could be represented explicitly in the data, or more likely, they are implicit, and these individual perspectors would infer the relationship between these features.
4. If the relationship conditions are met, they are aggregated into the macro-feature describing the configuration of interest.

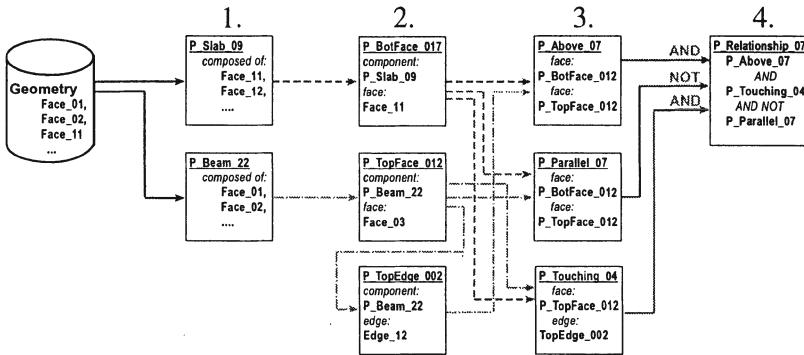


Figure 10. Perspectors analyzing a geometric model for the slab-beam macro-feature

## 5. Conclusions, Implications and Future Work

The test cases from the Walt Disney Concert Hall that showed that explicit representation in a product model breaks down as the models become large, and the multi-disciplinary concerns become complex. Specifically, the test cases showed how emergent spatial relationships between building components could not have been represented a priori, and visually identifying conditions a posteriori is equally problematic.

We present the notion of a perspector as a mechanism that analyzes a building product model, and returns a building product model with additional objects attributes and relationships. This reasoning capability would allow practitioners to construct simpler product models, knowing that certain types of objects, attributes, and relations can be easily inferred. We explore a product model consisting of solely geometric features (face, edge, vertex). Semantics are applied to features through layering of perspectors.

Not all of these transformations need to be reasoning based. Perspectors can employ both reasoning and representation for the same concept. For example, ‘Object Classification’, can simply be a query to the database requesting all geometry that has been classified as a beam. However, object classification could also embody analysis that infers from a series of geometrical features (faces, edges, vertices, etc.) which objects are beams. The output of the ‘beam perspector’ can then be used for higher-level queries. Such a modular approach would allow multiple representation vs. reasoning strategies on the same database, and allow incremental additions to the reasoning capability on the building product model. We explore the use of perspectors to infer spatial configurations between components.

Working with perspectors should simplify modeling by allowing practitioners to model fewer explicit semantic concepts, while increasing the usefulness of building product models as a collaborative tool by allowing disciplines not doing the modeling to construct useful queries. Perspectors could help transform the product model from a static project documentation tool, into an active project-reasoning tool, providing consistent, rapid, and up-to-date views of user-defined concepts.

We are currently implementing a prototype using the WDCH test case. Using the spatial reasoning mechanisms, users will be able to construct spatial relationship queries on the complex geometry to find conditions of interest. We will test the performance for locating these conditions using this system against actual performance at the WDCH. We will also seek to implement other types of reasoning and product model transformations using the formalism of perspectors.

## Acknowledgements

We would like to thank the following for their support in this research: M.A. Mortenson, Frank O. Gehry and Associates, Martin Brothers, Columbia Showcase, and the entire Walt Disney Concert Hall team; Walt Disney Imagineering, Consolidated Contractors Company, Barbara Tversky, and Ben Suter.

## References

- Aalami, F, Fischer, M, and Kunz, J: 1998, *AEC 4D Production Model: Definition and Automated Generation*, Working Paper Nr 52, CIFE, Stanford University
- Akbas, R, Fischer MA and Kunz J.C: 2001, Formalizing domain knowledge for construction zone generation, *Proceedings of the CIB-W78 International Conference IT in Construction in Pretoria*, South Africa, pp.16-30,
- Akinci, B; and Fischer, M: 2000, 4D WorkPlanner - A prototype system for automated generation of construction spaces and analysis of time-space conflicts, in R Fruchter, P-M

- Feniosky and WM Kim Roddis (eds), *Eighth International Conference on Computing in Civil and Building Engineering (ICCCBE-VIII)*, Stanford University, pp. 740-747.
- Amor, R and Hosking, J and Mugridge, W and Hamer, J and Williams, M: 1992, ThermalDesigner: an application of an object-oriented code conformance architecture, *Joint CIB Workshops on Computers and Information in Construction, CIB Proceedings 165*, pp 1-11.
- Aouad G, Marir F, Child T, Brandon P and Kawooya A: 1997, Construction integrated databases- linking design, planning and estimating, *Proceedings of the International Conference on the Rehabilitation and Development of Civil Engineering Infrastructures*, American University of Beirut, pp 51-60.
- Bjork B-C: 1987, *RATAS: A Proposed Finnish Building Product Model*, Studies in Environmental Research No. T6, Helsinki University of Technology, Otaniemi, Finland.
- Clayton, M, Teicholz, P, Fischer, M and Kunz, J: 1999, Virtual components consisting of form, function, and behavior, *Automation in Construction* **8**: 351-367,
- Darwiche, A, Levitt, RE, and Hayes-Roth, B: 1988, Oarplan: Generating project plans in a blackboard system by reasoning about objects, actions, and resources. *Journal of Artificial Intelligence for Engineering Design, Analysis and Manufacturing* **2**(3): 169-181.
- Dym, CL, Henchey, RP, and Gonick, S: 1988. A knowledge-based system for automated architectural code checking, *Computer Aided Design* **20**(3): 137-145.
- Egenhofer, M, Franzosa R : 1991, Point-set topological spatial relations, *International Journal of Geographic Information Systems* **5**(2): 160-174.
- Eastman, C, Jeng, T-S, Assal, H, Cho, M and Chase, S: 1995, *EDM-2 Reference Manual*, Center for Design and Computation, UCLA, Los Angeles.
- Fischer, MA: 1993, Automating constructibility reasoning with a geometrical and topological project model, *Computing Systems in Engineering* **4**(2-3): 179-192.
- Gielingh, W: 1988, *General AEC Reference Model*, ISO TC 184/SC4/WG1 doc. 3.2.2.1, TNO report B1-88-150.
- Hakim, MM and Garrett Jr, JH: 1997, An object-centered approach for modeling engineering design products: combining description logic and object-oriented models, *Journal of AI in Engineering Design and Manufacturing* **11**: 187-198.
- Han, CS, Law, K, Kunz J: 2000, *Computer Models and Methods for a Disabled Access Analysis Design Environment*, Technical Report Nr 123, CIFE, Stanford University.
- Haymaker, J and Fischer, M: 2001, *4D Modeling on the Walt Disney Concert Hall* ; TEC 21 Nr. 38, September 21, pp 7-12.
- Hernandez, D: 1994, *Qualitative Representation of Spatial Knowledge*, Lecture Notes in Artificial Intelligence No. 804, Springer Verlag.
- International Alliance for Interoperability*: 2002, <http://haiweb.lbl.gov/>
- Korman, TM and Tatum CB: 2001, *Development of a Knowledge-Based System to Improve Mechanical, Electrical, and Plumbing Coordination*, Technical Report Nr 129, CIFE, Stanford University.
- Laitinen J: 1998. *Model Based Construction Process Management*, Ph.D. Thesis, Royal Institute of Technology, Stockholm, Sweden.
- Phan, DHD and Howard, HC: 1983, *The Primitive-Composite (P-C) Approach: A Methodology for Developing Sharable Object Oriented Data Representations for Facility Engineering Integration*, Technical Report Nr 85, CIFE, Stanford University.
- Rosen, DW, Dixon, JR and Finger, S: 1994, Conversions of feature-based design representations using graph grammar parsing, *Journal of Mechanical Design, Transactions of the ASME* **116**: 785-792.

- Rosenman, MA and Gero, JS: 1996, Modeling multiple views of design objects in a collaborative CAD environment, *CAD, Special Issue on AI in Design* **28**(3): 207-221.
- Staub-French, S, and Fischer, MA: 2000, Formalisms and mechanisms needed to maintain cost estimates based on an IFC product model, in R Fruchter, P-M Feniosky and WM Kim Roddis (eds), *Eighth International Conference on Computing in Civil and Building Engineering (ICCCBE-VIII)*, Stanford University, pp. 716-723.
- Stouffs R and Krishnamurti R: 1997, Sorts: a concept for representational flexibility, in R Junge (ed.), *CAAD Futures 1997*, Kluwer, Dordrecht, pp.553-564.
- Turk, Z: 2001, Phenomenological foundations of conceptual product modeling in architecture, engineering and construction, *Artificial Intelligence in Engineering* **15**(2): 83-92.
- Zamanian, MK and Pittman, JH: 1999, A software industry perspective on AEC information models for distributed collaboration, *Automation in Construction* **8**: 237 - 248
- Zimmermann, K and Freksa C: 1996, Qualitative spatial reasoning using orientation, distance, and path knowledge, *Applied Intelligence* **6**: 49-58.

## PRODUCT DATA EXCHANGE USING ONTOLOGIES

CHRISTEL DARTIGUES AND PARISA GHODOUS  
*University of Lyon*  
*France*

**Abstract.** In the design anywhere manufacture anywhere paradigm, we need to have appropriate mechanisms for communicating design information to manufacturing software, in particular process planning. This is in addition to CAD-CAD interoperability. However, in current industrial practice, conceptual design, process/resource selection, time/cost estimation, detailed design, and process planning are performed independently, without integrated software tools. Design and manufacturing data and messages cannot be efficiently sent from one system to another. Interoperability is further complicated as we need to communicate between different domains (in contrast to CAD-CAD interoperability). In this paper, we first review the existing methods for product data exchange. Then we describe a new approach which improves the data exchange by taking into account the semantics of data. This approach is based on ontologies, which allows the formalization of concepts related to product development in an appropriate context. Finally, we verify this approach for two commercial software tools in the framework of design and process planning data exchange.

### 1. Introduction

These last decades, efforts for enterprise evolution, in computing environments and engineering methodologies, indicate that the engineering infrastructure will be distributed and collaborative. Indeed, working in an enterprise implies that the persons who are involved in the product development, such as designers, process planners, manufacturers, clients and other related domain personnel, communicate and exchange information. The key point is to allow different CAD applications to exchange data among them. This implies many problems because these applications do not necessarily use the same format for data input and output. Moreover, they can be of different nature. In this context, a

classification of the different existing CAD systems is given in (Fensel 2001):

- traditional CAD systems: they provide a tool to help designers to develop either 2D or 3D models of the design, and more generally comprehensive tools for generating geometric forms,
- knowledge based design systems: they focus first on the symbolic aspects of design and later map a symbolic structure to a geometric model. These systems use essentially artificial intelligence techniques to capture the knowledge of expert designers in a computer,
- immersive CAD applications: they integrate the human senses of the designer as a part of the design by using various immersive environments such as haptic, visual and speech interfaces. They can help the evaluation of the manufacturability of designs.

Thus, interoperability between different systems implies many problems and is the source of important efforts to find suitable solutions. One solution to respond this kind of problems is the development of standards. However, actual standards mostly address the interoperability issue between traditional CAD systems and take into account geometric data communication. Nevertheless, these data are not sufficient: designers need to handle data related to all the steps of the product life cycle (for example functions, behavior or structure of the product, historic of conception, ...). Moreover, some of these steps imply the use of specific mechanisms for communication of information, in design or in manufacturing. These geometric-based standards are unfortunately not sufficient to solve the problem of interoperability between different systems. Moreover, systems used in the different steps of the product development (conceptual design, process/resource selection, time/cost estimation, detailed design, process planning) have not been designed for this purpose. The difficulty is bigger if we consider that a system has needs, objectives and viewpoints different from any other system.

In the context of a collaborative work between experts related to different domains, such as design or manufacturability, using different systems, information exchange is an important problem. In this paper, we try to propose a solution to this problem. The organization of this paper is as follows. In the first part, we will review the existing methods for product data exchange and the related improvements and limits. We will next describe a new approach, which improves the data exchange by taking into account the semantics of the data. This approach is based on ontologies allowing the formalization of concepts related to product development in an appropriate context. Finally, we will apply this approach to two commercial

software tools in the context of data exchange of design and process planning in cutting field.

## 2. Existing Methods for Product Data Exchange

Improving interoperability has been a key point these last years. This had partly led to the elaboration of methods that try to allow information exchange between different applications. These methods can be mainly classified into four categories.

The first category corresponds to a manual translation of all the terms and concepts related to each application to exchange data. The principle of this method is relatively simple and necessitates the study of all the relevant terms and concepts realized by experts of each application. Once this study achieved, experts should associate manually one concept of an application to its corresponding concepts in the second one. Figure 1 gives an illustration of the method.

The problem related to this approach is that it is difficult to implement because an expert for each application is necessary and this process cannot be automated. Moreover, any change in one application implies to review the whole process of the manual translation. Finally, there remains some problems when a concept appears in one application and not in the other. For these reasons, researchers have tried to find another solution.

Another category of existing methods uses an interface to improve data exchange. This solution is used for example when the product is not manufactured only by one enterprise: a part of the product is constructed by a subcontractor. There is a problem when the enterprise and its subcontractor do not use the same software. The objective is to allow each of them to obtain all the necessary data in a comprehensible format.

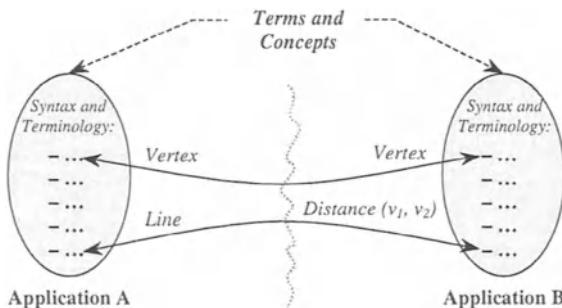
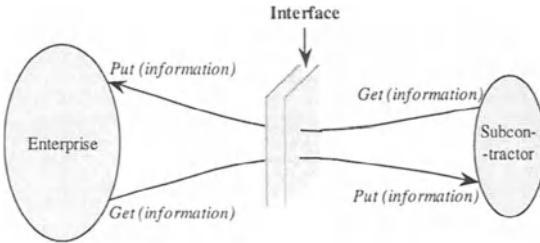


Figure 1. Manual translation between two applications

For this purpose, an interface between the two concerned applications can be used. This interface should allow an application to get back the necessary input data from any other application and to give to any other application the output data. Some functions such as get (information) or put (information) are then necessary. This approach is summarized in Figure 2.



*Figure 2. Use of interfaces for data exchange*

Two examples of such existing interfaces are the OMG and JCAD interfaces. The related problems of this approach are that it is necessary to implement as much interfaces as there are different applications. This can be difficult to manage, if there are a lot of subcontractors. Moreover, one change in an application implies the modification of all the related interfaces, which can necessitate a lot of development efforts.

This last problem has led to the elaboration of a third category of method that is based on the use of a unique standard format to exchange information in order to have a unique intermediary instead of a set of interfaces. Many standards have been elaborated. Some of them have been developed and used nationally and some of them internationally. The principle of their use is simple: before exchanging any information, the rules of translation of any data between the applications and the standard format are elaborated. Data are then translated, using these rules, into the standard format.

The existing standards for data exchange are mainly specialized in geometric data exchange. One of the most used standards is STEP (Standard for the Exchange of Product model data). Beside the geometrical data, this standard concerns other different kinds of data, such as tolerances, kinematics, materials, structure, manufacturing, process planning or design data. However its most complete and accepted part concerns the geometrical part (with AP 203). Figure 3 describes the use of STEP to exchange geometrical data. The last kind of approach for data exchange is the result of the extension of the use of a standard. One extension has then been developed these last years at NIST (National Institute of Standards and Technology).

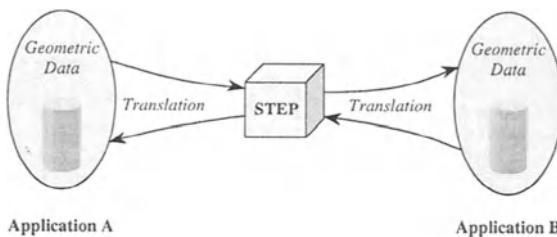


Figure 3. Data exchange using the standard STEP

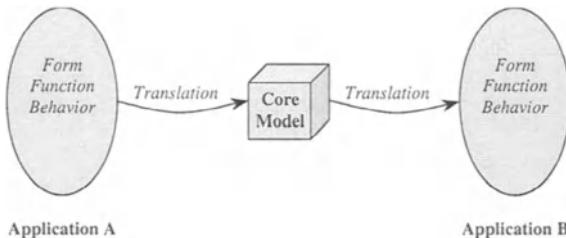
This extension is the result of a project, whose purposes are to identify the need in the knowledge representation and to develop a generic representation for the knowledge of the product development. This could be the base of the future product development systems. This project has led to the elaboration of a core product model (Fenves 2001). This model is based on the entity-relationship schema, which is equivalent to the notions of class and class association in UML (Unified Modeling Language). These two sets of classes are used in this model: object and relationship.

An objective of this project is to construct a representation that would be as robust as possible without having to predefine all possible attributes that might be relevant to any given domain. Hence, the core representation is limited to attributes required to capture generic types of product information and to create relationships among the classes. The representation intentionally excludes attributes that are domain-specific (e.g., attributes of mechanical or electronic devices) or object specific (e.g., attributes specific to function, form or behavior). This representation has been used for data exchange between different applications (Fenves 2001). The principle is similar to the use of a standard: data handled by an application are translated into the core model, which is the standard format. Once the first translation is finished, these new data are translated into the second application format. Figure 4 illustrates this method.

This method has the same advantage as the STEP based method: one change in an application implies only one change in the translation between this application and the core model. However, this model is supposed to be as generic as possible and its use for data exchange implies its adaptation to the application domains.

Despite of this inconvenient, this solution seems to be very promising for data exchange. This is a part of a more important project, which takes into account the knowledge modeling and more precisely the semantic of the data and has been the source of many research works these last years

(Devedzik 1999). One important part of this research concerns the ontologies.



*Figure 4. Data exchange with the core model*

### 3. Data Exchange Based on Ontologies

Ontologies have been developed in artificial intelligence to facilitate knowledge sharing, knowledge management and data retrieval (Brasethvik and Gulla 2001). There are different interpretations of this term (Guarino and Giaretta 1995):

1. an ontology is a philosophical discipline,
2. an ontology is an informal conceptual system,
3. an ontology is a formal semantic account,
4. an ontology is a specification of a conceptualization,
5. an ontology is a representation of a conceptual system via a logical theory,
6. an ontology is a vocabulary used by a logical theory,
7. an ontology is a (meta-level) specification of a logical theory.

In many studies, the fourth definition is used (Hermann and Studer 2001; Fensel 2001; Gruber 2001; Guarino and Welty 2000; Huhns and Singh 1997), (Uschold and King 1995), and a similar definition can be found in Fox and Gruninger (1994). The construction of the ontologies is a complex and costly process in terms of resources and works (Fernandez and Martinez-Bejar 2000). They are often the result of a collaborative work.

It may seem that there is not a big difference between an ontology and a dictionary (Perakath and al. 1994). However, a dictionary is a set of terms put together with their definition expressed in a natural language and an ontology contains a formal language, a very precise syntax and a clear formal semantics which permit to represent precisely a specific domain.

Ontologies provide a vocabulary of terms and relationships with which the domain can be designed (Fensel 2001). They are often captured in the form of a semantic network composed of properties, attributes, constraints, functions and rules related to the handled concepts (Huhns and Singh 1997).

In a more pragmatic manner, ontologies define the vocabulary that will allow designers to exchange requests (Gruber 2001). They take into account some relationships that connect the different concepts to each other (Huhns and Singh 1997): generalization and inheritance, aggregation, instantiation, owns, causes and constraints.

Because of this huge amount of data that should be managed, the ontology design process is very complex and is related to some criterion (Gruber 1993). The ontology design process contains the following stages (Uschold and King 1995):

- identification of the purposes,
- construction of the ontology : capture and coding of the ontology and integration of the existing ones,
- evaluation,
- documentation.

In the first step, the designers must determine why the ontology is built (for example how people will use it) in order to understand the related context. The second step, which is described in (Uschold and King 1995), corresponds to the construction of the ontology, which first implies its capture. There are several ways to capture an ontology. In (Uschold and King 1995), this capture begins with an identification of the key concepts and relationships in the domain of the ontology. This identification is followed by their definition and by the identification of all the terms that refer to the concepts. Another ontology capture method is proposed in (Fox and Gruninger 1994). The objects manipulated by the ontology are first identified and represented with constants and variables that will permit a better understanding of these objects. Using the properties of the objects and the relationships that connect them, the object's definitions are refined. These last are represented by predicates.

Despite of the differences between these two methodologies, there are some key steps in the capture of an ontology:

- identification of the concepts manipulated in the ontology domain,
- definition of these concepts, implying the identification and the definition of all the data related to the objects and their relationships. These data also comprise the different terms that are applied to a given object.

Once the concepts, attributes, relationships and terminologies are defined, they must be represented in an adequate language. The choice of such a language must respect some criterion (Uschold and King 1995):

- is the language clear?,

- what is the conceptual distance of the language (how the semantic primitives are close to the language and how the user thinks about the concepts that must be represented)?,
- is the language expressive?,
- is the language related to any standard?,
- is the language translatable/transportable?,
- is the language supported by any methods or guidelines for its use?,
- does the language have formal semantics that may insure the consistency?,
- is the language easily comprehensive by the users?,
- what kind of prerequisite knowledge are required to use it?,
- is the language flexible?.

Ontologies can serve as a basis for the development of a method for information exchange. This is done at the NIST, where a project called PSL (Process Specification Language) is developed to improve interoperability between different systems (Schlenoff et al. 1999).

#### **4. Application: Data Exchange Between Two Softwares in the Cutting Domain**

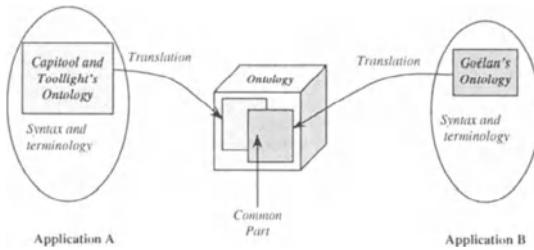
As a real case study we consider the exchange of data between two companies: Tool and CNIndustries (Deshayes et al. 2001). The Tool company is specialist in the cutting domain and has two software called Capitool and Toollight which acquire the knowledge of cutting domain and help experts to decide the optimized cutting parameters. The CNIndustries company develops a CAD/CAM tool called Goélan. The ontology that we define has the objective to improve exchange between these applications for the cutting domain.

The method that we use is similar to the data exchange with the core model method described in section 3: an ontology is developed and the data of an application are translated in a neutral format using this ontology, Figure 5.

The development of an ontology implies to understand the common domain of the two applications that exchange data. In our case the common domain is the cutting domain. We should identify and define the related concepts of this domain.

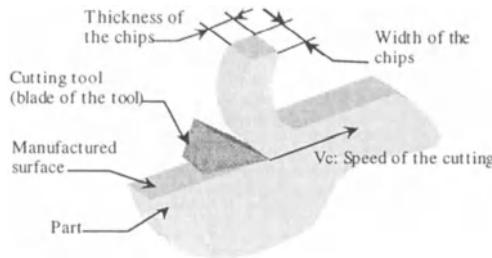
The cutting is a machining process by removal of material which allows, to obtain from a raw part a final part. The tool follows a trajectory with regard to the part fixed on the machine tool, Figure 6. There exists two

categories of tools: fixed tools and turning tools. The fixed tools are put on a lathe.



*Figure 5.* Data exchange between Tool and CNIndustries using an ontology

The turning tool are put on a drilling or a milling machine. Figure 7a gives an illustration of a fixed tool, and Figure 7b gives an illustration of a turning tool. In Figure 7a the term “machine tool” specifies the device on which are placed all the other elements. The tool holder links the machine tool with the handle, the cutting tool holder and the cutting tool. The activity of design and manufacturing of a cutting part is presented in Figure 8 using IDEF0 formalism.



*Figure 6.* A representation of the cutting of material

In our study, we have considered the following five main domains that are interconnected Figure 9: the product description, the equipment description, the tool description, the process description and finally the process parameters.

In this paper we describe four of these domains (product description, tool description, equipment description and process description) for the different viewpoints that are associated with CNIndustries's viewpoint, Tool's viewpoint and a generic viewpoint.

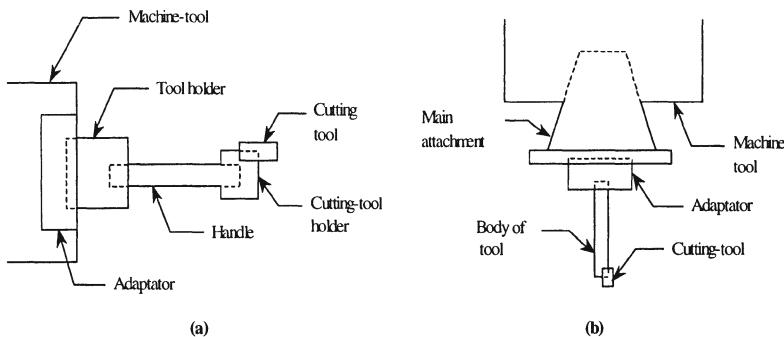


Figure 7. Representation of the two kinds of tool used in the cutting domain: (a) a fixed tool and (b) a turning tool

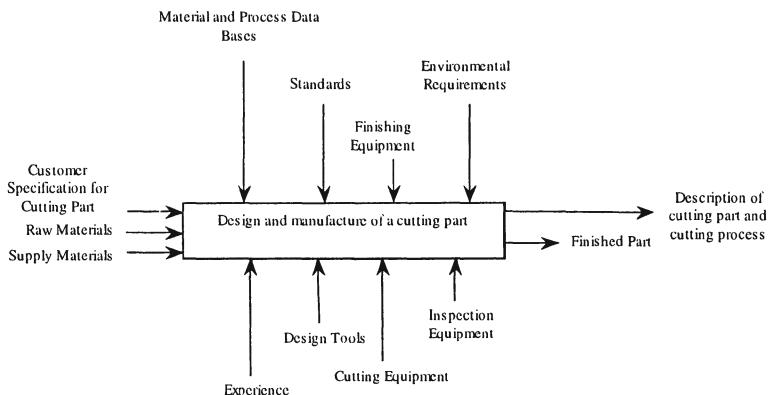


Figure 8. Activity of design and manufacturing of a cutting part

To describe the ontologies, we have used the IDEF5 method. The IDEF5 method is a pragmatic tool, which provides a cost-effective mechanism to acquire, store and maintain scaleable and re-usable ontologies. On the other hand, it is usable by a personnel at varying skill levels and from a variety of different kinds organizations (Perakath et al. 1994). One of the advantage of the IDEF5 method is that it allows the representation of concepts that are not managed by EXPRESS or EXPRESS-G, such as the different states of an object or the relation of entities with their instances.

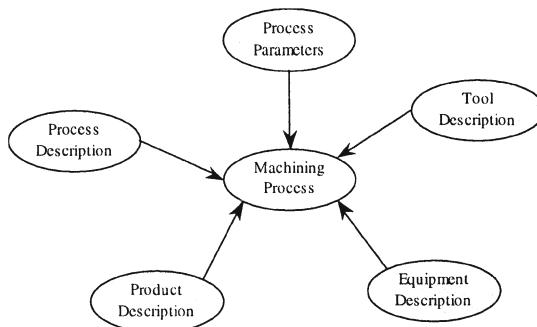


Figure 9. Domains involved in the machining process

#### 4.1 PRODUCT DESCRIPTION

In a first step, a study has been made to compare the terms used by CNIndustries and Tool, Table 1. This table shows how the different entities involved in the product description are defined in the two viewpoints.

TABLE 1. Comparison of vocabulary defined by CNIndustries and Tool.

| Goélan CAD-CAM software                                                                                                 |                         | Toolight cutting tool software                                                              |                         |
|-------------------------------------------------------------------------------------------------------------------------|-------------------------|---------------------------------------------------------------------------------------------|-------------------------|
| PRODUCT entity                                                                                                          |                         |                                                                                             |                         |
| Part:<br>represents the manufactured part                                                                               | Material of composition | Part:<br>represents the manufactured part but includes also information about the operation | Material of composition |
|                                                                                                                         | Geometrical shape       |                                                                                             | Geometrical shape       |
|                                                                                                                         | Machining to realize    |                                                                                             | Machining to realize    |
|                                                                                                                         | Machining quality       |                                                                                             | Machining quality       |
|                                                                                                                         | Dimensions              |                                                                                             | Manufactured length     |
|                                                                                                                         |                         |                                                                                             | Manufactured diameter   |
| Tool: represents the active part. The tool has a designation, which includes all the information about the tool-holder. | turning                 | Tool:<br>represents the tool or the insert                                                  | turning                 |
|                                                                                                                         | milling                 |                                                                                             | milling                 |
|                                                                                                                         | drilling                |                                                                                             | drilling                |
| Tool-holder                                                                                                             | Attachment 1            | Tool-holder                                                                                 | turning                 |
|                                                                                                                         | .....                   |                                                                                             | milling                 |
|                                                                                                                         | Attachment 5            |                                                                                             | drilling                |
| Part-holder                                                                                                             | Not formalized          | Part-holder                                                                                 | Not considered          |

The ontology's schema representing the product description of CNIndustries is presented in Figure 10. In this representation, a product can be either a part, a part-holder or a set of tool-attachment. It is made of some material and it is represented by a geometrical form, which can be of different kinds (B-Rep or basic geometrical entities).

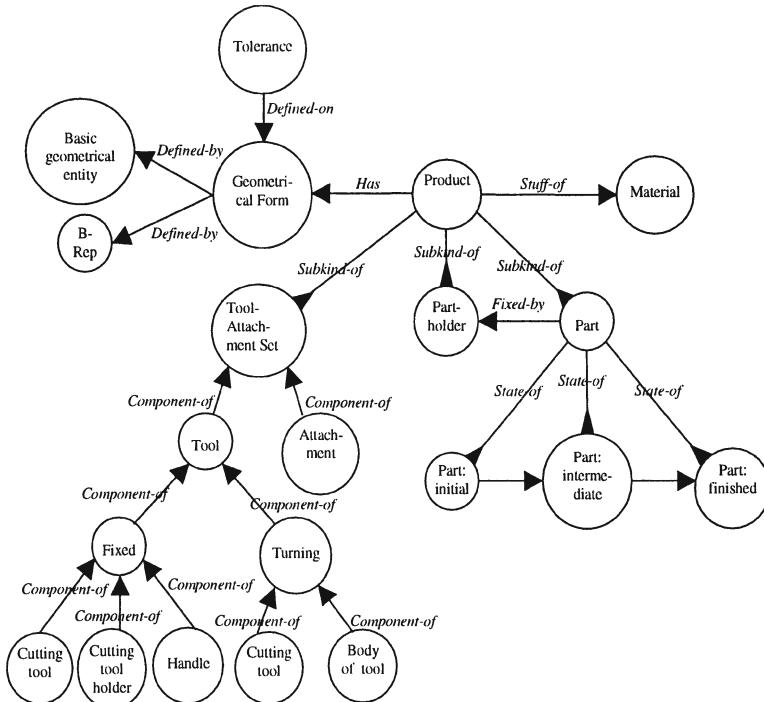


Figure 10. Product representation of CNIndustries

A part can have different states: it can be an initial part (at the beginning of the process), an intermediate part (if one treatment has been already done) or finished part (when the cutting process applied on this part is finished). The part holder is used to fix the part during the cutting process, in order to avoid any movement.

The "Tool-Attachment Set" kind in the schema is composed of a Tool and an Attachment, a Tool can be either a fixed tool or a turning tool, and a fixed tool is composed of a cutting tool, a cutting tool holder and an handle, whereas a turning tool is composed of a cutting tool and a body of tool. Examples of fixed tools and turning tools are shown in Figure 11.

We have also defined the ontology schema of product representation of the Tool company, Figure 12. This representation is similar to the CNIndustries product representation. A product can still be a part, a part holder, a machine tool or the set of tool-tool holder. A part has different states, which are initial, intermediate or finished. It is also connected to the part holder. The product has a geometrical form, which can be represented either by a B-Rep or by basic geometrical entities.

A tool can be either fixed or turning. For the Tool company, a fixed and a turning tool are composed of a cutting tool or a mono-bloc body of tool if the body of tool and the cutting tool are made in only one bloc. From these two different product representations, we have defined a generic product representation that could be used to exchange data between Tool and CNIndustries.

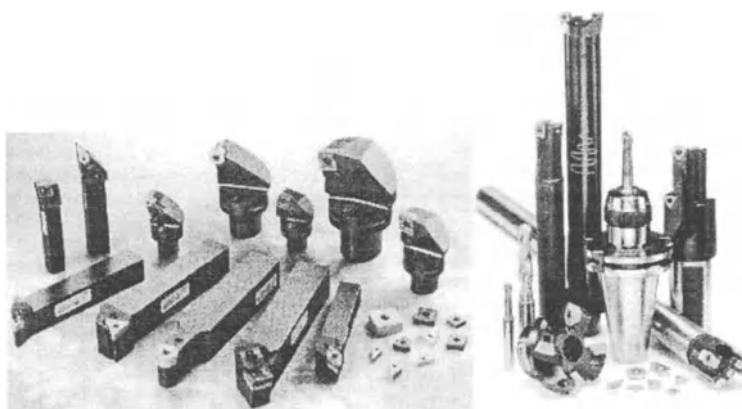


Figure 11. Fixed and Turning Tools considered by CNIndustries.

For the generic product representation, Figure 13, we have used some terms defined in STEP product model. The product representation should be able to describe the product during its life cycle; hence, each definition of the product can be described and is traceable in the model. The instance of product version concept is used to describe the products at different times, and the geometry of the product is associated to the product definition.

To support the connections between a product and its related information, for example assembly, tolerance and shape representations, the concept of product definition relationship is defined. The product relationship can be used to define assembly relationships where the relating product definition represents the assembly and the related product definition

represents a component of the assembly. Products can also be designated as belonging to specific product categories

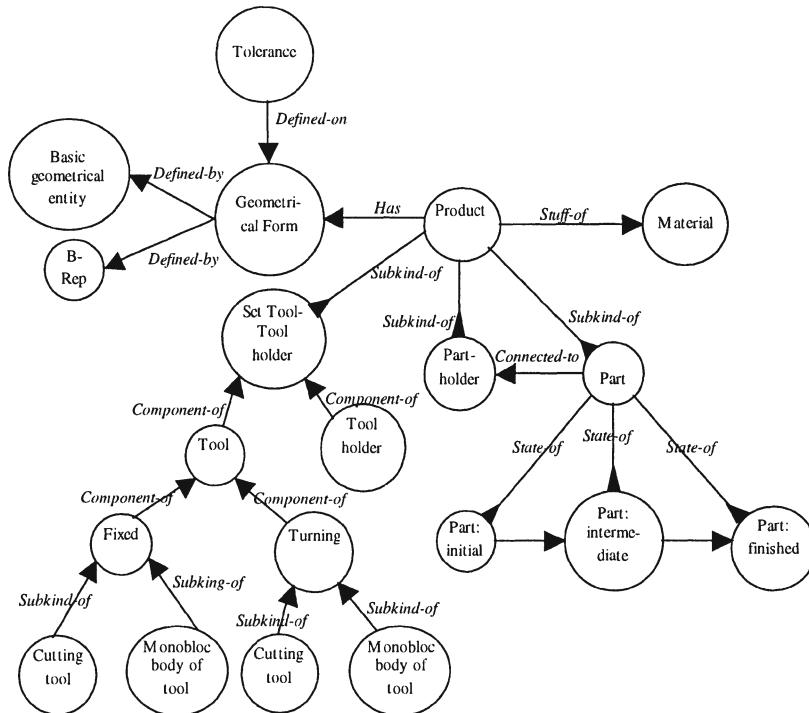


Figure 12. Product representation of Tool

The configuration of relationships between different products (product's structure) is not sufficient as a complete product information. The geometric representation of product is essential for engineering analysis. Figure 13 also shows the relationship between product structure (product-definition, product-definition-relationship) and product shape representations, tolerance and material models.

This ontology's schema is normalized, generic and more complete than the two specific view points. The CNIndustries viewpoint or TOOL viewpoint can be easily translated to this schema. The definition of each terms are clearly expressed and the mapping rules between each diagrams are defined. We have also stocked how and why each information is defined in this way.

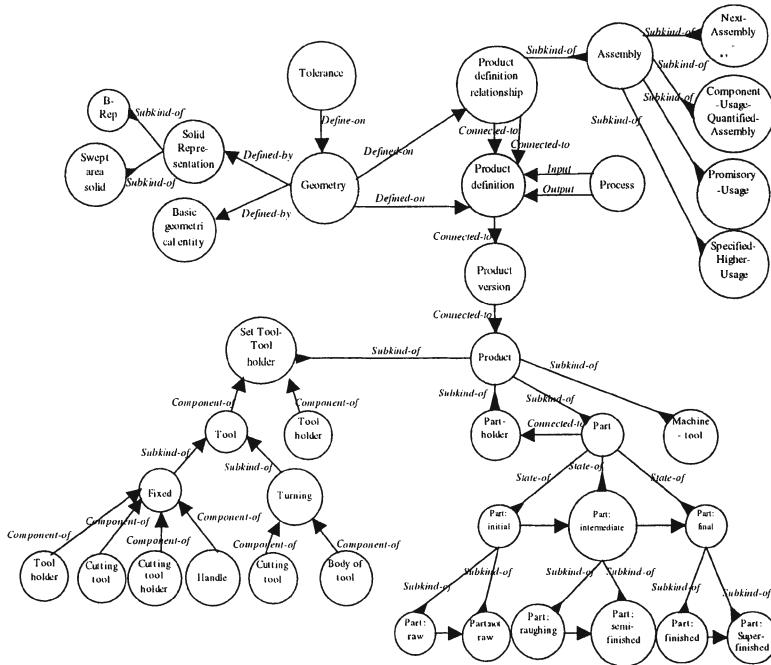


Figure 13. The generic product representation

#### 4.2 MACHINING RESSOURCE DESCRIPTION

The description of the machining resources that are used in the design and manufacturing processes are also studied. The viewpoint of CNIndustries is described in Figure 14.

We can see that the machining process depends on different resources such as machine-tool, Attachment, tool and part-holder. The machine-tool kind has different sub-kinds such as: lathe, bore and milling machine. Moreover, a part-holder fixes a part and an attachment fixes a tool.

If we refer to he Table 1 of the preceding subsection, we can see that CNIndustries defines different instances called Attachment to represent the tool holder. These instances are numbered from one to five. An attachment is connected to a tool, which is composed of an adaptator and a cutting-tool. Different kinds of tool exist: turning tool, boring tool and milling tool. Another kind defined in Figure 14 is the part-holder, which can be either stndard, specific or modular. We have also defined the machining resources

ontology for the Tool company, which is similar to the generic representation, Figure 15.

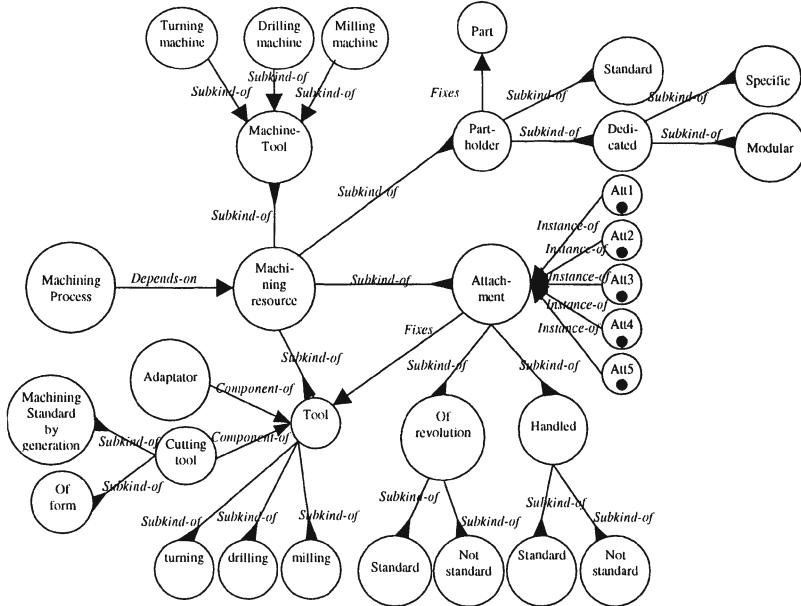


Figure 14. Representation of machining resources for CNIndustries

#### 4.3 PROCESS DESCRIPTION

The process description used by CNIndustries is represented in Figure 16. In this description, a machining process has different sub-kinds, such as boring, turning and milling. These concepts have some sub-kinds. For example, in the case of the milling, we have defined external turning, boring on a turning machine, internal turning , back turning and facial turning. Moreover, a procedure of process is associated with the machining process, and a procedure of operation is associated with any operation.

A machining process is composed of a set of operations that have some states (finished, roughing, ...). This process uses as input an initial part, which can be either a raw part or an intermediary part, and provides as output a final part, which is either an intermediary part or a finished part. Finally, each operation is realized by a tool.

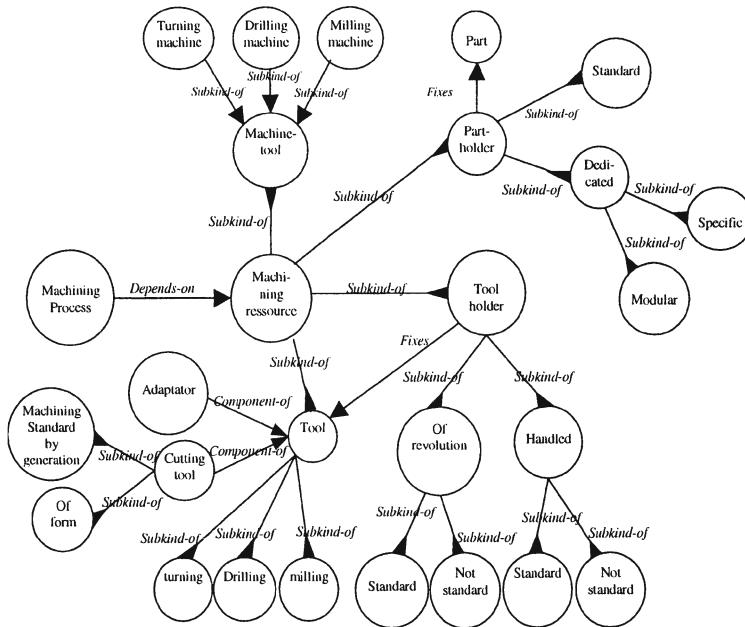


Figure 15. Generic representation of machining resources

The process description used by Tool is different and less complex, Figure 17. The equivalent of the machining process kind managed by CNIndustries is the machining operation by chip removal. This kind can have different sub-kinds, milling, boring and turning, which have also their own sub-kinds. Finally, the machining operation by chip removal depends on some environments (machine tools and lubricification).

The generic process description that we have defined is similar to the process description used by CNIndustries, Figure 18. The main difference is that the machining process uses some machine-tools and is defined by some procedures. Another difference concerns the operation kind, an operation is defined by a procedure.

## 5. Conclusions and Perspectives

The ability to fix a domain vocabulary and its meaning in the context is critical for efficient product data exchange and sharing. A large engineering or manufacturing project involves the resources of many different cooperative agents (human or otherwise) in the given endeavour. Each agent makes its own contributions, and the overall success of the project depends in large measure on the degree of integration between those different agents

throughout the development process. A key to effective integration is the accessibility of rich ontologies characterizing each of the domains addressed by each agent.

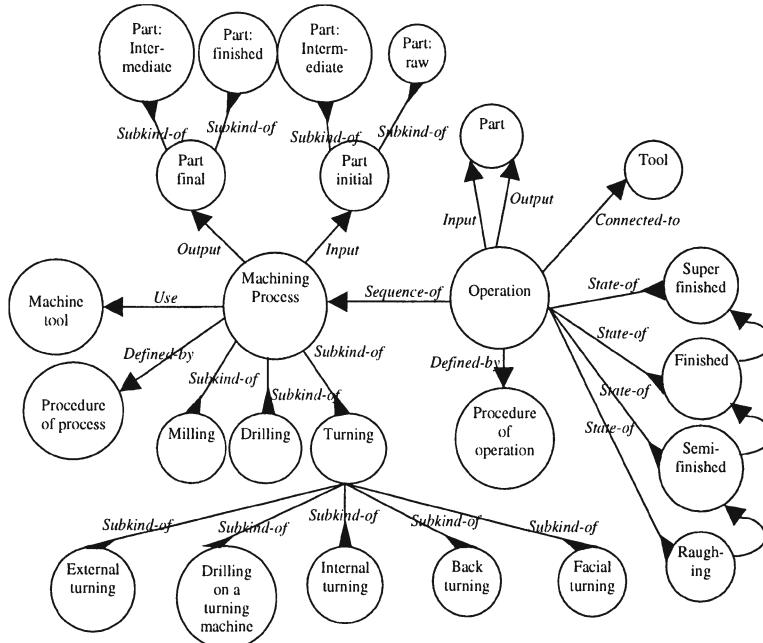


Figure 16. Process representation of CNIndustries

For instance, access to a manufacturing ontology that includes constraints on how a given part is manufactured can aid designers in their design of a complex product by giving them insight into the manufacturing implications of their design concepts. Similarly, access to an engineering ontology that includes constraints on how a design part is to function given a particular shape or fit can aid process planners in their development of the appropriate manufacturing processes. A commonly accessible collection of relevant ontologies thus permits more efficient sharing of information arising from various sources within the enterprise. The ontology capture permits the standardization of terminology, reusability and scalability.

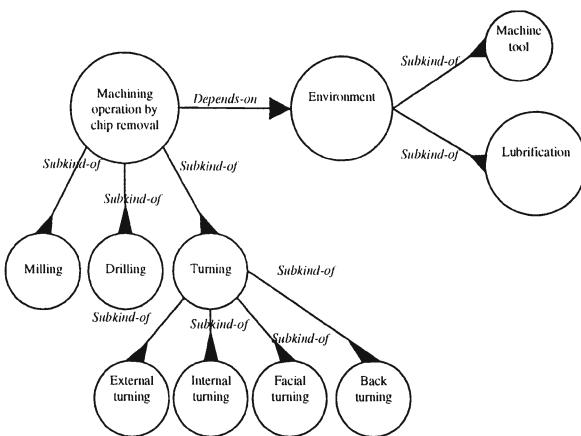


Figure 17. Process representation of Tool

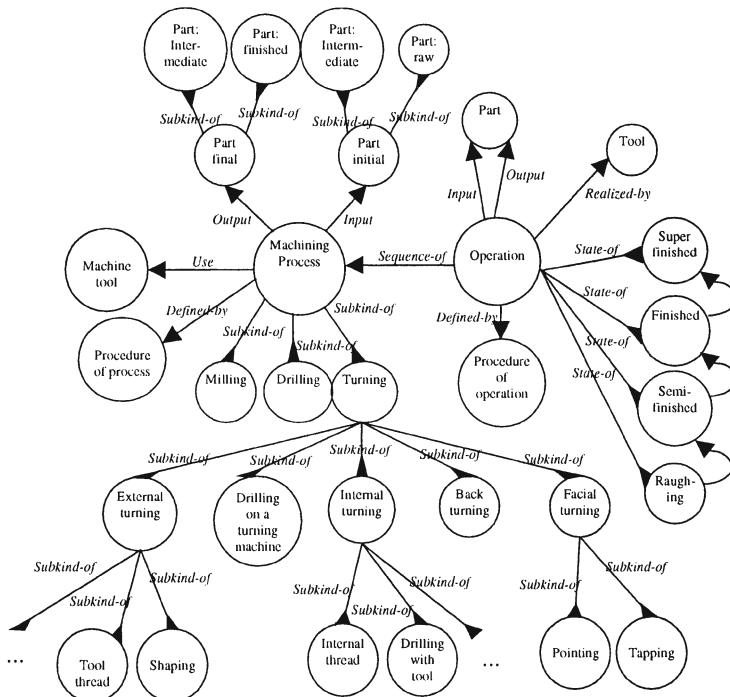


Figure 18. A generic process representation

In this paper we describe an approach which improves the data exchange by taking into account the semantics of data. This approach is based on ontologies, which allows the formalization of concepts related to product development in the cutting domain context. We have verified our approach for two commercial software tools in the framework of design and process planning data exchange. By this experimentation, we have arrived to this conclusion that the definition of a generic ontology is possible. Actually, we are developing a software which permits the ontology capture. With this system, each expert can define his own ontology or use the ontologies defined by the other experts, a help is also provided to define a generic ontology. For the future works we consider to complete our generic domain ontology by considering the tool parameters and we will also finish our software development in order to help the experts to define the semantics related to their data by ontologies.

### Acknowledgements

This work has been realized by the help of CNIndustries, TOOL company, Rhône-Alpes Region, CASM Laboratory (L. Deshayes and J.F. Rigal), Rémy Gervais and NIST (Pr. R. Sriram). We are very thankful for their helps and suggestions.

### References

- Brasethvik, T and Gulla, J: 2001, Natural language analysis for semantic document modeling, *Data and Knowledge Engineering* **38**: 45-62.
- Deshayes, L, Dartigues, C, Ghodous, P and Rigal JF: 2001, Distributed and standard system for cutting data management, *Proceedings of the 8th ISPE International Conference on Concurrent Engineering: Research and Applications (CE2001)*, pp. 310-317.
- Devedzic, V: 1999, A survey of modern knowledge modeling techniques, *Expert Systems with Applications* **17**: 275-194.
- Erdmann, M and Studer, R: 2001, How to structure and access XML documents with ontologies, *Data and Knowledge Engineering* **36**: 317-335.
- Fensel, D: 2001, *Ontologies: A Silver Bullet for Knowledge Management and Electronic Commerce*, Springer Verlag, Berlin.
- Fenves, SJ: 2001, A core product model for representing design information, *NIST Internal Report 6736*.
- Fernandez-Breis, JT, Martinez-Bejar, R; 2000, A cooperative tool for facilitating knowledge management, *Expert Systems with Applications* **18**: 315-330.
- Fox, MS and Gruninger, M: 1994, Ontologies for enterprise integration, *Proceedings of the 2nd Conference on Cooperative Information Systems*, CDROM.
- Gruber, TR: 2001, What is an ontology?, <http://www-ksl.stanford.edu/ksl/what-is-an-ontology.html>.
- Gruber, TR: Towards principles for the design of ontologies used for knowledge sharing, *Technical Report KSL93-04*, Knowledge Systems Laboratory, Stanford University.

- Guarino, N and Giaretta P: 1995, Ontologies and knowledge bases, towards a terminological clarification, <http://www.ladseb.pd.cnr.it/infor/Ontology/Papers/KBKS95.pdf>.
- Guarino, N and Welty C: 2000, Towards a methodology for ontology based model engineering, <http://www.ladseb.pd.cnr.it/infor/Ontology/Papers/ecoop-OO-ws.pdf>
- Huhns, MN and Singh MP: 1997, Ontologies for agents,  
<http://www.csc.ncsu.edu/faculty/mpsingh/papers/columns/aow-1-6-97.pdf>
- Perakath, CB, Menzel, CP, Mayer, R, Fillion, F, Futrell, MT, deWitte, PS and Lingineni M: 1994, *IDEF5 Method Report*, Knowledge Based Systems, Texas.
- Schlenoff, C, Gruninger, M and Ciocoiu, M: 1999, The essence of the process specification language, *Transactions of the Society for Computer Simulation* **16**(4): 204-216.
- Sriram, RD: 1999, Standards for the Collaborative Design Enterprise, Response to Object Management Group's (OMG), *Minutes of the MfgDTF Plenary* (Manufacturing Domain Task Force Plenary and Breakouts), San Jose, CA.
- Uschold, M and King, M: 1995, Towards a methodology for building ontologies, *Technical Report AIAI-TR-183*, Artificial Intelligence, Applications Institute, Edinburgh University.

## AUTHOR INDEX

- Achten, H, 153  
Agogino, AM, 305  
Alber, R, 329  
Badjgholi, F, 521  
Brazier, FMT, 503  
Caldas, L, 353  
Chan, KH, 383  
Clarkson, PJ, 479  
Culley, SJ, 431  
Darlington, M, 431  
Dartigues, C, 617  
Datta, S, 25  
Do, EY-L, 165  
Dong, A, 305  
Duffy, A, 261, 571  
Eckert, C, 479  
Felfernig, A, 41  
Fischer, M, 593  
Friedrich, G, 41  
Gero, JS, 89  
Ghodous, P, 617  
Hamilton Frazer, J, 215, 383  
Haymaker, J, 593  
Heylighen, A, 285  
Hill, AW, 305  
Holden, T, 237  
Ishida, R, 371  
Jannach, D, 41  
Jonker, CM, 547  
Kannengiesser, U, 89  
Kittl, B, 521  
Krishnamurti, R, 105  
Kunz, J, 593  
Lacroix, Z, 453  
Leclercq, P, 285  
Lenart, M, 65  
Lu, WF, 407  
Okazaki S, 131  
Pasztor, A, 65  
Pranovich, S, 153  
Rudolph, S, 329  
Serearuno, M, 237  
Shah, JJ, 453  
Shiose, T, 371  
Smith, JS, 571  
Smithers, T, 3  
Stouffs, R, 105  
Stumptner, M, 521  
Sugiura, N, 131  
Summers, JJ, 453  
Tang, MX, 215, 383  
Taura, T, 371  
Toppano, E, 191  
Treur, J, 547  
Van Wijk, JJ, 153  
Whitfield, RI, 571  
Wijngaards, NJE, 503  
Wijngaards, WCA, 547  
Woodbury, RF, 25  
Wu, Z, 261  
Xu, ZG, 215  
Zanker, M, 41  
Zha, XF, 407

## CONTACT AUTHORS' EMAIL ADDRESSES

|                   |                              |
|-------------------|------------------------------|
| Achten, H,        | h.h.achten@bwk.tue.nl        |
| Alber, R,         | alber@isd.uni-stuttgart.de   |
| Brazier, FMT,     | frances@nlnet.nl             |
| Caldas, L,        | lcaldas@mit.edu              |
| Chan, KH,         | sdkhchan@polyu.edu.hk        |
| Darlington, M,    | ensmj@bath.ac.uk             |
| Datta, S,         | sdatta@deakin.edu.au         |
| Do, EY-L,         | ellendo@u.washington.edu     |
| Dong, A,          | adong@me.berkeley.edu        |
| Eckert, C,        | cme26@eng.cam.ac.uk          |
| Ghodous, P,       | ghodous@bat710.univ-lyon1.fr |
| Haymaker, J,      | haymaker@stanford.edu        |
| Holden, T,        | holden@eng.cam.ac.uk         |
| Jannach, D,       | dietmar@ifit.uni-klu.ac.at   |
| Kannengiesser, U, | udo@arch.usyd.edu.au         |
| Leclercq, P,      | pierre.leclercq@ulg.ac.be    |
| Lenart, M,        | lenartm@cs.fiu.edu           |
| Shah, JJ,         | jami.shah@asu.edu            |
| Shiose, T,        | shiose@mech.kobe-u.ac.jp     |
| Smithers, T,      | tsmithers@vicomtech.es       |
| Stouffs, R,       | r.stouffs@bk.tudelft.nl      |
| Stumptner, M,     | mst@cs.unisa.edu.au          |
| Sugiura, N,       | hm.sugi@archi.kyoto-u.ac.jp  |
| Toppano, E,       | toppano@dimi.uniud.it        |
| Treur, J,         | treur@cs.vu.nl               |
| Whitfield, RI,    | ianw@cad.strath.ac.uk        |
| Wu, Z,            | chao@cad.strath.ac.uk        |
| Xu, ZG ,          | chris_xu@yahoo.com           |
| Zha, XF,          | xfzha@gintic.gov.sg          |