21 Mar 2002

KCAP K-CAP Conference Support

Yolanda Gil
University of Southern California

20020517 021

Air Force Research Laboratory
Air Force Office of Scientific Research
Arlington, Virginia

REPORT DOCUMENTATION PAGE

AFRL-SR-AR-TR-02-

0l05

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REF ..... AND DATES COVERED
		01 June 01 - 31 December 01

4. TITLE AND SUBTITLE	5. FUNDING NUMBERS
KCAP K-CAP Conference Support	F49620-01-1-0437

6. AUTHOR(S)

Yolanda Gil

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)	8. PERFORMING ORGANIZATION REPORT NUMBER
University of Southern California School of Engineering 4676 Admiralty Way, Suite 1001 Marina del Rey, CA 90292-6601	

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSORING/MONITORING AGENCY REPORT NUMBER
AFOSR/NM 801 N. Randolph Street Room 732 Arlington, VA 22203-1977	F49620-01-1-0437

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION AVAILABILITY STATEMENT	12b. DISTRIBUTION CODE
APPROVED FOR PUBLIC RELEASE	

13. ABSTRACT *(Maximum 200 words)*

SEE ATTACHED SHEET

14. SUBJECT TERMS			15. NUMBER OF PAGES
			209
			16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT

Abstract F49620-01-1-0437

This report includes the compiled papers from the 1st International Conference on Knowledge Capture held 21-23 Oct 2001 and sponsored by AFOSR.

In today's Web-linked and data-rich world, there is a growing need to manage burgeoning amounts of information effectively. Although indexing and linking documents and other information sources is an important step, capturing the knowledge contained within these diverse sources is crucial for the effective use of large information repositories. Knowledge acquisition has been a challenging area of research in artificial intelligence, with its roots in early work to develop expert systems. Driven by the modern Internet culture and by knowledge-based industries, the study of knowledge acquisition has a renewed importance.

Although there has been considerable work in the area of knowledge capture, activities have been distributed across several distinct research communities. In machine learning, learning apprentices acquire knowledge by non-intrusively watching a user perform a task. In the human-computer interaction community, programming-by-demonstration systems learn to perform a task by watching a user demonstrate how to accomplish it. In knowledge engineering, modeling techniques and design principles have been proposed for knowledge-based systems, often exploiting commonly occurring domain-independent inference structures and reusable domain-specific ontologies. In planning and process management, mixed-initiative systems acquire knowledge about a user's goals by taking commands or accepting advice regarding a task. In natural language processing, tools can process text and create representations of its knowledge content. All of these approaches are related in that they acquire information and organize it in knowledge structures that can be used for reasoning. They are complementary in that they use different techniques and approaches to capture different forms of knowledge.

The aim of k-CAP 2001 was to provide a forum in which to bring together disparate research communities whose members are interested in efficiently capturing knowledge from a variety of sources and in creating representations that can be (or eventually can be) useful for reasoning. This conference promoted multidisciplinary research that could result in a new generation of tools and methodologies for knowledge capture.

Topics presented included:

- Knowledge acquisition tools
- Advice taking systems
- Authoring tools
- Programming-by-demonstration systems
- Learning apprentices
- Knowledge engineering and modeling methodologies
- Knowledge extraction systems

- Knowledge management environments
- User preferences elicitation tools
- Mixed-initiative decision-support tools
- Knowledge-based markup techniques

KCAP
K-CAP Conference Support

Final Report
March 21, 2002

Period of Performance
July 2001-December 2001

Yolanda Gil
USC/Information Science Institute
4676 Admiralty Way, Suite 1001
Marina del Rey, CA 90292
(310) 822-1511
gil@isi.edu

Table of Contents

## Invited Speakers' Abstracts

Technical Papers

Figure 3. OntoDesigner.

filling the form that appears in the middle of the screen, and contextual menus arise when right-clicking on any of the visualized components.

This user interface also includes the functionalities of exporting/importing ontologies into XML or varied ontology languages, inference engine and documentation.

OntoDesigner. OntoDesigner is a graphical user interface for the visual construction of taxonomies of concepts and ad-hoc relations between concepts, which is integrated in the WebODE ontology editor as an applet. Figure 3 shows a snapshot of OntoDesigner while editing an ontology on the domain of office furniture.

Using OntoDesigner, the user can create different views of the edited ontology, so that the visualization of parts of the ontology can be customized while creating it. Moreover, the user can decide at any time whether showing or hiding different kinds of relations (either predefined or ad-hoc) between concepts, in the sense of a graphical prune.

Axiom Manager. This applet is used to ease the management of formulae in the WebODE ontology editor. It allows the user to create axioms using a graphical interface and provides functionalities such as an axiom library, axiom patterns and axiom parsing and verification.

5. RELATED WORK

WebODE has a strong relationship with ODE [3]. Both applications allow building ontologies at the knowledge level, and translators are used to implement them in different ontology languages. ODE was created as a classical application for single users and was difficult to extend. Furthermore, ontologies were stored in a Microsoft Access database, which proved to be inefficient when dealing with large ontologies. However, while ODE knowledge model is flexible, WebODE knowledge model is fixed, as has been explained in this paper.

Protégé2000 and OntoEdit are ontology development tools developed at the same time than WebODE, and using a similar design rationale, although they are not web-based but stand-alone applications. In fact, they share many functionalities (ontology edition, ontology documentation, ontology exportation and importation into XML and other languages). Moreover, Protégé2000 has been developed using a plug-in architecture, where new services can be added easily to the environment. However, WebODE integrates all its services in a well-defined architecture, stores its ontologies in a relational database (avoiding the use of text files) and provides additional services such as the inference engine, the axiom builder, ontology acquisition or catalogue generation.

OilEd was developed in the context of the OntoKnowledge [22] EU project for the easy development of OIL ontologies. It is not intended as a complete ontology editor, but just "the Notepad for OIL ontologies".

Other "classic" editors, such as WebOnto, Ontolingua and OntoSaurus, can be used for the edition of ontologies in a specific language (OCML, Ontolingua and LOOM, respectively). They do not use databases for storing ontologies.

6. CONCLUSIONS

In this paper, we have stated the need for a workbench for ontological engineering that allows:

- the development and management of ontologies,

- a wide use and integration of ontologies using a set of useful ontology middleware services, and

- the rapid development of ontology-based applications for their integration in enterprise information systems.

We have presented the WebODE workbench as a solution for this needs, describing its expressive knowledge model for representing ontologies, several built-in services and additional reusable services, such as WebPicker, OntoMerge and OntoCatalogue.

The Association for Computing Machinery
1515 Broadway
New York, New York 10036

Notice to Past Authors of ACM-Published Articles

ACM intends to create a complete electronic archive of all articles and/or other material previously published by ACM. If you have written a work that has been previously published by ACM in any journal or conference proceedings prior to 1978, or any SIG Newsletter at any time, and you do NOT want this work to appear in the ACM Digital Library, please inform permissions@acm.org, stating the title of the work, the author(s), and where and when published.

ACM ISBN: 1-58113-380-4

Additional copies may be ordered prepaid from:

ACM Order Department
PO Box 11405
New York, NY 10286-1405

Phone: 1-800-342-6626
(US and Canada)
+1-212-626-0500
(all other countries)
Fax: +1-212-944-1318
E-mail: acmhelp@acm.org

ACM Order Number 607010
Printed in the USA

Cover Design
by Fanny Mak

FOREWORD

In today's Web-linked and data-rich world, there is a growing need to manage burgeoning amounts of information effectively. Although indexing and linking documents and other information sources is an important step, capturing the knowledge contained within these diverse sources is crucial for the effective use of large information repositories. Knowledge acquisition has been a challenging area of research in artificial intelligence, with its roots in early work to develop expert systems. Driven by the modern Internet culture and by knowledge-based industries, the study of knowledge acquisition has a renewed importance.

Although there has been considerable work in the area of knowledge capture, activities have been distributed across several distinct research communities. In machine learning, learning apprentices acquire knowledge by nonintrusively watching a user perform a task. In the human-computer interaction community, programming-by-demonstration systems learn to perform a task by watching a user demonstrate how to accomplish it. In knowledge engineering, modeling techniques and design principles have been proposed for knowledge-based systems, often exploiting commonly occurring domain-independent inference structures and reusable domain-specific ontologies. In planning and process management, mixed-initiative systems acquire knowledge about a user's goals by taking commands or accepting advice regarding a task. In natural language processing, tools can process text and create representations of its knowledge content. All of these approaches are related in that they acquire information and organize it in knowledge structures that can be used for reasoning. They are complementary in that they use different techniques and approaches to capture different forms of knowledge.

The aim of *K-CAP 2001* is to provide a forum in which to bring together disparate research communities whose members are interested in efficiently capturing knowledge from a variety of sources and in creating representations that can be (or eventually can be) useful for reasoning. This new conference will promote multidisciplinary research that could result in a new generation of tools and methodologies for knowledge capture. The twenty six papers included in these proceedings cover many important topics for the conference, including ontologies/knowledge representation, interactive acquisition tools, collaborative/distributed KA, information extraction, knowledge management, semantic markup, adaptive user interfaces, PSMs, and learning from examples. The papers were selected from eighty-two submissions. Our program committee members are largely responsible for the high quality of the conference program.

Our invited speakers John McCarthy, Ken Forbus, and Steve Lawrence, covered diverse topics of interest to this community and an abstract of their talks is included in these proceedings. Three invited tutorials were held prior to the main conference program. Frank van Harmelen, Dieter Fensel, and Heiner Stuckenschmidt talked about the Semantic Web, Henry Lieberman presented programming by example, and Guus Schreiber and Hans Akkermans gave an overview of CommonKADS. Dieter Fensel organized an excellent workshop program.
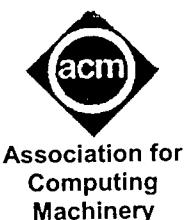
We would also like to thank other people who contributed to making this conference happen. John Gennari handled our finances and coordinated with ACM's sponsorship program. Rob Kremer proposed a wonderful location for the conference, the relaxing Laurel Point Inn, and handled the local arrangements together with Tim Menzies. Rob also managed the conference Web site and the registrations, and brought in Camille Sinanan and Roberto Flores to help with registrations and student volonteers respectively. Fanny Mak designed the logo on the cover of the proceedings, as well as many fliers and posters to advertise the conference. Lisa Tolles-Efinger at Sheridan Printing took over the production of the proceedings for publication by ACM. Jim Blythe, Pete Clark, John Gennari, and Xian-ping Ge helped create the Latex templates for the papers.

Finally, we would like to thank the conference sponsors for their financial and administrative support to the conference.

Yolanda Gil
Mark Musen
Jude Shavlik

K-CAP 2001 Sponsoring Organizations
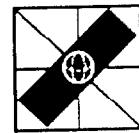
K-CAP 2001 is sponsored by the Association for Computing Machinery (ACM) and its Special Interest Group on Artificial Intelligence (SIGART), held in cooperation with the American Association for Artificial Intelligence (AAAI), and is an endorsed conference of Technical Committee 12 of the International Federation for Information Processing (IFIP).

Association for Computing Machinery

American Association for Artificial Intelligence

International Federation for Information Processing

US Air Force Office of Scientific Research

US Defence Advanced Research Project Agency

US Office of Naval Research

US Army Simulation, Training, and Instrumentation Command

Table of Contents

Invited Speakers' Abstracts

Technical Papers

K-CAP'01 Conference Organizers

Conference Co-Chairs: Yolanda Gil, *University of Southern California/Information Sciences Institute*

Mark Musen, *Stanford University*

Jude Shavlik, *University of Wisconsin at Madison*

Treasurer: John Gennari, *University of California at Irvine*

Local Arrangements: Rob Kremer, *University of Calgary*

Tim Menzies, *University of British Columbia*

Workshop Chair: Dieter Fensel, *Free University of Amsterdam*

Program Committee: Hans Akkermans, *Free University of Amsterdam*

Jeff Bradshaw, *University of West Florida*

Bruce Buchanan, *University of Pittsburgh*

Claire Cardie, *Cornell University*

B. Chandrasekaran, *Ohio State University*

Steve Chien, *Jet Propulsion Laboratory*

Paul Compton, *University of New South Wales*

Mark Craven, *University of Wisconsin at Madison*

Rose Dieng, *INRIA-Sophia-Antipolis*

Adam Farquhar, *Schlumberger*

Dieter Fensel, *Free University of Amsterdam*

Richard Fikes, *Stanford University*

Ken Forbus, *Northwestern University*

Peter Haddawy, *Asian Institute of Technology*

Eric Horvitz, *Microsoft Research*

Henry Kautz, *University of Washington*

Pat Langley, *Daimler-Benz Research*

Doug Lenat, *Cycorp*

Henry Lieberman, *MIT Media Lab*

Steve Minton, *Fetch Technologies*

Ray Mooney, *University of Texas at Austin*

Johanna Moore, *University of Edinburgh*

Enrico Motta, *Open University*

Karen Myers, *SRI International*

Dan O'Leary, *University of Southern California*

David Page, *University of Wisconsin at Madison*

Bruce Porter, *University of Texas at Austin*

Program Committee (continued): Charles Rich, *Mitsubishi Electric Research Laboratory*
Claude Sammut, *University of New South Wales*
Guus Schreiber, *University of Amsterdam*
Nigel Shadbolt, *University of Southampton*
Rudi Studer, *University of Karlsruhe*
Bill Swartout, *USC/Institute for Creative Technologies*
Loren Terveen, *AT&T Labs*
Manuela Veloso, *Carnegie Mellon University*
David C. Wilkins, *University of Illinois at Urbana-Champaign*
Ian Witten, *University of Waikato*

Additional Reviewers: Harith Alani, *University of Southampton*
Leslie Carr, *University of Southampton*
Srinandan Dasmahapatra, *University of Southampton*
John Domingue, *Open University*
James Fan, *University of Texas at Austin*
Eibe Frank, *University of Waikato*
Andy Garland, *Mitsubishi Electric Research Lab*
Nick Gibbins, *University of Southampton*
Siegfried Handschuh, *University of Karlsruhe*
Laurie Hiyakumoto, *Carnegie Mellon University*
Geoff Holmes, *University of Waikato*
Yannis Kalfoglou, *Open University*
Jihie Kim, *USC/Information Sciences Institute*
Fritz Lehmann, *Cycorp*
Alexander Maedche, *University of Karlsruhe*
Tim Menzies, *University of British Columbia*
Kieron O'Hara, *University of Southampton*
Bernhard Pfahringer, *University of Waikato*
Tom Russ, *USC/Information Sciences Institute*
Gerd Stumme, *University of Karlsruhe*
York Sure, *University of Karlsruhe*
Dan Tecuci, *University of Texas at Austin*
Maria Vargas-Vera, *Open University*
Peter Z. Yeh, *University of Texas at Austin*

Invited Tutorials

The Semantic Web
Sunday, 08:30-11:20

Frank van Harmelen, Free University of Amsterdam
Dieter Fensel, Free University of Amsterdam
Heiner Stuckenschmidt, Universität Bremen

The Semantic Web is the vision of having data on the Web defined and linked in a way that it can be used by machines not just for display purposes, but for automation, integration and reuse of data across various applications.

In this tutorial, we will discuss a number of key standards, technologies and policies that are currently being designed by leading European and American academic and industrial groups under the supervision of the W3C in order to make this vision a reality.

Programming by Example: Intelligent Interfaces for Teaching New Behavior to a Machine
Sunday, 11:30-15:20

Henry Lieberman, Media Lab, Massachusetts Institute of Technology
http://www.media.mit.edu/~lieber/

Programming by Example (also called Programming by Demonstration) is a powerful new technology that lets end-users create programs by recording actions in the user interface rather than by typing statements in a programming language. The user demonstrates a sequence of actions on a concrete example in a graphical user interface, and the system records the actions. Machine Learning and agent technologies are used to generalize programs that can be used in future situations that are analogous to, but not the same as, the situation on which the system was first taught. Programming by Example systems are "macros on steroids."

This tutorial will present this technology, which shows how intelligent user interfaces can dramatically improve the process of software development and make it accessible to users who do not have prior experience with programming. The ideas are, of course, best presented by example. We will survey many systems of this type, including live demonstrations. We will also do in-class design exercises, such as "Wizard of Oz" and "Short-Order Programming" exercises to give attendees hands-on experience with the technology.

Henry Lieberman is the editor of a new book, "Your Wish is my Command," published by Morgan Kauffman, which will serve as text for the tutorial. This book collects 19 articles which describe PBE systems for such diverse applications as text editing, graphical editing, CAD/CAM, animation, games, web browsing, teaching children programming, and others. He also maintains the Programming by Example Web site, at http://www.media.mit.edu/~lieber/PBE/. This site also contains the book, "Watch What I Do," Allen Cypher, ed. the other major reference in this field.

Knowledge Engineering with CommonKADS in perspective
Sunday, 15:50-18:10

Guus Schreiber, University of Amsterdam
Hans Akkermans, Free University of Amsterdam

In this tutorial we review what we perceive as the main contributions of the CommonKADS methodology for knowledge engineering. We give a synopsis of CommonKADS and discuss some topics in more depth, such as reusable task models, rule types, context models, and the link to object-oriented analysis and design. The tutorial is partly based on the experiences gained by the speakers in teaching CommonKADS to non-AI audiences.

Guus Schreiber is an associate professor of Social Science Informatics at the University of Amsterdam. His research interests lie mainly in the area of knowledge engineering and knowledge-system development. Recently he has also worked on knowledge modelling for the "semantic web". He published more than 80 papers in these areas. In 2000 he published with MIT Press a textbook on knowledge engineering and knowledge management, based on the CommonKADS methodology. He has been involved in numerous European research projects in the knowledge-engineering area, such as KADS & KADS-II (both on methodologies for knowledge-system development), REFLECT (reflective reasoning), GAMES (medical knowledge systems) and KACTUS (technical ontologies). He is currently involved in the Dutch ICES project MIA (Multimedia Information Amalysis) and in the European project IBROW (Intelligent Brokering on the Web).

Hans Akkermans is a professor of Business Informatics at the Free University Amsterdam, and an international consultant in information and knowledge management. He holds a cum laude PhD degree in theoretical physics from the University of Groningen. He has published over 125 refereed international journal articles and conference papers in various fields of informatics, physics, engineering, and e-business, and participates in many international cooperative industry-university projects. He regularly carries out evaluation and advisory assignments for the European Commission. His group at Amsterdam coordinates OntoWeb, the EU Thematic Network on Ontology and Semantic Web Technology for Knowledge Management and E-Commerce.

Invited Speakers' Abstracts

Knowledge Capture for Bootstrapping Intelligent Systems

Kenneth D. Forbus
Northwestern University
forbus@northwestern.edu

Abstract

Knowledge is the fuel for intelligent systems. Building software that comes closer to the breadth and flexibility of human reasoning will require substantially larger knowledge bases than any that have been built to date. This talk describes two ways we are exploring for bootstrapping intelligent systems via knowledge capture. First, analogy is useful for knowledge capture because people find it easier to articulate examples than universally valid principles. By exploiting recent advances in cognitive science, we are creating a technology of analogical processing that can (and has) been used with multiple large knowledge bases. Second, sketching is useful for knowledge capture because people find it easier to express many things spatially. By focusing on deeper visual and conceptual understanding of the contents of sketching, instead of recognition, we are building sketching systems that can be broadly applied in many domains. These efforts have benefited substantially from involvement with DARPA research communities, and some lessons learned from those experiences will be discussed.

Bio

Kenneth D. Forbus is a Professor of Computer Science and Education at Northwestern University. His research interests include qualitatitve reasoning, analogy and similarity, sketching and spatial reasoning, cognitive simulation, reasoning system design, articulate educational software, and the use of AI in computer gaming. He received his degrees from MIT (Ph.D. in 1984). He is a Fellow of the American Association for Artificial Intelligence and serves on the Governing Board of the Cognitive Science Society and the editorial boards of Artificial Intelligence, Cognitive Science, and the AAAI Press.

ResearchIndex: Inside the World's Largest Free Full-Text Index of Scientific Literature

Steve Lawrence
NEC Research Institute
lawrence@research.nj.nec.com

Abstract

ResearchIndex (also known as CiteSeer) is a digital library of scientific literature that aims to improve communication and progress in science. This talk covers the design, implementation, and operation of ResearchIndex.

Bio

Dr. Steve Lawrence is a Research Scientist at NEC Research Institute, Princeton, NJ. His research interests include information retrieval and machine learning. Dr. Lawrence has published over 50 papers in these areas, including articles in Science, Nature, CACM, and IEEE Computer. He has been interviewed by over 100 news organizations including the New York Times, Wall Street Journal, Washington Post, Reuters, Associated Press, CNN, MSNBC, BBC, and NPR. Hundreds of articles about his research have appeared worldwide in over 10 different languages.

Phenomenal Data-Mining

John McCarthy
Computer Science Department, Stanford University
jmc@cs.stanford.edu

Phenomenal data mining finds relations between the data and the *phenomena* that give rise to data rather than just relations among the data.

Science and common sense both tell us that the facts about the world are not directly observable but can be inferred from observations about the effects of actions. What people infer about the world is not just relations among observations but relations among entities that are much more stable than observations. For example, 3-dimensional objects are more stable than the image on a person's retina, the information directly obtained from feeling an object or on an image scanned into a computer.

This talk concerns what can be inferred by programs about phenomena from data and what facts are relevant to doing this. In order to infer phenomena from data, facts about their relations must be supplied. Sometimes these facts can be implicit in the programs that look for the phenomena, but more generality is achieved if the facts are represented as sentences of logic in a *knowledge base* used by the programs.

Creating knowledge bases containing both common sense knowledge and knowledge of the domain of the data will be a lot of work. This is unavoidable.

The result of phenomenal data-mining can include an extended database with additional fields on existing relations and new relations. Thus the relations describing supermarket baskets can be extended with a customer field, and new relations about customers and their properties can be introduced.

Bio
John McCarthy is Professor Emeritus of Computer Science at Stanford University. His recent research addresses various aspects of formal reasoning, including non-monotonic logic, commonsense reasoning, formalizing context, and elaboration tolerance. He was first to propose time-sharing computer systems and to develop the Lisp programming language. McCarthy has worked on Artificial Intelligence since its early days and is credited with coining the term. He McCarthy received the A. M. Turing award of the Association for Computing Machinery in 1971 and was elected President of the American Association for Artificial Intelligence for 1983-84 and is a Fellow of that organization. He is a member of the American Academy of Arts and Sciences, the National Academy of Engineering and the National Academy of Sciences. He received the National Medal of Science in 1990.

Technical Papers

K-CAP 2001

WebODE: a Scalable Workbench for Ontological Engineering

Julio C. Arpírez, Oscar Corcho, Mariano Fernández-López, Asunción Gómez-Pérez

Facultad de Informática. Universidad Politécnica de Madrid

Campus de Montegancedo, s/n. 28660 Boadilla del Monte. Madrid. Spain

+34 91 336 7439

jarpirez@delicias.dia.fi.upm.es; {ocorcho, mfernandez, asun}@fi.upm.es

ABSTRACT

This paper presents WebODE as a workbench for ontological engineering that not only allows the collaborative edition of ontologies at the knowledge level, but also provides a scalable architecture for the development of other ontology development tools and ontology-based applications. First, we will describe the knowledge model of WebODE, which has been mainly extracted and improved from the reference model of METHONTOLOGY's intermediate representations. Later, we will present its architecture, together with the main functionalities of the WebODE ontology editor, such as its import/export service, translation services, ontology browser, inference engine and axiom generator, and some services that have been integrated in the workbench: WebPicker, OntoMerge and the OntoCatalogue.

Keywords

WebODE, ontology engineering workbench, ontology building, translation, integration and merge.

1. INTRODUCTION

In the last years, several tools for building ontologies have been developed: Ontolingua [11], OntoSaurus [24], WebOnto [8], Protégé2000 [25], OilEd [20], OntoEdit [21], etc. A study comparing some of them can be found in [10]. Additional ontology tools and services have been built for other purposes: ontology merging (Chimaera [18], Ontomorph [4], PROMPT [14]), ontology access (OKBC [5]), etc. Finally, many applications have been built upon ontologies: Ontobroker [12], PlanetOnto [9], (KA)^2 [1], MKBEEM [19], etc. All these tools and applications have contributed to a high development of the ontology community, and have laid the foundations of an emergent research and technological area: the Semantic Web [2].

However, current ontological technology suffers from the following problems, which must be solved prior to its transfer to the enterprise world:

- There is no correspondence between existing methodologies and environments for building ontologies, except ODE and METHONTOLOGY [13].

- Existing environments just give support for designing and implementing ontologies, but they do not support all the activities of the ontology life cycle.

- There are a lot of isolated ontology development tools that cannot interoperate easily, because they are based on different technologies, on different knowledge models for representing ontologies, etc.

Consequently, there is a need for a common workbench to ensure a wide acceptance and use of ontological technology. We foresee three main areas in this workbench, as shown in figure 1:

- Ontology development and management, which comprises technology that gives support to ontology development activities: knowledge acquisition, edition, browsing, integration, merging, reengineering, evaluation, implementation, etc.; ontology management activities: configuration management, ontology evolution, ontology libraries, etc.; and ontology support activities: scheduling, documentation, etc.

- Ontology middleware services, which include different kinds of services that will allow the easy use and integration of ontological technology into existing and future information systems, such as services for accessing ontologies, integration with databases, ontology upgrading, query services, etc.

- Ontology-based applications development suites, which will allow the rapid development and integration of ontology-based applications. They will be the last step towards a real integration of ontologies into enterprise information systems.

In this paper, we will present WebODE as an scalable ontological engineering workbench that gives support to activities from the first two areas of the workbench previously identified. WebODE's ontology editor allows the collaborative edition of ontologies at the knowledge level, supporting the conceptualization phase of METHONTOLOGY and most of the activities of the ontology's life cycle (reengineering, conceptualization,

Figure 1. An ontological engineering workbench.

implementation, etc). Besides, WebODE provides high extensibility in an application server basis, allowing the creation of middleware services that will allow the use of ontologies from applications.

This paper is organized as follows: section *WebODE in a nutshell* gives a general overview of the main features of this ontological engineering workbench. Section *WebODE's knowledge model* presents the knowledge model used for representing ontologies in the WebODE workbench. Section *WebODE architecture* describes its main services and the WebODE ontology editor, as an applications that uses most of the services. Section *Related Work* gives a short overview of existing ontology editing applications. Finally, the *Conclusions* section summarizes the main conclusions of this work, projects in which WebODE has already been used, ontologies developed using the WebODE ontology editor and further work.

2. WEBODE IN A NUTSHELL

WebODE is not an isolated tool for the development of ontologies, but an advanced ontological engineering workbench that provides varied ontology related services and covers and gives support to most of the activities involved in the ontology development process.

WebODE workbench is built on an application server basis, which provides high extensibility and usability by allowing the addition of new services and the use of existing services. Examples of these services are *WebPicker*, *OntoMerge* and *OntoCatalogue*.

Ontologies in WebODE are stored in a relational database. Moreover, WebODE provides a well-defined service-oriented API for ontology access that makes it easy the integration with other systems.

Ontologies built with WebODE can be easily integrated with other systems by using its automatic exportation and importation services from and into XML, and its translation services into and from varied ontology specification languages (currently, RDF(S) [23], OIL [16], DAML+OIL [7], X-CARIN [19] and FLogic [17]).

WebODE's ontology editor allows the collaborative edition of ontologies at the knowledge level. Its knowledge model, which is described in depth in the next section, is mainly based on the set of intermediate representations of METHONTOLOGY and provides additional features.

Ontology edition is aided both by form based and graphical user interfaces, a user-defined-views manager, a consistency checker, an inference engine, an axiom builder and the documentation service.

Two interesting and novel features of WebODE with respect to other ontology engineering tools are: instance sets, which allow to instantiate the same conceptual model for different scenarios, and conceptual views from the same conceptual model, which allow creating and storing different parts of the ontology, highlighting and/or customizing the visualization of the ontology for each user.

The graphical user interface allows browsing all the relationships defined on the ontology as well as graphical-pruning these views with respect to selected types of relationships. Mathematical properties such as reflexive, symmetric, etc. and other user-defined properties can be also attached to the "ad hoc" relationships.

The collaborative edition of ontologies is ensured by a mechanism that allows users to establish the type of access of the ontologies developed, through the notion of groups of users. Synchronization mechanisms also exist that allow several users to edit the same ontology without errors.

7

Constraint checking capabilities are also provided for type constraints, numerical values constraints, cardinality constraints and taxonomic consistency verification [15] (i.e., common instances of disjoint classes, loops, etc.)

Finally, WebODE's inference service has been developed in Ciao Prolog. Although WebODE is not OKBC compliant yet, all the OKBC primitives have been defined in prolog for their use in its inference engine.

3. WEBODE'S KNOWLEDGE MODEL

WebODE's knowledge model is extracted from the set of intermediate representations of METHONTOLOGY. It allows the representation of concepts and their attributes (both class and instance attributes), taxonomies of concepts, disjoint and exhaustive class partitions, ad-hoc binary relations between concepts, properties of relations, constants, axioms and instances. It also allows the inclusion of bibliographic references for any of them and the importation of terms from other ontologies.

Additionally, WebODE improves the reusability of ontologies defining sets of instances, which allow the instantiation of the same conceptual model for different scenarios it may be used for.

In the following subsections we will describe each one of the components of the *WebODE's* knowledge model:

3.1. Concepts

In short, a concept (also known as a class) can be anything about which something is said, and, therefore, can also be the description of a task, function, action, strategy, reasoning process, etc.

Concepts are identified by their **name**, although they can also have **synonyms** and **abbreviations** attached to them. A natural language (NL) **description** can be also included.

The same applies to **references** and **formulae**, which will be described later in this section. Any component in WebODE may have any amount of references and reasoning formulae attached to it.

Class attributes are attributes whose value must be the same for all instances of the concept. They are not components themselves in WebODE's knowledge model, as they are always attached to a concept (and to its subclasses, because of the inheritance mechanism).

The information stored for a class attribute is the following: its **name** (which must be different from the rest of attribute names of the same concept); the **name of the concept** it belongs to (attributes are local to concepts, that is, two different concepts can have attributes with the same name); its **value type**, also called range, which can be a basic data type (String, Integer, Cardinal, Float, Boolean, Date, Numeric Range, Enumerated, URL) or an instance of a concept (in this case, the name of the concept must be specified), and, finally, its **minimum and maximum cardinality**, which constrains the number of values that the class attribute may have.

Optional information for class attributes consists of its **NL**

description, the **measurement unit** and its **precision** (the last two ones just in case of numeric attributes).

Finally, the **value(s)** of the class attribute can be specified once it has been defined completely. These values will be attached to the class attribute where they have been defined.

Instance attributes are attributes whose value may be different for each instance of the concept. They have the same properties than class attributes and two additional properties, **minimum value** and **maximum value**, which are used in attributes with numeric value types. Values inserted for instance attributes are interpreted as default values for them.

3.2. Groups

Groups, also called partitions, are used to create disjoint and exhaustive class partitions. They are sets of disjoint concepts that have a **name**, the **set of concepts** they group together and, optionally, a **NL description**. A concept can belong to several groups.

3.3. Built-in Relations

This subsection deals with predefined relations in the WebODE's knowledge model, related to the representation of taxonomies of concepts and mereology relationships between concepts. They are divided into three groups:

Taxonomical relations between concepts. Two predefined relations are included: *subclass-of* and *not-subclass-of*. Single and multiple inheritance are allowed.

Taxonomical relations between groups and concepts. A group is a set of disjoint concepts. There are two predefined relations available, whose semantics is also explained:

- *Disjoint-subclass-partition*. A disjoint subclass partition Y of class X defines the set Y of disjoint classes as subclasses of class X. This classification is not necessarily complete: there may be instances of X that are not included in any subclass of the partition.

- *Exhaustive-subclass-partition*. An exhaustive subclass partition Y of class X defines the set Y of disjoint subclasses as subclasses of the class X, where X can be defined as union of all the classes of the partition

Mereological relations between concepts. Two relations are included: *transitive-part-of* and *intransitive-part-of*.

3.4. Ad-hoc relations

WebODE allows just binary ad-hoc relations to be created between concepts. The creation of relations of higher arity must be made by reification (creating a concept for the relation itself and *n* binary relations between the concepts that appear in the relation and the concept that is used for representing the relation).

Ad-hoc relations are characterized by their **name**, the name of the **origin** (source) and **destination** (target) concepts, and its **cardinality**, which establishes the number of facts (instances of the relation) that can hold between the origin and the destination term. Their cardinality can be restricted to 1 (only one fact) or N (any number of facts).

Additionally, there is some optional information that can be provided for an ad-hoc relation, such as its **NL description** and its **properties** (they are used to describe algebraic properties of the relation).

References and formulae can be also attached to the ad-hoc relations, as happened with the concepts.

3.5. Constants
Constants are components that have always the same value. They are included in the knowledge model of WebODE to ease the maintenance of ontologies. They are available for their use in any expression in the ontology.

The information needed for a constant is: **name, value type** (the same as shown for attributes of concepts, except for instances of concepts), **value** and **measurement unit**. Its **NL description** can be optionally provided.

3.6. Formulae
There are three types of formulae that can be created in WebODE: axioms, rules and procedures. All of them are represented by their **name**, an optional **NL description** and a **formal expression** in first order logic, using a syntax provided by WebODE.

Axioms model sentences, using first order logic, that are always true. They may be included in the ontologies for several purposes, such as constraining its information, verifying its correctness or deducting new information.

Rules are included in the ontology for the inference of new knowledge in the ontology from the knowledge already included in it. Their chaining mechanism is not explicitly declared, although WebODE's inference engine uses backward chaining.

Procedures are used for declaring sequences of actions. Currently, the user is free to use any syntax for these components, because it is too much tight to the target language in which the ontology will be used.

The axiom generator, which will be described later in this paper, allows the user create axioms more easily than if they were created from scratch. WebODE also provides a library of axiom patterns for common used expressions.

3.7. Instances
There are two kinds of instances that can be created in WebODE: instances of concepts and instances of relations (also called facts).

Instances of concepts represent elements of a given concept. They have their own **name**, and a set of **instance attributes** with their **values**. Instance attributes are inherited from the concept they belong to and its superclasses.

Instances of relations are used to represent a relation that holds between individuals (instances of concepts) in the ontology. They have their own **name**, the names of the **relation** and the **instances** that participate in it.

WebODE allows grouping both kinds of instances in sets of instances. **Instance sets**, which are described by their **name**

and an optional **description**, allow the distributed use of the ontology in different frameworks. In other words, the same ontology can be instantiated for different applications, and instances in an instance set are independent from instances in another one. This, along with the import/export features, permits the isolation of the main two parts of an ontology: its conceptualization and its instances (the knowledge base).

3.8. References
References are used for adding bibliographic references in the ontology. The information needed for references is their **name** and an optional **description**. They can be attached to any component of the WebODE's knowledge model.

3.9. Properties
They are used to describe algebraic properties of ad-hoc relations. They are divided in two groups:

- **Built-in properties**: reflexive, irreflexive, symmetric, asymmetric, antisymmetric and transitive.

- **Ad-hoc properties**. The user can define them and attach them to ad-hoc binary relations to describe either algebraic or other kinds of properties of them.

3.10. Imported terms
Imported terms are components that are included in the current ontology from other ontologies. The user must provide their **name**, the **host** for retrieving the term from, the name of the **ontology** where to retrieve the term from and the **original term name**.

Currently, only concepts from other ontologies can be imported into WebODE. In the future, this will be expanded to any kind of components of the ontology (groups, relations, axioms, etc.).

4. WEBODE ARCHITECTURE
The architecture of the WebODE workbench is explained in this section, according to the classical three tiers architecture commonly found in web applications: data tier, business logic tier and presentation tier.

4.1. Data Tier
Ontologies are the central element in our workbench. They can be stored in a relational database with JDBC support (it has been tested both in Oracle and MySQL).

The module for database access is included as a core service inside the Minerva Application Server (which is explained later in this section). Its main features are the optimization of connections to the database (*connection pooling*) and transparent *fault tolerance* capabilities.

4.2. Business Logic Tier
This tier usually is divided in two different ones: the presentation sub-tier and the logic sub-tier.

The **presentation sub-tier** is responsible for generating the content to be presented in the user's browser. It also handles user requests from the client (form handling, queries, etc.) and forward them to business logic services. *Servlets and* JSPs (*Java Server Pages*) are used in it.

The **logic sub-tier** comprises the applications' business-logic services. All the implemented services are available from the Minerva Application Server, through RMI-IIOP technology. We distinguish two groups of services: services from the Minerva Application Server, which are not tied specifically to the WebODE workbench but can be used by any other service, and business-logic services for WebODE, which are specific to this workbench.

Modules from Minerva Application Server
This application server has been developed in our lab. In this subsection we will describe its main modules:

Authentication module. All the authentication and security controls in the application server are based on this module. It allows managing access control lists for all the services of applications built upon the server, groups for sharing ontologies, information protection, etc.

Currently, it uses an internal format for storing and accessing information. However, it is possible to develop additional modules for the authentication of users using other authentication systems (from Windows NT, UNIX, LDAP, etc.). This would allow the integration of the workbench in the authentication schema of the organization.

Log module. This module is in charge of auditing tasks. Its verbose level can be configured, depending on the audit needs for the system.

Administration module. It allows the administration of the application server by using the *Minerva Management Console (MMC)*, which allows the server administrator to manage locally or remotely every installed service, to start and stop services, to manage users, groups and access control lists through the authenticator service, etc.

Thread management module. This module optimizes the use of threads in the server for any task, using thread-pooling techniques, which improve drastically their execution time. Additionally, it is possible to change thread priorities: some tasks can be executed before other ones.

Planning module. This module, which depends on the thread management module, allows the planning of periodical tasks, such as cache management, periodical backups, ontology consistency checking, etc.

Backup module. Using this module, ontologies can be safely stored in any destination (which is configurable). Backups can be scheduled as needed.

This service makes use of the planning and the ontology XML exportation services. This last service will be explained later in this section.

Business logic modules for WebODE workbench
These modules provide services for the WebODE ontology editor, although they can be used for any other application.

Ontology access service. This module is in charge of managing the ontologies' conceptual model, by inserting, deleting and modifying the definitions of all the terms in a domain.

It uses the database access service, and, optionally, cache and consistency check services, which are explained below.

Cache module. This module, which uses the database access and planning services, speeds up the access to ontologies, using several caching techniques that increase the performance of the ontology access service.

Consistency check module. This module also uses the database access and planning services from the Minerva application server. It performs consistency checks during taxonomy building, as presented in [15], decoupling these verifications from the ontology access service.

Ontology access API. Ontologies can be accessed from other applications through this well-defined API. This API is supported by the Minerva application server and can be accessed through RMI-IIOP.

XML ontology exportation module. It exports ontologies to valid XML, according to a well-defined DTD. This XML code can be used by other applications able to use this format or for later importations of the ontologies into other instances of the WebODE workbench.

XML ontology importation module. It imports ontologies in the XML format described by the DTD used in the XML exportation service. These ontologies must also accomplish consistency rules used by the consistency check service.
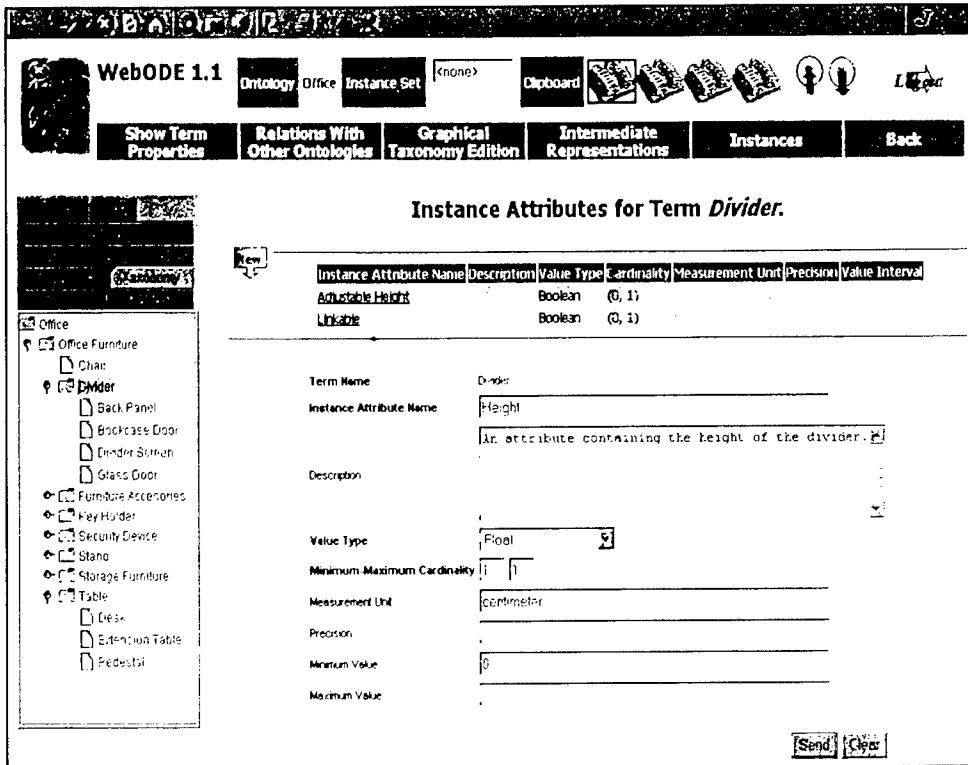
Ontology languages exportation/importation. Currently, several services exist in WebODE for the exportation and importation of ontologies to the following languages: RDF(S), OIL, DAML+OIL, X-CARIN and FLogic.

OKBC-based inference engine. It allows the ontology developer to perform queries and inferences on the ontology. The user can use predefined access primitives, which are based in the OKBC protocol, and create his/her own Prolog programs to perform inferences reusing the primitives already provided. It is based on Ciao Prolog.

Axiom prover module. It makes use of the inference engine, allowing the ontology developer to test if knowledge currently included in the ontology is consistent with its axioms. Each axiom is translated into Horn clauses and can be tested independently from the other ones.

Documentation module. WebODE ontologies are automatically documented in different formats, such as HTML tables (intermediate representations of METHONTOLOGY), HTML documents or XML files. The whole ontology or parts of it can be selected for this documentation generation. Views generated with OntoDesigner can be also selected for their documentation.

WebPicker: Ontology Acquisition from Web Resources. WebPicker is a service for the ontology acquisition from web resources that has been used for the acquisition of several standards and initiatives of products and services classifications in the e-commerce domain (UNSPSC, e-cl@ss and RosettaNet) as described in detail in [6].

Figure 2. Snapshot of WebODE's ontology editor while editing an instance attribute of a concept.

Information represented in web resources is transformed into a conceptual model specified in the XML syntax of WebODE, which is imported later into WebODE, so that its ontology editor can be used to redesign it.

OntoMerge: Ontology Merge. This service performs the merge of concepts (and their attributes) and relations of two ontologies built for the same domain. First, it assists the revision of both ontologies, based on a set of design criteria and semantic and syntactic relationships among the components of the ontology. Later, it uses natural language resources for establishing relationships between both ontologies. It performs a supervised merge of components from both ontologies using this information. Finally, it assists the final revision of the resulting merged ontology.

OntoCatalogue: Catalogue Generation from Ontologies. This service generates electronic catalogues out from ontologies, taking into account several configuration parameters, such as the depth of the taxonomy of products, attributes to be generated, the mappings between relations in the ontology and links in the catalogue, navigation hints through the catalogue, parts of the taxonomy to be generated, etc.

The catalogue generation from ontologies ensures a good and rich classification of products/services in it.

4.2.1. User Interface Tier: WebODE Ontology Editor

The WebODE ontology editor is an application for the development of ontologies at the knowledge level, based on the knowledge model already presented, which uses most of the services that have been presented above. Its user interface uses HTML, CSS (*Cascading Style Sheets*) and XML (*Extended Mark-up Language*). JavaScript and Java are used for several kinds of user validations.

Some specialized applets have been also included in the user interface, such as OntoDesigner, the axiom manager, the ontology browser and the clipboard.

The design rationale for this user interface is based on an easy-to-use and clarity basis. Figure 2 shows one of the screens of the editor, while including a new instance attribute for a concept. We will explain the most relevant components in this figure:

The clipboard applet is available in the upper part of the screen. It is used to copy and paste components' definitions, which is useful when creating components that are very similar to others. It has enough space for four definitions.

The ontology browser is placed on the left. It aids the navigation through the taxonomy of concepts, formulae, references and imported terms in the ontology. New components can be added by just double clicking on it and

11

Its ontology editor integrates in a common user interface most of the activities of the ontology life cycle, using the services available in the workbench. Its most interesting functionalities are: multiple-users support, guided conceptualization through the use of a very intuitive and simple user interface, multiple choice clipboard for easily copying and pasting components, complete consistency checks to ensure that the ontology contains valid knowledge, easy taxonomy edition either by using the form based user interface or a more complex and powerful graphical editor (OntoDesigner), an advanced term import providing by reference and by value fashions, instance handling independent from the ontology conceptualization, an API for accessing ontologies from any application using RMI or CORBA, and, finally, maximum interoperability thanks to the use of XML and several ontology specification languages.

This workbench has been successfully used in several projects: B2B and B2C ontology creation and reengineering in MKBEEM (IST 1999-10589), ontology acquisition through Webpicker in ContentWeb (*UNSPSC, RosettaNet* and *e-cl@ss*), ontology building and ontology metrics in (Onto)^{2}Agent (*Reference Ontology*), ontology building in project UPM:AM-9819 "Environment Ontology" (*Elements* and *Environmental Ions*) and electronic catalogues merging in MRO.

In the future we will provide extra services both to the WebODE ontology editor and the middleware area, such as ontology translation manager, ontology configuration management capabilities, ontology upgrading, etc.

ACKNOWLEDGEMENTS

REFERENCES

1. Benjamins, V.R., Fensel, D., Decker, S., Gómez-Pérez, A. *(KA)^2: Building Ontologies for the Internet: a Mid Term Report*. IJHCS, 51:687-712. 1999.

2. Berners-Lee, T., Fischetti, M. *Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web by its Inventor*. Harper. S Francisco. 1999.

3. Blázquez, M.; Fernández-López, M.; García-Pinar, J.M.; Gómez-Pérez, A. *Building Ontologies at the Knowledge Level using the Ontology Design Environment*. KAW98. Banff, Canada. 1998.

4. Chalupsky, H. *OntoMorph: A Translation System for Symbolic Knowledge*. KR-2000. 471-482. 2000.

5. Chaudhri V. K.; Farquhar A.; Fikes R.; Karp P. D.; Rice J. P. *The Generic Frame Protocol 2.0*. Technical Report, Stanford University.1997.

6. Corcho, O., Gómez-Pérez, A. *WebPicker: Knowledge Extraction from Web Resources*. NLDB'01. Madrid. June, 2001.

7. DAML+OIL. http://www.daml.org

8. Domingue, J. *Tadzebao and Webonto: Discussing, Browsing and Editing Ontologies on the Web*. KAW98. Banff, Canada. 1998.

9. Domingue, J., Motta, E. *A Knowledge-Based News Server Supporting Ontology-Driven Story Enrichment and Knowledge Retrieval*. EKAW 1999.

10. Duineveld, A.; Studer, R.; Weiden, M; Kenepa, B.; Benjamis, R. *WonderTools? A comparative study of ontological engineering tools*. KAW99. Banff. 1999.

11. Farquhar A., Fikes R., Rice J., *The Ontolingua Server: A Tool for Collaborative Ontology Construction*. 10th Knowledge Acquisition for Knowledge-Based Systems Workshop, Banff, Canada. 1996.

12. Fensel, D., Angele, J., Decker, S., Erdmann, M., Schnurr, H., Staab, S., Studer, R., Witt, A. *On2broker: Semantic-Based Access to Information Sources at the WWW*. WebNet 99. Honolulu. USA. October, 1999.

13. Fernández, M.; Gómez-Pérez, A.; Pazos, J.; Pazos, A. *Building a Chemical Ontology using methontology and the Ontology Design Environment*. IEEE Intelligent Systems and their applications. #4 (1):37-45. 1999.

14. Fridman, N., Musen, M. *PROMPT: Algorithm and Tool for Automated Ontology Merging and Alignment*. AAAI-2000. Austin, Texas. August, 2000.

15. Gómez-Pérez, A. *Evaluation of Ontologies*. International Journal of Intelligent Systems. 16(3). March, 2001.

16. Horrocks, I., Fensel, D., Harmelen, F., Decker, S., Erdmann, M, Klein, M. *OIL in a Nutshell*. EKAW'00. Juan les Pins. France. October, 2000.

17. Kifer, M.; Lausen, G.; Wu, J. *Logical Foundations of Object-Oriented and Frame-Based Languages*. Journal of the ACM. 1995.

18. McGuinness, D., Fikes, R., Rice, J., Wilder, S. *The Chimaera Ontology Environment*. AAAI-2000. Austin, Texas. August, 2000.

19. MKBEEM. http://mkbeem.elibel.tm.fr

20. *OILEd*. http://img.cs.man.ac.uk/oil/

21. *OntoEdit*. http://www.ontoprise.de/co_produ_tool3.htm

22. *OntoKnowledge*. http://www.ontoknowledge.org

23. RDF. http://www.w3.org/TR/REC-rdf-syntax/

24. Swartout, B.; Ramesh P.; Knight, K.; Russ, T. *Toward Distributed Use of Large-Scale Ontologies*. AAAI Symposium on Ontological Engineering. Stanford. USA. March, 1997.

25. *Using Protégé-2000 to Edit RDF*. Technical Report. Stanford University. http://www.smi.Stanford.edu/projects/protege/protege-rdf/protege-rdf.html.

A Library of Generic Concepts
for Composing Knowledge Bases

Ken Barker and Bruce Porter

Department of Computer Sciences
University of Texas at Austin
Austin, TX 78712 USA
{kbarker, porter}@cs.utexas.edu

Peter Clark

Knowledge Systems
Boeing Math and Computing Technologies
m/s 7L66, PO Box 3707, Seattle, WA 68124 USA
peter.e.clark@boeing.com

ABSTRACT
Building a knowledge base for a given domain traditionally involves a subject matter expert and a knowledge engineer. One of the goals of our research is to eliminate the knowledge engineer. There are at least two ways to achieve this goal: train domain experts to write axioms (*i.e.*, turn them into knowledge engineers) or create tools that allow users to build knowledge bases without having to write axioms. Our strategy is to create tools that allow users to build knowledge bases through instantiation and assembly of generic knowledge components from a small library.

In many ways, creating such a library is like designing an ontology: What are the most general kinds of events and entities? How are these things related hierarchically? What is their meaning and how is it represented? The pressures of making the library usable by domain experts, however, leads to departures from the traditional ontology design goals of coverage, consensus and elegance. In this paper we describe our *component library*, a hierarchy of reusable, composable, domain-independent knowledge units. The library emphasizes coverage (what is an appropriate set of components for our task), access (how can a domain expert find appropriate components) and semantics (what knowledge and what kind of representation permit useful composition). We have begun building a library on these principles, influenced heavily by linguistic resources. In early evaluations we have put the library into the hands of domain experts (in Biology) having no experience with knowledge bases or knowledge acquisition.

Keywords
knowledge engineering, ontologies, knowledge reuse

INTRODUCTION
The traditional audience for concept taxonomies includes knowledge engineers, ontologists and philosophers. This audience is often interested in ontologies as elegant models capturing a natural division of kinds of things in the universe of discourse. When the intended audience includes experts in particular fields of knowledge who hope to use the ontology to represent abstractions from their fields, the pressures on the design of the ontology shift.

It is a claim of our research [28] that users with no experience in knowledge engineering will be able to represent knowledge from their domain of expertise by instantiating and composing generic components from a small, hierarchical library. Components are coherent collections of axioms that can be given an intuitive label — usually a common English word. The components should be general enough that their axiomatization is relatively uncontroversial. Composition consists of specifying relationships between instantiated components so that additional implications can be computed.

As a guiding principle in building the library we have chosen to restrict both the number of components (to a few hundred) and the size of the language of composition — the relationships between components (currently less than a hundred). Our goal is to achieve coverage through composition rather than through enumeration of a large number of concepts. The small library and simple composition language also have the benefit of being easy to learn for users with no knowledge engineering experience.

This design principle presents two research questions: 1) is such a system easy for users to master? 2) is such a system sufficient to represent sophisticated domain knowledge? We have evidence that the system is indeed usable by domain experts. The quality of the representations created by our domain experts is under review.

In an attempt to make the library more accessible to users unfamiliar with knowledge engineering, we have taken a somewhat different approach to building our ontology: we have taken inspiration from English lexical resources (such as dictionaries, thesauri and English word lists) and Linguistics research. We are certainly not rejecting traditional knowledge engineering approaches, trying instead to reconcile them with language usage. Rather than try to avoid the clash between knowledge base concepts and

English words, we are attempting to make our component library intuitive to users accustomed to expressing knowledge with natural language.

This paper is part of a larger context of ongoing research on knowledge base construction by composition. Elsewhere we have discussed:

- motivations for the approach and algorithms [5, 6, 7]
- a graphical user interface [8]
- a knowledge representation and reasoning system [6]
- question answering and explanation generation [17, 24]

Within that context, this paper provides a brief tour of an early version of our component library to highlight its requirements, construction, contents and applications.

In the following section, we will describe our research project in more detail and the design constraints it places on our component library. We will then expose the contents of the library: what components it contains, what the language for composing components is and how we arrived at these. We will describe the ways in which the user accesses the library and report on some early observations of domain experts using the library.

The component library itself is online and can be browsed at http://www.cs.utexas.edu/users/mfkb/RKF/tree/.

THE PROJECT

A challenge problem for DARPA's Rapid Knowledge Formation (RKF) project [11] is to provide a software environment in which a biologist can build a knowledge base from information found in a textbook on Cell Biology. It must be possible to query the resulting knowledge base to obtain answers to the kinds of questions typically found at the end of a textbook chapter.

Our component library is being used in software (called *SHAKEN*) being developed by SRI, one of the primary contractors on the RKF project [27]. A user of *SHAKEN* builds a knowledge base by taking generic components from the library, instantiating them in a graph and connecting the instantiations to represent such things as static relationships between concepts, temporal and spatial information, event structure and process plans.

Requirements for Library Components

Given the project requirements, it is imperative that the user have a sufficient variety of components (*coverage*), that components that satisfy user expectations can be found easily (*access*) and that components are general enough to be used in a variety of contexts but specific enough to express non-trivial knowledge (*semantics*).

Coverage

There should be components to allow the user to encode a variety of knowledge from any domain. This is not to say that there should be as many components as there are words in a dictionary. Rather, the library should be broad-coverage with components specific enough that a user is

willing to make the abstraction from a domain concept. Conversely, the components should not be so specific that the user is handcuffed or does not care enough about the fine distinctions to use the components consistently.

Access

Although knowledge engineers and philosophers are interested in the structure of upper-level ontologies, it is less likely that a biologist describing DNA replication will be interested in learning our hierarchy in order to find components. Furthermore, since we are restricting the library to a small number of components, it is unlikely that there will be an exact match for a concept required by the user. For both these reasons, it is important that the interface help the user to find appropriate components.

Semantics

Our library is not merely a taxonomy of concepts. Each component contains axioms that encode the meaning of the component as well as how the component interacts with other components. These axioms must be general enough that the components are reusable. They must also be written in such a way that they do not clash with the axioms of other components when composed.

In the next sections we will discuss how these criteria, along with previous successful work on broad-coverage intuitive semantic inventories have guided the construction of our library.

RELATED WORK

In theory an ontology could be strong on all dimensions: coverage, access, semantics. In practice, however, an ontology, like most artifacts, is the result of engineering tradeoffs. For example, consider WordNet [21] and Sensus [16]. On one hand, they are as easily accessed as a thesaurus and have very broad coverage — they include the variety of concepts, relations, and modifiers used in everyday text. On the other hand, they provide very shallow semantics. For each English word, these ontologies give its senses along with their definitions, parts of speech, subclasses, superclasses and sibling classes. The definitions are free text (of limited use to computer programs) and the encoded relations are the only semantics.

The ontologies in Ontolingua [14] represent a different point in the space of tradeoffs. These ontologies are very limited coverage (they apply mainly to isolated topics in Engineering), but they have rich semantics. For example, they can be used to compute answers to Engineering problems stated in their vocabulary.

Cyc [10, 18, 19] represents yet another point. Its coverage is arguably as broad as WorldNet's, including many senses for entries in its lexicon. By one account, however, it receives lower scores on semantics and accessibility. Parmar [23] compared the representations of a handful of actions in Cyc and our component library. She found that Cyc often lacks axioms that capture the effects of actions — the representation does not support automated reasoning

about change. In terms of accessibility, Parmar measured the time she spent searching the Cyc ontology for entries that correspond to fifteen common actions represented in the component library.[1] On average, she spent over 3.5 minutes finding the Cyc term that most closely matches each action. By her assessment, many of these matches were not close: on a scale from 1 (poor) to 10 (perfect), the average score was less than 6.5.

It is clear, though, that Cyc is an example of a very different approach to coverage than what we propose, making it difficult to compare Cyc and our library. Cyc achieves coverage through enumeration. The semantics of concepts is often encoded in the fine distinctions between specialized subclasses.

Given our small number of components and relations, there is an obvious overlap with work on semantic primitives in Linguistics and Natural Language Processing. Schank's Conceptual Dependency Theory [25] enumerated a very small number of primitive actions and the relations between actions and their participants. A later refinement [26] included relations between pairs of actions. The extremely small number of primitives forces each one to cover many different concepts. The avoidance of names that clash with English words makes the Conceptual Dependency language less intuitive to users.

For our project, rich semantics is the first priority. The semantics of each component is expressed in KM [6], which in turn is defined in first-order logic. KM includes situation calculus — a knowledge representation and reasoning formalism for actions and the changes they cause. For example, the component for ENTER includes KM encodings of these axioms:

- ENTER is a type of MOVE, so instances of ENTER inherit axioms from MOVE, such as: the action changes the location of the object of the MOVE

- before the ENTER, the object is outside some enclosure

- after the ENTER, the object is inside that enclosure and contained by it

- during the ENTER, the object passes through a portal of the enclosure

- if the portal has a covering, it must be OPEN; and unless it is known to be CLOSED, assume that it is OPEN.

We plan to achieve good coverage by encoding a small set of general components for breadth. Depth can then be achieved through specialization and composition of components, without having the user write axioms.

We consider accessibility especially important, given that our users are not knowledge engineers. The library has been

---

[1] BREAK, CARRY, CREATE, ENTER, EXIT, MAKE-ACCESSIBLE, MAKE-CONTACT, MAKE-INACCESSIBLE, MOVE, RELEASE, REMOVE, REPAIR and TRANSFER.

designed to allow retrieval of components by means of semantically related search terms (as described below: see *Searching the Library*).

THE COMPONENT LIBRARY

In deciding what components to encode, we took inspiration from linguistic resources (such as dictionaries and thesauri). Our goal was not to build an online dictionary, but rather a library of components representing concepts that are general and intuitive enough to have obvious labels among common English words.

Furthermore, since domain experts are accustomed to expressing their knowledge with words, having explicit links between our components and dictionaries will help provide access to the library (as described below). These linguistic resources have much to offer:

- They have broad coverage of common terms. Our goal is to have a library of domain-independent, general components. This is where the strengths of general-purpose dictionaries and thesauri lie.

- Lexicographers pay attention to consensus view of the semantics of terms and common usage. Most dictionaries and thesauri are the result of many years of studying how terms are commonly used.

- They often group semantically related words into general semantic categories. These categories may be thought of as the most general concepts.

The Longman Dictionary of Contemporary English (LDOCE) [29] uses a "defining vocabulary" of about 2,000 words. All definitions in the dictionary ground out eventually to the defining vocabulary. WordNet groups semantically similar words into "synsets", which are themselves linked hierarchically. Roget's Thesaurus [20] divides the universe into six classes. Each class is subdivided into multiple sections, themselves subdivided. The thousand leaves in Roget's tree contain semantically related words (not quite synonyms), one of which is chosen as the representative for the group: the *headword*.

Each of these resources (the Longman defining vocabulary, a horizontal slice of the WordNet hierarchy, the Roget headwords) could be used as a list of general concepts, or as inspiration for an original list. None of these would suit our purposes *as-is*: the LDOCE vocabulary is not organized semantically; WordNet has considerably less coverage and fewer relations among non-nouns; Roget is somewhat arbitrary, and obviously influenced by his culture.

Generic Events

The main division in our component library is between *entities* (things that *are*) and *events* (things that *happen*). Events are states and actions. States represent relatively static situations brought about or changed by actions.

Actions

The actions are grouped into fifteen top-level clusters, each having several more specific subclasses (Table 1). The list

was developed under consultation of WordNet, the LDOCE defining vocabulary and Roget.

For example, the list of actions was compared to those headwords in Roget's Thesaurus that most naturally describe actions. In Roget, each headword heads several paragraphs; each paragraph contains words of the same part of speech. Although the headwords themselves are all nouns, some of the nouns are nominalizations and represent events more naturally than entities (for example, headwords #161: Production and #264: Motion). For these headwords, the noun paragraphs are often relatively empty, or contain more nominalizations. Their verb paragraphs are the richest. Although there are over one thousand headwords in Roget, our actions are general enough to cover most of the more action-like headwords (with the exception of those having to do with "sentiment and moral power" — an area we have so far ignored).

States

States are relatively temporally stable events. They are coherent collections of axioms that represent situations brought about or changed by actions. Many of our actions are defined in terms of the change in state they cause.

This relationship between actions and states is made explicit in the library: there are actions that put objects into states, actions that take objects out of states and actions whose behavior is affected by objects being in states. For example, the BREAK action puts an object into a BE-BROKEN state. The REPAIR action takes an object in a BE-BROKEN state out of that State. If an object is in a BE-BROKEN state, it may not be the instrument of any of the events for which

it is the intended instrument (though it may be instrument of other actions, such as using a broken computer to hold a door open). Other states include BE-RUINED, BE-CLOSED, BE-CONFINED, BE-TOUCHING, BE-ATTACHED-TO, etc.

There are other events that seem to fit somewhere between our actions and states, such as "being in motion". We expect that most of our actions have non-conclusive, durative counterparts (such as MOVING, CREATING, etc.). We are investigating continuous representations of our actions for the purpose of simulation. For now, our actions are all represented as discrete events.

Entities and Roles

To date, we have concentrated on events. We plan to research generic entities in a similar way. Our entity hierarchy is currently a relatively impoverished tree, serving as the root of a number of concepts from our test domain: Cell Biology (just over 500 at the time of writing).

Our preliminary investigation into entities led us to distinguish a separate class of *role concepts*. A role can be thought of as a temporally unstable entity. It is what an entity *is* in the context of some event. For example, PERSON is an entity while EMPLOYEE is a role. A PERSON remains a PERSON independent of the events in which she participates. Conversely, someone is an EMPLOYEE only by virtue of participation in an EMPLOY event.

Our library allows instances of roles to be linked to instances of entities as *adjunct instances* that can be used to capture both the role that an entity plays in an event, and the role it is intended to play (its *purpose*).

In order to determine how common role concepts are, we

Action	Description	Example Subclasses
ADD	add a part to an entity	--
REMOVE	remove a part from an entity	--
COMMUNICATE*	transfer information	INTERPRET, ENCODE, REPLY
CREATE	bring a new entity into existence	COPY, PRODUCE, PUT-TOGETHER
BREAK	cause an entity to be unable to be used as instrument (for events in which it is the intended instrument)	DESTROY, RUIN, TAKE-APART
REPAIR	"undo" a BREAK	--
MOVE	change the location of an entity	CARRY, ENTER, SLIDE
TRANSFER	change the possessor of an entity	DONATE, LOSE, TAKE
MAKE-CONTACT	make entities touch	ATTACH, COLLIDE
BREAK-CONTACT	make touching entities touch no longer	DETACH, DISPERSE
MAKE-ACCESSIBLE	allow an entity to participate (in various ways) in events	ADMIT, EXPOSE, RELEASE
MAKE-INACCESSIBLE	prevent an entity from participating in events	BLOCK, CONCEAL, CONFINE
PERCEIVE*	discern using senses	IDENTIFY, TOUCH
SHAPE*	change the shape of an entity	FLATTEN, FOLD
ORIENT*	change the orientation of an entity	FACE, ROTATE, TURN

Table 1: The top-level action clusters (actions marked * are under construction in the library)

conducted an experiment with the Collins online dictionary [9]. In that experiment we estimated that as many as 6% of nouns satisfy our criteria for role concepts. Furthermore, the most frequent nouns in the British National Corpus [4] also contain an estimated 6% role concepts. A more detailed discussion of roles and justification for a separate role hierarchy appear in [13].

COMPOSITION

The precoded axioms in library components provide much of the power that allows domain experts to build knowledge bases. Equally important is the ability to connect components in such a way that our knowledge representation system (KM [6]) can draw inferences from the composition beyond the union of the individual axioms of the components.

From the point of view of a user, composition is simply the linking together of library components. From this linking, however, KM is able to draw inferences by way of the knowledge encoded in components:

- *Conditional rules*: many components specify additional axioms that are asserted conditionally, dependent on the kinds of components they are composed with and the kinds of connections between them. For example, if the raw material of a PRODUCE is a SUBSTANCE, then the product is composed of that SUBSTANCE. If the raw materials are OBJECTs, then the product has those OBJECTs as parts.

- *Definitions*: many components specify the sufficient conditions under which KM can automatically reclassify instances. For example, an instance of MOVE whose destination is inside a container is automatically reclassified as an instance of ENTER, allowing KM to apply the axioms of that more specific component.

- *Simulation*: many components include preconditions that must be satisfied for an action to take place and the axioms that get asserted (or retracted) as a result of the action taking place. KM is able to simulate complex combinations of events and their participating entities.

The Language of Composition

In order to enable the kind of inferencing we have described, composition must have predictable semantics, which we accomplish by defining a restricted composition language of relations and properties. These relations and properties have their own axioms defining what inferences will be drawn from the composition of components.

Relations

We have defined a small set of relations to connect Entities and Events. Keeping the set small — we currently have about eighty — will allow us to maintain detailed axioms for each relation that capture the semantics of the composition of the related components. Writing such axioms for an open-ended set of relations might not be as feasible. The small number of relations also makes it easier

for our inexperienced users to learn to use them. Our relations that link an event to an entity describe the participants involved in the event. Our original set was inspired by a comprehensive study of *case roles* in Linguistics [3]. The set has been refined to account for the kinds of relationships expected for our particular event components. Event-to-Entity relations include agent, donor, instrument, object, recipient, result, etc.

To account for relationships between entities, we drew on previous research into the semantics of English noun phrases [2]. Since nouns can represent many things (not just entities) the semantic relationships within noun phrases are a superset of what is required to account for relations between our entities. The set of entity-to-entity relations currently includes content, has-part, location, material, possesses, region, etc.

The choice of relationships between events followed from studies in discourse analysis [1] and process planning [22]. These relations include causes, defeats, enables, entails, inhibits, by-means-of, prevents, resulting-state and subevent.

In addition to the relations among events and entities, we have a very small number of relations that involve roles [13]: relations that link an Entity to the Role it plays (or is intended to play) and between the Role and the Event.

Properties

We also have a small number of *properties*. Properties link entities to values. For example, the *size* of an entity is a property that takes a value. The value can be a cardinal (*25 kilograms*), a scalar (*big relative to housecats*) or a categorical (*brown*).

To define our set of properties, we turned once again to linguistic studies. Whereas events and entities usually surface as verbs and nouns in language, properties are closely related to adjectives. Since adjectives can also represent entities, we restricted our study to those adjectives that ascribe values to features of the nouns they modify. These adjectives are often called *ascriptive* adjectives.

We consulted work in Linguistics on adjective semantics, most notably Dixon [12] and Frawley [15]. We then conducted two exercises to build a list of properties.

For the first exercise we once again used WordNet, which explicitly distinguishes ascriptive adjectives from non-ascriptive adjectives (called *pertainyms* in WordNet). For the ascriptive adjectives, there are occasionally links to the noun that best describes the "attribute" to which the adjective ascribes a value. For example, the attribute for *large* is *size*. WordNet identifies about 160 unique nouns that are used as attributes. We used these attributes to populate the adjective classes proposed by Dixon, resulting in a first draft of a list of properties.
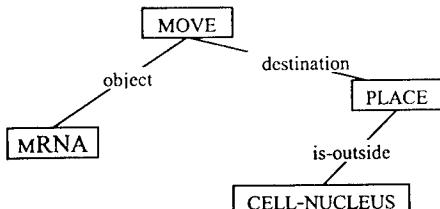
For the second exercise we once again consulted Roget. As described earlier, although the Roget headwords are nouns,

some of them more naturally describe events and have rich verb paragraphs. For other headwords, the adjective paragraphs are the richest (for example, headword #192: Size). Some of the headwords so naturally indicate properties that the verb paragraphs contain little more than "be <adjective>". In headword #201: Shortness, the first entry in the verb paragraph is "be short". Our experiment pulled headwords whose verb paragraphs begin with copular adjectival complementation phrases[2], producing a list of candidates for properties. This test also singles out ascriptive adjectives, since nonascriptive adjectives do not appear as copular complements. Unfortunately, the test did not filter out certain relations (such as "be identical to", "be different from", etc.). In the exercise we removed these relations by hand. The result was a list of approximately 230 headwords representing properties.

We then grouped all of the candidate properties into approximately 25 general categories. This final list of properties includes such properties as age, area, capacity, color, length, shape, size, smell and wetness.

A Simple Example of Composition

Consider the simple example of messenger RNA (mRNA) leaving a cell nucleus. In our interface, the user might describe this action by making an MRNA the object of a MOVE whose destination is outside a CELL-NUCLEUS:

The composition is richer than the mere connection of components due to the extra inferences KM can draw from the connection. Since the destination of the MOVE is outside some place, KM will recognize that this MOVE satisfies the definition of EXIT and will reclassify this instance of MOVE to be an instance of EXIT. Through the semantics of EXIT, KM will infer that prior to the EXIT the MRNA was inside the CELL-NUCLEUS, and that the MRNA must have EXITed through a portal in the CELL-NUCLEUS. KM can also infer that CELL-NUCLEUS must be playing the role of CONTAINER, and that its content prior to the EXIT included the MRNA. In simulating the EXIT, KM will assert that the location of the MRNA is a PLACE outside the CELL-NUCLEUS in the situation immediately following the EXIT.

USING THE LIBRARY

User Interface

Access to the component library is through a web-based tool for building compositions through graph operations:

[2] a phrase of the form "<copula> <adjective>", where the copulas include "be", "become", "seem", etc.

- add a component to a graph
- connect two components with a relation
- specialize a component to one of its subclasses
- unify two component instances

The use of this tool for knowledge entry is described in detail in [8].

Searching or Browsing the Library

The main disadvantage in restricting ourselves to a small number of generic components is that the library will probably not have a component that exactly matches the concept a user is looking for. The library interface, then, must make it easy for the user to find a close enough match. Our interface supports two modes of access to the library: a tree-based browser and a search tool.

Browsing the Library
Since the components are arranged hierarchically in the library, they can be browsed in the form of a tree. Our library browser allows the user to selectively expand the tree to view a component's subclasses. Since the library is small (by design) and we have attempted to make the component names intuitive and transparent, browsing the library through the tree is feasible. Nonetheless, given that our users are expected to have little or no experience with concept hierarchies, we believe it will be easier for them to find components through a search facility.

Searching the Library
The SHAKEN interface allows two kinds of searching: token match searching and semantic matching.

Token matching will return a component whose name exactly matches (or contains) the search term.

Semantic matching traverses the WordNet hierarchy for terms semantically related to components.

As part of the documentation for each component we have identified the WordNet entries that most closely match the semantics of the component. Our WordNet-based search tool finds the search term in WordNet, then climbs the hierarchy of hypernyms (more general terms) finding all components listing those hypernyms in their documentation.

Table 2 shows examples of search terms and their results.

search term	components found
assemble	ATTACH, CREATE, COME-TOGETHER, MOVE-TOGETHER
mend	REPAIR
gum-up	OBSTRUCT, BLOCK
busted	BE-BROKEN, BE-RUINED

Table 2: Examples of search terms and components found

One of the advantages of semantic searching is that results are sorted on the WordNet distance between the search term and the component, and on the depth of the component in our hierarchy. This gives preference to more specific library

components, meaning the user is more likely to choose a more specific (and therefore more semantically loaded) component than if she browsed top-down through the tree.

Documentation

One of the disadvantages of giving components names that are also English words is that users may have different biases about the senses of those words. These expectations may clash with the semantics of the components. This problem underlines the need for good documentation. Our documentation for components includes several things:

- Definitions. Given our particular users, it is important to have simple, non-technical definitions that describe all of (and only) the meaning of each component.

- One-line glosses. Early experiments have shown that users often accept or reject a component based solely on its name in a list. In our web-based interface one-line glosses are displayed as the mouse hovers over a component name.

- More detailed documentation. We also document the full semantics of components, including participants in an event, subevents, parts of an entity, etc.

- Examples. Several examples of varying complexity help show the intended use of components.

- Neighboring concepts. We are adding information to the documentation on "neighboring concepts" (similar components and how they differ from each other).

All documentation is available through the user interface, which can also show a graph representation of components in the library. In the graph the user can choose to see all, none or any subset of the links between the component and components connected to it through our relations.

EVALUATION

We have conducted three experiments in which biologists with no experience in knowledge engineering were asked to encode knowledge using our library and *SHAKEN*. In the first two experiments, users were given roughly one day of training on how to use the system and one day to encode a biological process (DNA transcription). In the third experiment we provided roughly one week of training. Users then (over eight weeks) encoded the knowledge from one chapter of a textbook in Cell Biology. All interaction between the users and developers was indirect, mediated by an impartial "gatekeeper" knowledge engineer.

In all three experiments the users have shown that our library search facility is able to guide them to generic components that they are willing to accept as abstractions of concepts they wish to encode. For example, our users were comfortable defining biological processes in terms of such generic actions as COLLIDE, SLIDE, ATTACH, RELEASE, etc.

At the end of each experiment, users were also asked to fill out a questionnaire. On average users found it moderately easy (slightly over 3 on a scale of 1-to-5) to find relevant

components in the library. They found it easy (4 on a scale of 1-to-5) to understand the components. They found the components useful (4) for representing knowledge. They found the restricted language of relations easy (4) to understand and use. They found it moderately difficult (slightly under 3) to cast biological knowledge in terms of the components and relations in the library.

More objective data on the quality of the representations of the knowledge built by the users is being collected.

LIMITATIONS AND FUTURE WORK

It is part of our claim that arbitrarily complex knowledge can be represented through the composition of simple generic components. The goal of knowledge reuse, however, suggests that the library will be even more powerful if it allows users to compose these more complex compositions themselves. One of our main tasks ahead is to populate the library with more complex (yet still general) components, such as processes of DELIVERY, PRODUCTION, COMMUNICATION, etc. We are also investigating ways to automate the composition of more complex components, such as through the use of interface templates.

Users identified several ways in which the library could be improved. For example, biologists are often interested in representing *functional* aspects of processes, not just *physical/spatial* descriptions. Including role concepts and the notion of *purpose* is currently making the encoding of functional knowledge easier. We have made extensions to our relationship language and are continuing to expand our role hierarchy and our generic entity hierarchy to admit functional representations more easily.

Experiments also underlined the importance of simple component names that do not have strong connotations in a particular domain. Our general actions of REPLICATE and TRANSCRIBE caused confusion among the biologists. We are currently reviewing the names (and documentation) of all our components. We are also reviewing components that users found unintuitive for other reasons. For example, users were not interested in the distinction between MOVE and its subclasses MOVE-FROM and MOVE-TO. We will likely remove these components for the sake of simplicity.

SUMMARY

In this paper we have described the process of building a library of knowledge components under the pressures imposed by our intended audience: domain experts with no experience in ontologies or knowledge engineering. In order to make the library accessible, we have taken inspiration from linguistic resources and built hooks to language into the components. In order to achieve power through composition of components, we have limited the library to a small number of components and relations with rich semantics. Preliminary trials with users inexperienced in knowledge engineering have been promising, giving us hope that domain experts will soon be able to encode their expertise in powerful knowledge bases.

REFERENCES

1. Barker, K., and Szpakowicz, S. Interactive Semantic Analysis of Clause-Level Relationships, in *Proceedings of PACLING '95* (Brisbane, April 1995).

2. Barker, K., and Szpakowicz, S. Semi-Automatic Recognition of Noun Modifier Relationships, in *Proceedings of COLING-ACL '98* (Montréal, August 1998), 96-102.

3. Barker, K., Copeck, T., Delisle, S. & Szpakowicz, S. Systematic Construction of a Versatile Case System. *Journal of Natural Language Engineering 3, 4* (December 1997), 279-315.

4. BNC. The British National Corpus. Available at http://info.ox.ac.uk/bnc/, 2001.

5. Clark, P., and Porter, B. Building Concept Representations from Reusable Components, in *Proceedings of AAAI '97* (Providence RI, July 1997), AAAI Press, 369-376.

6. Clark, P. and Porter, B. KM – The Knowledge Machine: User's Manual and Situations Manual. Available at http://www.cs.utexas.edu/users/mfkb/RKF/km.html, 2001.

7. Clark, P., Thompson, J. and Porter, B. Knowledge Patterns, in *Proceedings of KR-2000* (Breckenridge CO, April 2000), Morgan Kaufmann, 591-600.

8. Clark, P., Thompson, J., Barker, K., Porter, B., Chaudhri, V., Rodriguez, A., Thoméré, J., Gil, Y. and Hayes, P. Knowledge Entry as the Graphical Assembly of Components: The SHAKEN System, in *Proceedings of K-CAP 2001* (Victoria BC, October 2001).

9. COLLINS. *The Collins English Dictionary*. William Collins Sons, 1979. At the Linguistic Data Consortium, http://www.ldc.upenn.edu/Catalog/LDC93T1.html, 1993.

10. CYC. The Cyc Upper Ontology. Available at http://www.cyc.com/cyc-2-1/index.html, 2001.

11. DARPA. The Rapid Knowledge Formation Project. Available at http://reliant.teknowledge.com/RKF/, 2000.

12. Dixon, R. M. W. *Where Have all the Adjectives Gone?* Mouton, The Hague, 1982.

13. Fan, J., Barker, K., Porter B. and Clark, P. Representing Role Concepts, in *Proceedings of K-CAP 2001* (Victoria BC, October 2001).

14. Fikes, R. and Farquhar, A. Distributed Repositories of Highly Expressive Reusable Ontologies. *IEEE Intelligent Systems 14, 2* (March/April 1999), 73-79.

15. Frawley, W. *Linguistic Semantics*. Lawrence Erlbaum Associates, Publishers, Hillsdale NJ, 1992.

16. Knight, K. and Luk, S. Building a Large-Scale Knowledge Base for Machine Translation, in *Proceedings of AAAI '94* (Seattle WA, July-August 1994), AAAI Press, 773-778.

17. Lester, J. and Porter, B. Developing and Empirically Evaluating Robust Explanation Generators: The KNIGHT Experiments. *Computational Linguistics 23, 1* (1997), 65-101.

18. Lenat, D. Cyc: A Large-Scale Investment in Knowledge Infrastructure. *Communications of the ACM 38, 11* (November 1995), 33-38.

19. Lenat, D. and Guha, R. *Building Large Knowledge Based Systems*. Addison Wesley, Reading, Massachusetts, 1990.

20. Lloyd, S. M. *Roget's Thesaurus*. Longman, Essex, 1982.

21. Miller, G. A. (ed.). WordNet: An Online Lexical Database. *International Journal of Lexicography 3, 4* (Winter 1990).

22. Narayanan, S. Reasoning about Actions in Narrative Understanding, in *Proceedings of IJCAI'99* (Stockholm, August 1999), Morgan Kaufmann, 350-358.

23. Parmar, A. The Representation of Actions in KM and Cyc. Department of Computer Science, Stanford University technical report (forthcoming), 2001.

24. Rickel, J. and Porter, B. Automated Modeling of Complex Systems to Answer Prediction Questions. *Artificial Intelligence Journal 93, 1-2* (1997), 201-260.

25. Schank, R. C. *Conceptual Information Processing*. North-Holland Publishing Company, Amsterdam, 1975.

26. Schank, R. C. and Abelson, R. P. *Scripts, Plans, Goals and Understanding*. Erlbaum, Hillsdale NJ, 1977.

27. SRI. SRI's Rapid Knowledge Formation Team. Available at http://www.ai.sri.com/~rkf, 2001.

28. SRI. Proposal to DARPA's Rapid Knowledge Formation Project. Available at http://reliant.teknowledge.com/RKF/proposals/SRI/SRIproposal.htm, 2000.

29. Summers, D. (ed.). *Longman Dictionary of Contemporary English: New Edition*. Longman, Essex, 1987.

Knowledge Entry as the Graphical Assembly of Components

Peter Clark
John Thompson
Knowledge Systems
M&CT, Boeing
Seattle, WA 98124

Ken Barker
Bruce Porter
Computer Science
Univ. Texas
Austin, TX 78712

Vinay Chaudhri
Andres Rodriguez
Jerome Thomere
Sunil Mishra
SRI International, CA 94025

Yolanda Gil
ISI
USC
Marina del Rey
CA 90292

Pat Hayes
Thomas Reich-
herzer
IHMC, U W Florida
FL 32514

Abstract

Despite some successes, the lack of tools to allow subject matter experts to directly enter, query, and debug formal domain knowledge in a knowledge-base still remains a major obstacle to their deployment. Our goal is to create such tools, so that a trained knowledge engineer is no longer required to mediate the interaction. This paper presents our work on the knowledge entry part of this overall knowledge capture task, which is based on several claims: that users can construct representations by connecting pre-fabricated, representational components, rather than writing low-level axioms; that these components can be presented to users as graphs; and the user can then perform composition through graph manipulation operations. To operationalize this, we have developed a novel technique of graphical dialog using *examples* of the component concepts, followed by an automated process for generalizing the user's graphically-entered assertions into axioms. We present these claims, our approach, the system (called SHAKEN) that we are developing, and an evaluation of our progress based on having users encode knowledge using the system.

Keywords

Graphical knowledge entry, knowledge acquisition, components, composition, knowledge-based systems.

INTRODUCTION

Despite some successes, the lack of tools to allow subject matter experts to directly enter, query, and debug formal domain knowledge in a knowledge-base (KB) still remains a major obstacle to their deployment. Our goal is to create such tools, so that a trained knowledge engineer is no longer required to mediate the interaction. This paper presents our work on the knowledge entry part of this overall knowledge capture task. In particular, we present a novel technique of graphical dialog using *examples* of component concepts, and an evaluation of this technique. The particular applica-

tion domain we are working with is cell biology (although our techniques are not specific to this domain), and our focus has been on capturing domain knowledge, as opposed to problem-solving knowledge (the "what" rather than the "how to" knowledge). This work is being conducted as part of DARPA's Rapid Knowledge Formation (RKF) project [8].

CONTEXT, GOALS, AND CLAIMS

Context of the Work

Component-based approaches for knowledge capture are currently popular, and so we first describe where our work fits in this context. It is useful to view expertise as comprising *problem-solving* knowledge ("how" knowledge) and *domain* knowledge ("what" knowledge). Perhaps the most successful component-based, knowledge capture work has been with problem-solving knowledge, where reusable problem-solving methods (PSMs) can be assembled to produce task-specific problem-solvers, e.g., [4, 18]. Moreover, PSMs can also be used to guide acquisition of domain facts, as they "expect" certain types of knowledge in order to operate, e.g., the Expect system [2], Protege-derived tools [13].

Less work has been devoted to capture of domain knowledge, and it is capture of this kind of knowledge that is the primary focus of our work. Existing tools have focussed only on entry of taxonomic ("isa") knowledge and database-style facts, e.g., WebOnto [9], or have been targeted for use by knowledge engineers rather than subject matter experts, requiring logical axioms to be directly entered, e.g., Ontolingua [10], GKB [15], and the HITS Knowledge Editor [17]. Our goal is to allow users to encode these more complex, declarative axioms, describing both static objects and dynamic processes in the world, without requiring expertise in logic or AI. Our approach is analogous to work on PSMs: We similarly assume a library of components (but about objects and processes in the world, rather than about problem-solving strategies); and components similarly provide "expectations" to guide the user (but about how domain knowledge should be represented). The result of the knowledge capture process is a new set of axioms about a domain-specific object or process, which can then be used for question-answering.

Concerning our use of graphs for interacting with users, graph-

ical notations have frequently been found to be intuitive to users, e.g., in "concept maps," an informal graphical notation developed for educational settings, and used in tools such as WebMap [12] and by Univ. West Florida [3]. Similarly they have sometimes been found intuitive to knowledge engineers themselves, e.g., [16, 11]. Our goal is to exploit this intuitiveness for working with a pre-built library of representational components.

Claims

A central claim of our approach is that users can construct axiomatic representations by connecting pre-fabricated, representational components, rather than writing low-level axioms directly. By component, we mean a coherent set of axioms which describe some abstract phenomenon (e.g. the concept of "invade"), and which are presented to the users as a single representational unit. By composition, we mean the connection of such components together, and the computation of additional implications of the composite set of axioms. Components are intended to encode fairly abstract phenomena, such as knowledge about the concepts "invade," "break," "container," and "control system." Our goal is thus to recast the knowledge capture process as one of instantiation and assembly, rather than of axiom writing.

A second, related claim of our work is that components can be presented to users as graphs, and the user can then perform composition through graph manipulation operations. As a result, details of the underlying logic are hidden from users. Two implementation challenges for this are first expressing components as graphs, and second translating the user's graph manipulation operations back into logic, so that as the user manipulates graphs, the system records the logical equivalent of those operations. Our novel solution to this challenge is to have the dialog with the users be in terms of *examples* of their concepts of interest, coupled with a process for generalizing the user's graphically-entered assertions.

These two claims are related. In particular, the claim that knowledge capture can be treated primarily as an assembly process suggests that just a small number of axiom types (for stating connections and instantiations of existing components) will be sufficient to allow the users to build adequate representations. Although any full axiomatization of the user's concepts of interest may require complex axioms about space, time, actions, movement, etc., these will have been pre-built in the KB, and the user's job is thus simplified to describing domain-specific concepts using them. This provides a basis for the design of the graphical interface, as it only needs to support entry of this restricted set of axiom types, rather than the full range of possible first-order logic expressions. To the extent that these claims hold, a parsimonious tool for knowledge capture can be constructed, and to the extent that they do not, special add-ons will be needed to accomodate the user's needs.

TECHNICAL APPROACH

The user's goal is to create/extend a representation of a concept, i.e., encode axioms describing the properties, structure, and behavior of some domain-specific object or process. The user's activities are to: (1) Identify relevant components from the prebuilt KB; (2) connect and extend them to build a new representation; (3) save the result; and (4) test and ask questions about the new concept. The focus of this paper is on step 2 above, the construction of new representations.

Components

A component is a small set of first-order logic (FOL) axioms about a particular concept, gathered into a single data structure, encoding a coherent description of that concept [5]. The user is provided with a pre-built library of such components to work with. (Creating this library is a separate, major goal of our project [1]). For example, consider a (much simplified) component describing the process of "Invasion". It might include axioms stating that:

• The defending object has some barrier to protect it
• During an invasion, the invader penetrates that defensive barrier, then enters through it, then takes control of the attacked object.
• The invading agent is a tangible entity
• etc.

Statements such of these are encoded in first-order logic in the KB using (in our case) the frame-based language KM [6]. A simplified example of this notion of invade looks (in KM notation, with examples of equivalent FOL notation as footnotes)[1]:

```
;;; [1] "The invading agent is a tangible entity"
;;; [2] "The subevents of an invasion are a penetrate,
;;;       an enter, and a take control."
;;; [3] "During the penetrate, the invader penetrates
;;;       the defensive barrier of the attacked object."
;;; [4] "The first subevent is a penetrate event"
;;; etc.
(Invade has (superclasses (Attack)))
(every Invade has
  (agent ((a Tangible-Entity)))            ; [1]
  (object ((a Tangible-Entity with
               (has-part ((a Barrier))))))
  (subevent (
    (a Penetrate with                      ; [2a]
      (agent ((the agent of Self)))        ; [3a]
      (object ((the Barrier has-part of    ; [3b]
                 (the object of Self))))
      (next-event ((the Enter subevent
                               of Self))))
    (a Enter with                          ; [2b]
      (agent ((the agent of Self)))
      (object ((the object of Self)))
      (next-event (
        (the TakeControl subevent of Self))))
    (a TakeControl with                    ; [2c]
```

---

[1] Briefly on KM's syntax: slots (lowercase) are binary predicates, classes (mixed case) are sorts/types, 'every' denotes universal quantification and 'a' denotes existential quantification. See [6] for further details.

23

```
          (agent ((the agent of Self)))
          (object ((the object of Self))))))
      (first-subevent (
          (the Penetrate subevent of Self))))  ;[4]
```
Or in standard FOL syntax:

[1] $\forall i\ isa(i, Invade) \rightarrow$
    $\exists y\ isa(y, Tangible\text{-}Entity) \wedge agent(i, y)$

[2a,3] $\forall i\ isa(i, Invade) \rightarrow$
    $(\exists p\ subevent(i, p) \wedge isa(p, Penetrate) \wedge$
    $(\forall a\ agent(i, a) \rightarrow agent(p, a)) \wedge$
    $(\forall o, b\ object(i, a) \wedge has\text{-}part(o, b) \wedge$
       $isa(b, Barrier) \rightarrow object(p, b)))$

etc.

These axioms provide one fairly general model of invasion for the user to start from, and use concepts which themselves already have rich semantics in the KB. For example, axioms about the concept of Enter (not shown here) encode that if something is entered, then the entering object will be spatially inside afterwards, that the path of entry will necessarily cross the boundary of the entered thing, etc. The KB uses a rich language for describing the properties and effects of actions, allowing questions to be answered through both deductive reasoning and running simulations.

Displaying Axioms to the User

To present the axioms about a concept C to the user, the raw axioms are not presented directly. Rather, the user sees an *example I* of that concept, i.e., a set of ground facts about I, computed from those axioms. Ground facts are both comprehensible and graphable, and provide an easy-to-grasp (although approximate) summary of what the KB "has to say" about a concept. The user then builds new concepts by interacting with this and other examples.

For instance, suppose the user is wanting to build a representation of how a virus invades a cell. One starting point for this is the pre-built concept of *Invade*, which the user would locate by browsing the component library. To then display the axioms for invade, our system SHAKEN then:

1. creates an instance I of *Invade* (i.e., asserts the existence of an individual of type *Invade*), then
2. queries the KB for values for each of I's slots (i.e., uses inference to compute all ground facts of the form $slot(I, x)$). Existentially quantified variables are Skolemized, and thus the result of this is typically a set of ground, binary facts between Skolem individuals. An example is shown shortly.
3. Recursively applies step 2 to each such value x found, up to a certain depth limit.
4. Presents this database of ground facts to the user as a graph, where each Skolem instance is a node, and each binary relation is an arc. Nodes are labelled with the most specific class(es) (i.e., sort, type) that each instance belongs to.

The boundaries of this procedure, and hence the extent of the resulting graph, are set manually by us, the knowledge engineers, by pre-specifying which slots should be included in the graph, and the depth of recursion. An autolayout algorithm then determines the spatial layout of nodes and arcs.

The graphs are computed dynamically by this procedure at run-time, and thus can automatically adapt as new axioms are added to the system. From here, the user can modify the initial presentation by moving, expanding, or contracting nodes in the graph, hiding or exposing edges, and saving his/her revised presentation so new uses of that concept will appear the same way.
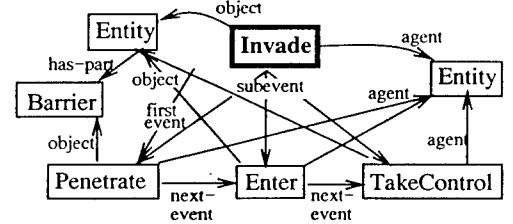
Applying this procedure to our (simplified) *Invade* representation, SHAKEN would first generate a Skolem instance denoting an example of invade, e.g., named *Invade1* (terms ending with numbers denote Skolem constants), and hence the set of facts:

   agent(Invade1, Tangible-Entity2)
   object(Invade1, Tangible-Entity3)
   has-part(Tangible-Entity3, Barrier4)
   first-subevent(Invade1, Penetrate5)
   subevent(Invade1, Penetrate5)
   agent(Penetrate5, Tangible-Entity2)
   object(Penetrate5, Barrier4)
   next-event(Penetrate5, Enter6)
   subevent(Invade1, Enter6)
   agent(Enter6, Tangible-Entity2)
   object(Enter6, Tangible-Entity3)
   next-event(Enter6, TakeControl7)
   subevent(Invade1, TakeControl7)
   agent(TakeControl7, Tangible-Entity2)
   object(TakeControl7, Tangible-Entity3)

From this, a graph would be synthesized and displayed, where each node corresponds to a Skolem instance and each arc a binary relation. The resulting graph may look, for example:

Entering Knowledge by Interacting with the Graph

Suppose that the user wishes to build a representation of how a virus invades a cell. He/she would first provide a name for this new concept (e.g., $VirusInvasion$), and then locate one or more components in the KB to start from. In this example, the user may select the *Invade* concept shown earlier. As a result, SHAKEN generates the above database, and also adds the assertion that the root instance denotes (an example of) the user's new concept, i.e., asserts:

   $isa(Invade1, VirusInvasion)$

As a result, the label on this root node appears as "VirusInvasion," and would appear as shown in the top graph in Figure 1. The entire graph is treated as a "representative instance" of the user's new concept.

To develop a model of how (for example) a virus invades a cell using this and other components, the user needs to encode facts such as:

A1. the invading agent is a virus
A2. the invaded object is a cell
A3. the penetrate is by means of endocytosis
A4. the agent in the endocytosis is the invaded object (i.e., the cell)
A5. there is also a delivery (of DNA) taking place
A6. there are certain correspondences between the invade and the delivery e.g.,
    A6.1 the invader (i.e., the virus) is the same as the agent in the delivery
    A6.2 the thing delivered is the DNA of that virus.

Rather than writing these statements in logic, the user makes them through the graphical interface via graph manipulation operations. This is possible because these axioms (specifying the composition) are generally all of a simple form: the complex axioms about virus invading a cell, e.g., how the spatial relationships of the objects change during the process, are mainly applications of more general axioms which already reside in more general components, and thus have already been pre-encoded in the component library. The user's job (and thus the interface) is thus simplified to using just this restricted subset of axiom types. As stated earlier, this is an important claim of our work, namely that by pre-encoding components well, a set of simple types of connections between them will be adequate for KB construction by a user who is not a trained knowledge engineer.

SHAKEN currently supports four types of axiom-building graph operations (plus others for controlling layout and node visibility). Each graphical operation corresponds to a simple, ground assertion about the example he/she is working on, and each time the user performs an operation, SHAKEN makes the corresponding logical assertion in the KB. On completion, an algorithm generalizes these assertions to hold for *all* instances of the concept C the user is describing, and the resulting axiom set captures the knowledge the user has entered. If the user is happy with his/her work, the axiom set is added to the KB, and can be further refined later and/or used itself as a component for defining new concepts.

The four graphical operations and their corresponding axioms are listed in Table 1 and illustrated in Figure 1. They are as follows:

Specialize: Refine an object's most specific class(es). In Figure 1, the user has clicked on the first *Entity* node and then selected *Cell* from a menu, to state that the thing being invaded is a cell. This asserts *isa(Tangible-Entity3,Cell)* in the KB.

Add: Add a new participant to the representation. In Figure 1, the user has selected the graph for *Virus*. This asserts $\exists x\, isa(x, Virus)$, which is then Skolemized to *isa(Virus8, Virus)* and asserted in the KB.

Unify: State that two objects are coreferential. In Figure 1,
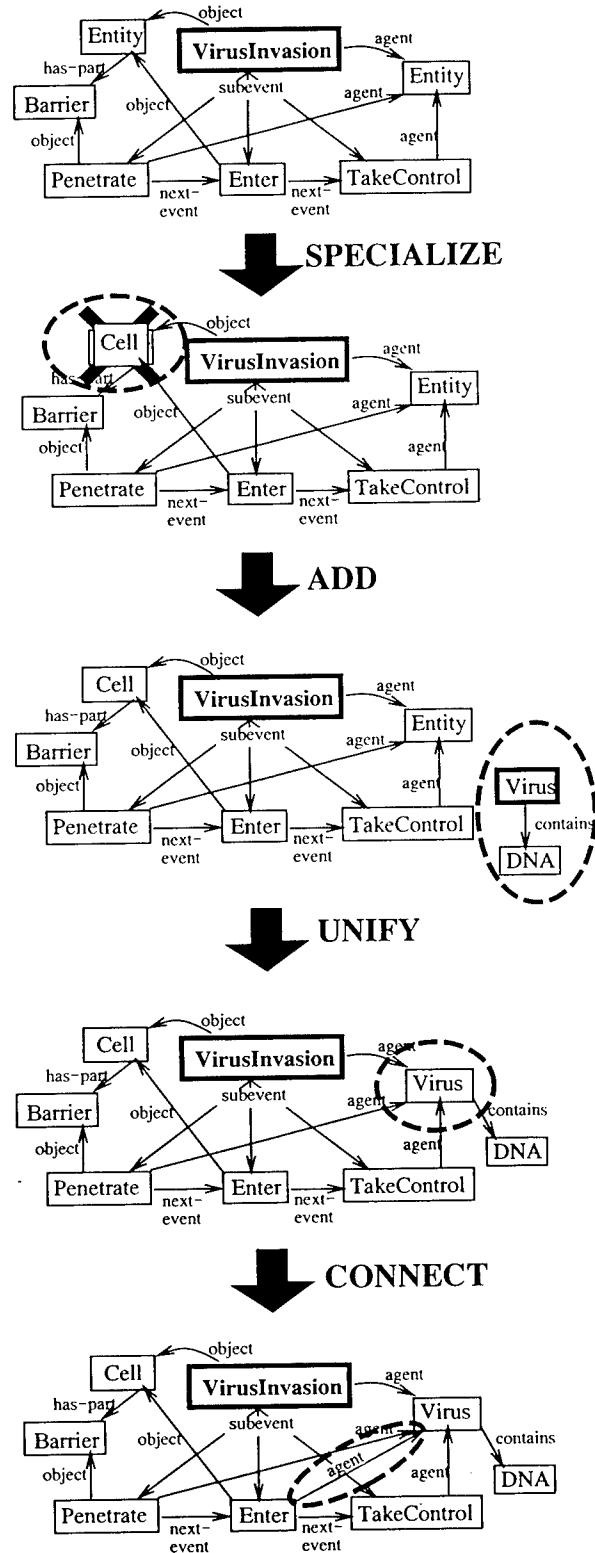
Figure 1: Examples of the four axiom-asserting graphical operations that the user can use in SHAKEN.

25

Operation	Examples	Graphical Action	Graphical Result	Logical Assertion
specialize	A1, A2	click node I + select class	I's label changes to $Class$	$isa(I, Class)$
add	A3, A5	click button + select class	graph for class appears	$\exists c\ isa(c, Class)$
unify	A4, A6.1, A6.2	drag node I onto node I'	nodes fuse	$I = I'$
connect	A3, A4	sketch arc R between I, I'	arc appears	$R(I, I')$

Table 1: The four graphical operations in SHAKEN, and their logical equivalent. The examples refer to the axioms listed in the body of this paper.

the user has stated that the invader is the same object as the virus he/she just introduced, by dragging one on top of the other (which then fuse). This asserts *Tangible-Entity2 = Virus8*.

Connect: Assert a relation holds between two nodes, by sketching an arc. In Figure 1, the user's action results in $agent(Enter6, Virus8)$ being asserted.

Implications of the User's Assertions

The user's assertions may have logical implications in the KB, and hence may imply changes to the graph the user is viewing. For example, if the user has two graphs for two distinct viruses displayed (similar to the $Virus$ graph in Figure 1), and he/she then unifies the two viruses, this implies (from constraints in the KB) that the two DNA nodes must also be coreferential, and so should also be unified. To feed these changes back to the user, first these "knock on" effects are computed in the KB, and then the graphs the user is viewing are recomputed and redisplayed (preserving as much of the original spatial layout as possible).

Thus SHAKEN is not just a passive graph editor, but is actively engaged in showing the user consequences of his/her assertions when they affect the visible graphs. This is an important and distinctive property of our interface, and necessary to keep the graphs and the KB synchronized so that the dialog remains coherent.

Axiom Synthesis from Graph Operations

Through the above means, the user can only enter ground facts about this particular *example* of his/her new concept. The final stage of this knowledge entry phase (before testing and debugging) is the automatic generalization of those assertions to hold for *all* instances of the user's new concept. This generalization process is algorithmic (rather than inductive), which we now describe.

The axioms which the user has graphically entered are all relationships either between Skolem instances, or between a Skolem instance and a class. For example, the user would enter the earlier assertion A2 that "the invaded object is the cell" by a 'specialize' graphical operation on the node denoting the invaded object, namely *Tangible-Entity3*. This is illustrated in the first step of Figure 1, and results in the corresponding logical assertion being added to the KB:

$isa(Tangible-Entity3, Cell)$

To generalize this to apply to *all* instances of the user's new concept $VirusInvasion$, the algorithm behaves as follows:

1. First, the axiom is rephrased to only mention the "root" Skolem instance R, namely the one denoting the concept the user is defining. In our example here, the root Skolem R is *Invade1*, denoting the user's example of *VirusInvasion*. Informally, this means a statement like

   "*Tangible-Entity3* is a Cell"

   is rephrased as

   "the object of *Invade1* is a Cell"

   Note that the latter statement only mentions the root instance $Invade1$. This is required for step 2.

   Formally, each Skolem instance I in the ground assertion is replaced with a variable v, and a formula is added as an antecedent which uniquely identifies v as that Skolem instance I, and no other. In other words, this formula is a *description* of I, stating the unique way it is related to the root R, i.e. is true only when $v = I$. In SHAKEN, this formula is a path (role chain) of relationships from the root instance R to I, found by a simple graph search procedure starting at R and looking for path(s) to I. The resulting formula has the form:

   $$p_1(R, x_1) \land p_2(x_1, x_2) \land ... \land p_n(x_{n-1}, v)$$

   and thus the rephrased axiom has the form:

   $$\forall x_1, ..., v\ p_1(R, x_1) \land ...p_n(x_{n-1}, v) \rightarrow axiom(v)$$

   In the above example, the ground fact

   $isa(Tangible-Entity3, Cell)$

   would thus be rephased as

   $$\forall v\ object(Invade1, v) \rightarrow isa(v, Cell)$$

   If there are multiple such objects, then additional predicates are added to the formula until it only holds for I (here *Tangible-Entity3*).

2. This axiom is then generalized so that it holds for *all* examples of the concept $NewC$ being defined. This is done by replacing the root instance R in the axiom with a variable r, and adding an antecedent stating the axiom holds for *all* cases when r is an instance of $NewC$, i.e., when $isa(r, NewC)$ is true. The final axiom will thus have the form:

$$\forall r\ isa(r, NewC) \rightarrow formula(r)$$

where $formula(r)$ is the axiom from step 1 with R replaced by r. In the example, the final result would be:

```
;;; "The invaded object is the cell."
∀r isa(r, VirusInvasion) →
      (∀v object(r, v) → isa(v, Cell))
```

It should be clear that the purpose of step 1, rewriting in terms of R, is to pave the way for step 2, where R is replaced by a universally quantified variable.

One complication must be dealt with for the 'Add' operation. When the user adds a new component to the screen through the 'Add' operation, it is initially disconnected from the root graph describing the user's new concept. This means that there is no path connecting the root instance R to instances in that new graph, and thus the reformulation in step 1 will fail. To handle this, a (graphically invisible) "participant" relation is asserted to hold between R and the new instance whose existence has been declared, stating that the new instance is a "participant" in R. As a result, the procedure in step 1 can now find a path from R to that instance and others in its graph by traversing that participant relation. For example, in graph 3 of Figure 1, the user has added the graph for $Virus$, so the assertion is added to the KB:

$$participant(Invade1, Virus8)$$

This allows paths to instances in the new graph to be found.

Axiom Synthesis with 'Delete' Operations

An undesirable characteristic of this axiom synthesis routine is that it assumes a monotonically growing KB. As each axiom includes logical descriptions of the objects the user manipulated, generated at a fixed moment in time, the user cannot later delete facts about those objects without risking invalidating those descriptions, and hence his/her earlier synthesized axioms. In the earlier example, if the user were to later delete the assertion $object(Invade1, Tangible-Entity3)$, then the synthesized axiom shown would no longer be valid.

We have recently prototyped (but not deployed) an alternative, and very different, axiom synthesis routine which supports non-monotonic change, thus providing the user with a much desired 'Delete' (of a node or arc) operation, and which we briefly describe here. Rather than converting each user *action* into an axiom, this alternative approach stores the user's *final graph* itself as a (large) "forall...exists..." axiom stating that "forall instances of the concept being defined, all the objects and relationships in the graph exist." This axiom is created only at the end of the user's session, and overwrites any previous axiom for that concept, thus allowing the user to delete as well as add to the graph. To support this, two extensions were needed for the inference engine: First, the user's delete operations must override implications from the KB, to prevent SHAKEN re-inferring the deleted arc/node. Second, when inferencing with several "forall...exists..." statements
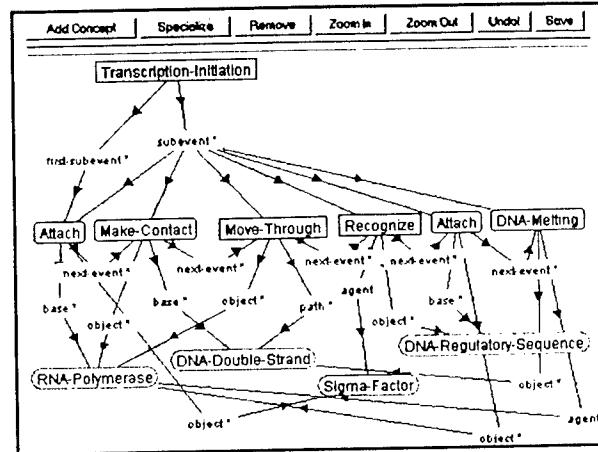
Figure 2: A screendump of SHAKEN's graph interface, showing a user's representation of how the process of RNA transcription is initiated.

like this, the inference engine needs to heuristically determine coreferences between instances in the (logical equivalent of) the multiple graphs, so that they are appropriately merged together. Although these extensions complicate the formal semantics of axioms in the KB, they will provide users with a much-desired 'Delete' capability in later versions of the system. It has become increasingly clear from our experiments that assuming simple axiom semantics and a monotonically growing KB are difficult positions to maintain for real-world knowledge acquisition.

Entering Knowledge through the Interface

Through these operations, the user's task is to assemble a representation of new concepts. The user first provides a name for his/her new concept, then selects from the KB the most appropriate, pre-built generalization of that concept to start from. SHAKEN then displays the initial graph for (an instance of) that new concept. From here, the user adds, specializes, connects, and unifies nodes on the screen to gradually build a representation. An example of one of the simpler representations built by a user during the experimental evaluation is shown in Figure 2. Separate testing and question-answering tools [14, 7] allow the users to debug and pose questions to their representations until they are satisfied.

EVALUATION AND DISCUSSION

During the summer of 2001, an extensive evaluation was performed on SHAKEN, including the graphical entry component described in this paper. Four users who were trained in biology (three graduate students, one undergraduate), and who had no background in programming or formal logic, underwent a week's training in using the system. Following this, they then independently worked over a period of four weeks (except for one who worked for three weeks due to vacation constraints) on encoding an 11-page subsection from a graduate-level textbook on cell biology, including debugging and testing their representations. These trials was run by

an independent contractor (IET, Inc.), rather than ourselves. For the trials, the basic component library was augmented with representations of the prerequisite knowledge needed to understand the subsection. This augmentation was carefully controlled by IET to prevent knowledge from the subsection itself being included in the initial library.

Most significantly, all four users were able to both grasp the basic approach of assembling components, and construct representations using the graphical interface. Over the four week period (three for one user), the users constructed representations of 442 biological concepts (approximately 100 each) ranging in complexity from a single node (i.e., just a concept name) to graphs containing over 100 nodes. The total number of synthesized axioms in the users' final representations (where each axiom-building graphical action results in one axiom being synthesized) were 1408, 567, 1296, and 921 respectively. The users also tested their representations by posing (independently set) questions to them using a menu-driven question-asking interface. The questions were approximately high-school level difficulty, and were mainly "reading comprehension" type questions requiring only simple inference, although a few required more complex inference and simulation. Sometimes this testing revealed errors or inadequacies in the representations, which the users would then correct. The final, system-generated answers to the test questions were collected, and, after the four week period was complete, were scored by an independent biologist on a 0-3 scale (0 = completely incorrect, 1 = mostly incorrect, 2 = mostly correct, 3 = completely correct). At time of writing the final scores are still being tallied, but the evaluators report that the average score is close to 2, reflecting that the users had successfully constructed reasonably accurate, inference-capable representations. These results are significant: they suggest that the basic machinery works, providing a basic vehicle for axiom-building without the users having to encode axioms directly (or even encounter terms like "concept," "relation," "instance," "quantification," etc.); and that those axioms are built in terms of prebuilt knowledge, hence bringing background knowledge into the representations for future reasoning and question-answering tasks. This is an important achievement for this project. In a separate questionnaire to the three users at the end of the four weeks (the fourth user still to complete the questionnaire when back from vacation), all three rated SHAKEN as "useful" as a tool to enter knowledge (on a scale of useless/not so useful/moderate/useful/very useful), and "easy" to use (on a scale of very easy/easy/moderate/difficult/very difficult). This again points to the viability of this approach.

Although the users were able to encode a lot of knowledge with SHAKEN, there was also knowledge they were unable to encode due to the limited expressivity of the interface. The most significant of these, as reported by the users, were:

- simple attribute values (which had to be represented as classes in the current system), e.g., rates, sizes

- equational information e.g., how rates vary with time
- temporal relations, e.g., simultaneous/temporally overlapping events
- pre/post conditions for actions
- richer process models, e.g., repetitive events
- sequences, e.g., nucleotide sequences
- negative information, e.g., being able to say something *doesn't* happen
- locational/spatial information b
- how things change with time (fluent information). The system assumes the graph describes the world at the start of a process, and so, for example, it is not possible to describe what an object looks like at the *end* of a process.

Similarly, comparing the users' source text with what they actually encoded, it is clear that they abstracted away many of the details contained in the text. For example, the source text for the user-built representation in Figure 2 begins:

"In bacteria, RNA polymerase molecules tend to stick weakly to the bacterial DNA when they make a random collision with it; the polymerase molecule then slides rapidly along the DNA..."

If we compare this text with what was actually encoded (see Figure 2) by one user, we can see that events like "stick" and "slide" have been abstracted to `Make-Contact` and `Move-Through` (whose representations are pre-built in the library), and other phrases like "weakly", "rapidly", "random", and "tend to" have been omitted. (This user has also added an extra prerequisite step, mentioned in text elsewhere, of the sigma factor attaching to the polymerase). In fact, to our surprise, the users seemed to have little or no trouble abstracting out details when building their representations, and they quickly grasped what could be represented and what could not using SHAKEN. In contrast, users (such as ourselves) with more experience in knowledge representation sometimes had more difficulty abstracting in this way when attempting the same encoding task. Interestingly, despite these limitations, the users themselves felt they had managed to encode much of the core knowledge. After the trials were completed, they were each asked: "Next week SHAKEN will be asked questions and answer them using the knowledge you entered, and based on that it will be given a grade. Do you think it will be a passing grade?". All three users responded quite confidently, saying things like "definitely" and "oh yes". When asked "What kind of grade?", two users answered A-, the third said B. Independent of whether this perception is correct or not, it is interesting that the users themselves felt they had been able to teach the system much of the biological knowledge in the selected subsection.

Another surprise to us was the size of the representations the users created. Some of the users' graphs contained over 100 nodes in, and were rich in relationships and associations. (The users could manage graphs this size as the interface allows them to hide/expose parts of the graph, so not all nodes

need be visible at once). The graph shown in Figure 2 is thus not representative of the typical complexity that the users were able to build. The fact the users were able to build such sophisticated representations perhaps partially explains their confidence in the amount of knowledge encoded.

Despite the reasonable performance scores, there were still errors in the users' final representations. Some of these arose due to the use of linguistic-style devices (e.g., metonymy, analogy, metaphor, approximation) in their graphical assertions. Examples we observed include: indirect reference; interchangeably referring to an object and an event; interchangeably referring to an object and a location; missing coreference statements; overgenerality; missing context (stating a conditional fact as a universal statement); and misuse of case roles. An important future task is to make SHAKEN more active in interpreting and critiquing the users' input, so these errors are detected and corrected more agressively.

A final, interesting point concerns the interaction between representation and question-answering. SHAKEN assumes a single, universal representation for each biological concept, while sometimes the users wanted to be able to represent the same concept in multiple ways, depending on what kind of tasks they wanted their representation to support. Sometimes this resulted in the users creating multiple representations for the same concept (using slightly different concept names). A more principled method for handling different viewpoints like this, either in the KB itself and/or in the reasoning and question-answering procedures, would be desirable.

SUMMARY
We have presented a method for knowledge capture, in which knowledge entry is viewed primarily as a task of component assembly rather than axiom-writing, and shown how it can be implemented using a graph-based interface, based on a novel technique of dialog using examples. Our trials suggest that users can both grasp the approach and construct sophisticated, axiomatic representations, despite having no formal training in logic or AI. This is a potentially significant achievement for enabling subject matter experts to build KBs directly.

ACKNOWLEDGEMENTS

REFERENCES
1. K. Barker, B. Porter, and P. Clark. A library of generic concepts for composing knowledge bases. In *Proc. 1st Int Conf on Knowledge Capture (K-Cap '01)*, 2001.

2. J. Blythe, J. Kim, S. Ramachandran, and Y. Gil. An integrated environment for knowledge acquisition. In *Int. Conf. on Intelligent User Interfaces*, 13-20, 2001.

3. A. J. Canas, K. M. Ford, J. D. Novak, P. Hayes, T. Reichherzer, and N. Suri. Using concept maps with technology to enhance cooperative learning in latin america. *Science Teacher*, 2001. (To appear).

4. B. Chandrasekaren. Generic tasks in knowledge-based reasoning: High-level building blocks for expert system design. *IEEE Expert*, pages 23-30, Fall 1986.

5. P. Clark and B. Porter. Building concept representations from reusable components. In *AAAI-97*, pages 369-376, CA, 1997. AAAI.

6. P. Clark and B. Porter. KM – the knowledge machine: Users manual. Technical report, UT Austin, 1999. (http://www.cs.utexas.edu/users/mfkb/km.html).

7. P. Clark, J. Thompson, and B. Porter. A knowledge-based approach to question-answering. In R. Fikes and V. Chaudhri, editors, *Proc. AAAI'99 Fall Symposium on Question-Answering Systems*. AAAI, 1999.

8. DARPA. The rapid knowledge formation project (web site). http://reliant.teknowledge.com/RKF/, 2000.

9. J. Domingue. Tadzebao and webonto: Discussing, browsing, and editing ontologies on the web. In *Proc. KAW'98*, 1998.

10. A. Farquhar, R. Fikes, and J. P. Rice. A Collaborative Tool for Ontology Construction. *International Journal of Human Computer Studies*, 46:707-727, 1997.

11. B. R. Gaines. An interactive visual language for term subsumption languages. In *IJCAI'91*, 1991.

12. B. R. Gaines and M. L. G. Shaw. Webmap: Concept mapping on the web. *The World Wide Web Journal*, 1(1), 1996.

13. W. E. Grosso, H. Eriksson, R. W. Fergerson, J. H. Gennari, S. W. Tu, and M. A. Musen. Knowledge modeling at the millennium. In *Proc. KAW'99*, 1999.

14. J. Kim and Y. Gil. Knowledge analysis of process models. In *IJCAI'01*, 2001. (to appear).

15. S. M. Paley, J. D. Lowrance, and P. D. Karp. A Generic Knowledge Base Browser and Editor. In *Proc. IAAI'97*. AAAI Press, 1997.

16. J. F. Sowa. *Conceptual structures: Information processing in mind and machine*. Addison Wesley, 1984.

17. L. G. Terveen and D. A. Wroblewski. A collaborative interface for browsing and editing large knowledge bases. In *AAAI'90*, pages 491-496, 1990.

18. B. J. Wielinga, A. T. Schreiber, and J. A. Breuker. KADS: A modelling approach to knowledge engineering. *Knowledge Aquisition*, 4(1), 1992.

Supporting Ontology Driven Document Enrichment within Communities of Practice

John Domingue, Enrico Motta, Simon Buckingham Shum, Maria Vargas-Vera, Yannis Kalfoglou

Knowledge Media Institute

Nick Farnes

International Centre for Distance Learning

The Open University
Walton Hall, Milton Keynes, MK7 6AA, UK
{j.b.domingue; e.motta; s.buckingham.shum; m.vargas-vera; y.kalfoglou; n.farnes}@open.ac.uk

ABSTRACT

Formative work by Lave and Wenger has articulated how *practices* emerge through the interplay of informal processes with symbolic codifications and artifacts. In this paper, we describe how ontologies can serve as symbolic tools within a community of practice supporting communication and knowledge sharing. We show that when a community's perspective on an issue is stable, it opens the possibility for introducing knowledge services, based on an ontology co-constructed by knowledge engineers with stakeholders. Using a case study we describe our approach, ontology driven document enrichment, looking at how ontology construction and population can be supported by web based technologies.

Keywords

Ontology, Semantic Web, Communities of Practice, Knowledge Management.

INTRODUCTION

Formative work by Lave and Wenger [13, 22] has articulated the nature of the *practices* from which the term *community of practice* derives its name. Practices emerge through the interplay of informal processes with symbolic codifications and artifacts:

> ...Such a concept of practice includes both the explicit and the tacit. It includes what is said and what is left unsaid; what is represented and what is assumed. It includes language, tools, documents, images, symbols, well-defined roles, specified criteria, codified procedures, regulations, and contracts that various practices make explicit for a variety of purposes. But it also includes all the implicit relations, tacit conventions, subtle cues, untold rules of thumb, recognizable intuitions, specific perceptions, well-

tuned sensitivities, embodied understandings, underlying assumptions, and shared world views. Most of these may never be articulated, yet they are unmistakable signs of membership in communities of practice and are crucial to the success of their enterprise. ([22], p. 47)

In this paper, we describe how ontologies [9] can serve as symbolic tools within a community of practice. We show that when a community's perspective on an issue is stable (i.e. there is reasonable consensus), it opens the possibility for introducing knowledge services, based on an ontology co-constructed by knowledge engineers with stakeholders. The ontology reflects a "shared world view", codifying "well-defined roles", "specified criteria" and "codified procedures." Throughout, we regard representations such as ontologies as *boundary objects* [2] whose role is to support communication and negotiation over meaning between stakeholders within and across communities of practice.

Once an ontology has been constructed a population phase uses the ontology to describe web documents from a communal viewpoint. Two key questions which arise in this type of enterprise and that we address in this paper are: who develops the ontology? and how is the ontology population phase supported?

We believe that knowledge engineers are crucial in the ontology development phase. The main reason for this choice is that a careful design of the ontology is crucial to ensure the success of any particular document enrichment initiative. The ontology specifies the selected communal viewpoint, circumscribes the range of phenomena we want to deal with and defines the terminology used to acquire domain knowledge. In our experience small errors/inconsistencies in any of these aspects can make the difference between success and failure. Moreover, ontology design requires specialist skills which are normally not possessed by the members of our target user communities.

Our approach is to develop the ontology using a participatory design methodology. The ontology is developed during a series of face-to-face meetings between knowledge engineers, who are concerned with issues such as representational

consistency and completeness, and a representative group of the target community.

In contrast it is essential that ontological enrichment occurs without the aid of knowledge engineers. Unless enriched web resources are a "living archive" the resultant services will soon fall into disuse. In describing the APECKS personal ontology server Tennison and Shadbolt [20] make a case for "living ontologies".

In the rest of this paper we shall illustrate our approach, which we term *ontology driven document enrichment* [16], using a case study. We start by outlining the domain, the architecture of the application and one of the knowledge services that we created. We then describe the design of the ontology and four ways in which we support the ontology population process. Related work is briefly summarized before ending with some conclusions.

CASE STUDY AN OBSERVATORY ON LIFELONG LEARNING INITIATIVES

Case Study Background
In its Green Paper, 'The Learning Age', the UK Government set out its vision of 'a learning society in which everyone, from whatever background, routinely expects to learn and upgrade their skills throughout life.' One of the significant steps carried out by the UK Government to fulfil this vision was the creation of the University for Industry (Ufi) in the autumn of 2000. The overall goal for Ufi was to provide flexible learning packages which would improve the quality of life of individuals and to boost business competitiveness.

Promoting and supporting lifelong learning is a very difficult activity which requires knowledge of a number of disparate research areas including learning theory, organisation science and sociology. For the Ufi to be successful associated researchers and policy makers would need to discover and disseminate good practice on lifelong learning. It was decided that the main supporting mechanism for this would be a Web portal, termed the *National Observatory* (available at www.lifelonglearning.ac.uk), which was setup in the early part of 2000. By the time the Ufi was launched the observatory contained a number of resources including a bulletin board and a web based newsletter. The main resource was a 'Good Practice' database which held several hundred hand-coded summaries of articles describing lifelong learning initiatives. Although the database entries were highly regarded the text based search mechanisms provided a poor method of accessing relevant items.

Our goal in this project was to provide a semantic query service for lifelong learning researchers and policy makers who wanted to analyse relevant case studies, and for organisations that required help in understanding their learning needs.

Approach and Overall Design
The semantic query service was constructed collaboratively by knowledge engineers at the Knowledge Media Institute (KMi) within the Open University (OU), lifelong learning researchers at the International Centre for Distance Learning (ICDL) also at the OU and a number of external lifelong learning researchers. The lifelong learning researchers specified a number of questions that the observatory should be able to answer. The questions were categorised into three main themes deemed important by the lifelong learning research community. Each theme contained three or four sub-themes. The themes were:

- Widening participation,
- Organisational change, and
- Funding.

The specified questions were relatively broad and high level. For example, one of the questions associated with widening participation is "What techniques are needed to target the needs of socially excluded groups?" and one of the questions associated with organisational changes is "What strategies appear most effective in attracting SMEs to learning?".

The main concepts and relations within the themes and questions were used as the basis for an initial observatory ontology. The ontology was then expanded over a period of four months so that the formulated questions could be answered whilst ensuring that any new concepts and relations conformed to the view of the lifelong learning researchers.

The ICDL researchers then populated the ontology with instances which reflected the knowledge content of the learning initiatives in the Good Practice database. In this paper we describe how we supported these researchers in their population task.
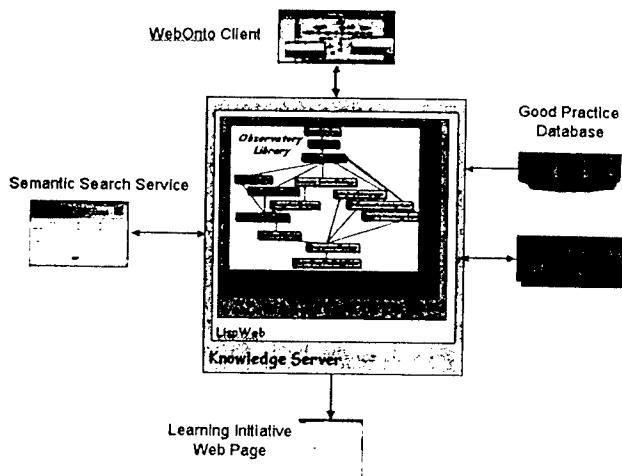
Architecture
The overall architecture of the system is shown in Figure 1. At the centre of the architecture is a knowledge server whose main role is to retrieve appropriate learning initiatives from the database from end-user queries. The main components of the server are as follows:

- *LispWeb* - a customised HTTP server [18] which offers a library of high-level Lisp functions to dynamically generate HTML pages.

- *WebOnto Server* - WebOnto [3], composed of a central server and a Java based client, enables users to collaboratively browse and edit knowledge models over the web.

- *OCML* - An operational knowledge modelling language [15], which provides the underlying representation for our ontologies and knowledge models.

- *Observatory Library* – a set of knowledge models which includes the observatory ontology used to index the learning initiatives in the good practice database.

Connected to the central server are:

- *The Good Practice Database* – a database containing several hundred summaries of documented examples of lifelong learning.

- *Named Entity Recognizer* – this uses the Marmot and Badger systems from Riloff [17] in combination with a regular expression matcher to support the automatic creation of OCML entities from text in web pages.

- *WebOnto Client* – a Java based client to the WebOnto server.

- *Semantic Search Service* – a service for retrieving learning initiatives from high level queries.

Figure 1. The architecture of the Observatory.
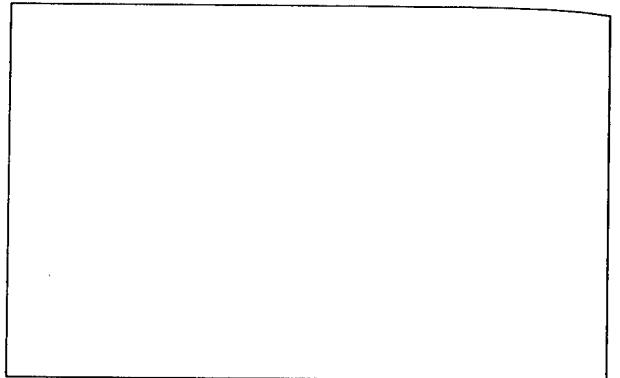
In contrast with other approaches to semantic annotation we decouple the knowledge structures from the web resources. This architecture allows us to provide multiple knowledge services, possibly for different communities of practice, over the same set of web documents. For example, a community of graphic designers may be interested in the typography and layout of a set of web pages whereas experienced website developers may be interested in the structure of the underlying HTML code. Another feature of this architecture is that the interfaces are directly connected to the ontology – there is no intermediate web crawling or compilation phase.

An Semantic Search Service
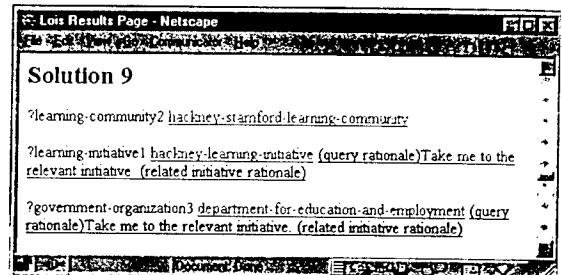
The semantic search service is designed to be easy-to-use by non-IT specialists and to provide answers to policy level questions. Figure 2 shows a screen snapshot of a web interface, constructed in Flash™, for finding learning initiatives according to the type of funder or the characteristics of the targeted learning community. In the figure the user is asking for a government funded learning initiative which involved a socially excluded community.

The query is run in OCML on the knowledge server. A set of rules link OCML knowledge items to relevant learning initiatives within the good practice database. Figure 3 shows the 9th (of 11) solutions. Each solution contains links to a knowledge item, a related learning initiative and links to an explanation of why they were returned. The explanation, shown in figure 4, describes why the target learning
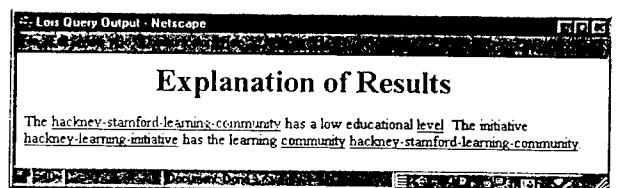
community, the members of the Stamford housing estate, were considered to be socially excluded.

Figure 2. A screen snapshot showing the query interface asking for a government funded learning initiative which involved a socially excluded learning community.

Figure 3. A screen snapshot showing the results of the query in figure 2.

Figure 4. The explanation generated for the query formulated in figure 2.

Ontology Design

As we outlined earlier in this paper, there were several constraints which had to be satisfied when creating the observatory ontology. The ontology had to characterise the domain such that a) the types of questions posed by the lifelong learning policy makers and researchers could be answered, b) there was a mapping to the existing database of learning initiatives, and c) the characterisation conformed to the viewpoint of the researchers.

We should emphasise the importance of the last constraint. It was important that all of the 'observatory team' understood and had ownership of the ontology. Also as outlined in [5] in their analysis of the KA^2 initiative, and in [11] in their description of a SHOE case study, ontology development and representing specific resources are intertwined activities.

Slot Name	Documentation	Value Type
Has-title	The title of the initiative.	A string.
Has-location	The location of the initiative. This includes information on the social geography of the area.	A learning related location.
Has-initiative-date	The starting date for the initiative.	An integer representing the year (used within the existing database).
Has-rationale	The underlying rationale for the initiative.	A rationale for learning.
Has-funder	The funding organisation or person.	Either an organisation or person.
Other-involved-parties	Organisations, individual people and communities which take part in the initiative.	Either an organisation, generic-organisation, person or community.
Has-learner	The target audience for the initiative.	A learning-community.
Has-deliverable	The tangible results of the initiative.	A document, technology or organisation (a project may create a new organisation).

Table 1. The definition of the learning-initiative class.

The conceptual design of the ontology was developed in a series of weekly meetings involving the whole observatory team. A number of the meetings included external policy makers and lifelong learning researchers the end users of the observatory. Once an initial version of the ontology had been implemented in WebOnto a sample population phase followed. In the early part of this phase the knowledge engineers and populators collaboratively coded 10 practices in the database. Coding difficulties would either result in immediate changes to the ontology or be logged and changed later. The populators then coded a further 20 practices on their own reporting problems by phone or email. Additionally, the team continued to meet face-to-face weekly to discuss problems and changes to the ontology. These discussions would invariably result in changes to the ontology and occasionally in the addition of new tools. WebOnto's architecture meant that any changes to the ontology (or to WebOnto itself) were immediately available to the populators.

Because the domain, the intersection of learning and social policy, was relatively broad we created and reused a number of higher level ontologies. Figure 5 shows the structure of the relevant portion of our library. The arrows indicate that an ontology *uses* its parent ontology (i.e. inherits all of the OCML entities). The observatory knowledge base currently indexes several hundred good practice case studies.

The core of the ontology is based on a learning initiative class which represents a single documented case in the Good Practice database. As we can see from table 1 the main attributes of learning initiatives are the title, location, date, learning rationale, funders, organisations involved, target learners and the tangible results. Often the descriptions of
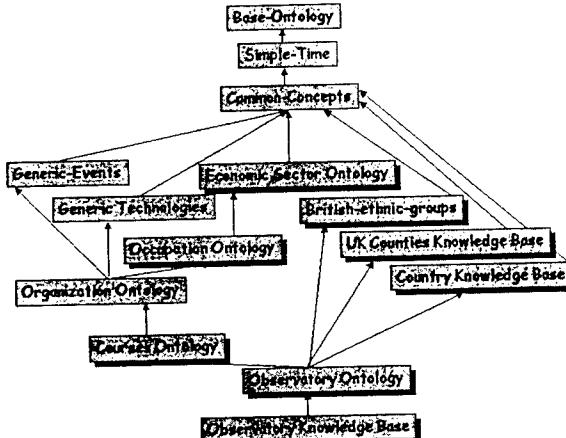
Figure 5. Each node represents an ontology or knowledge base. The shadowed nodes indicate knowledge models which were created during the project.

learning initiatives describe generic rather than specific entities. For example, involved parties are sometimes described using phrases such as "a local college" or "a few mechanical engineering SMEs". These types of statements are captured using the generic-organisation class – the instances of this class are classes of type organisation.

The other key definition within the ontology is the learning-community class. We do not have space here to include this definition but the key attributes include the affiliation, ethnic group, occupation, gender, age, skill level and dependents. This broad range of slots reflects the diverse attributes that learning and social policy researchers argue can affect access to learning within a community.

Ontology Population

Although WebOnto is primarily aimed at expert model builders we have recently provided a number of tools to allow non-experts to populate ontologies. Integrating support for ontology creation and population within WebOnto contrasts with the approach taken in tools such as Protégé [8] where ontology construction and population are separated.

Help in WebOnto is provided in four main ways:

- *Multiple visualizations* – aid in reviewing what has been created.

- *Automatically generated instance forms* – support the addition of instances.

- *Knowledge items from web pages* – information extraction techniques have been coupled with direct manipulation techniques to enable OCML entities to be created from web pages.

- *Automatic type checking* – automatically checking for undefined values and constraint violations.

Multiple Visualizations

The use of visualizations has long been acknowledged to be important in the creation of knowledge models [4]. The key is to provide support for high level or coarse grained views which are tightly coupled to multiple fine grained views. WebOnto provides high level graphical views of class hierarchies tied to fine grained views which use font and colour to differentiate between types of OCML entities.

A significant task where visualizations can aid populators is in validation. Populators need easy-to-read detailed descriptions of the entered knowledge structures. Often the ontological enrichment of a web resource is based on a single class or on a set of related classes - typically class A constrains the type of a slot in class B. Specific resources are represented by a set of connected instances. This heuristic provides the basis for the design of a connected instances visualization. This view displays all the instances connected to a selected instance. Figure 6 shows a connected instances view of the `hackney-learning-initiative`. Within this view instance names are shown in black, classes in green and slot names in a light blue. Knowledge items which were entered by the user are shown in bold. Any slot values which are instances are expanded. Each instance is picked out using background shading.

Within figure 6 we can see that the `hackney-learning-initiative` is an instance of `learning-initiative`. The `has-location` slot has the value `hackney-li-location` which is an instance of `learning-related-location`. The `has-premises-type` slot of `hackney-li-location` has two values - the classes `community-centre-premises` and `library-premises`. The `department-for-education-and-employment` instance was created by the user but the values of its slots were not. The depth of the inline expansion is defined by the user. Selecting any instance in the view creates a new connected instances view. We elected to provide these

visualizations in HTML format so that they could easily be printed and viewed in hardcopy format – a requirement from the lifelong learning researchers populating the ontology.
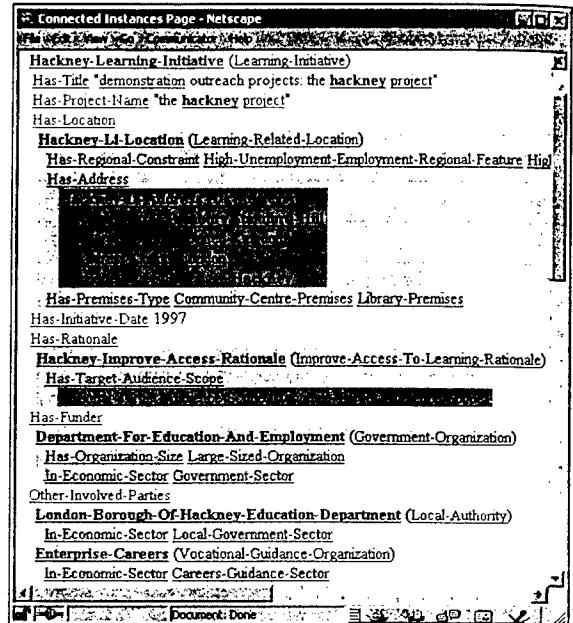
Figure 6. A screen snapshot of a connected instance based visualization. Items in bold were defined by the ontology populators. Colour coding distinguishes between instances, classes and relations. Individual instances are picked out with background shading (enhanced for this paper).

Automatically Generated Instance Forms
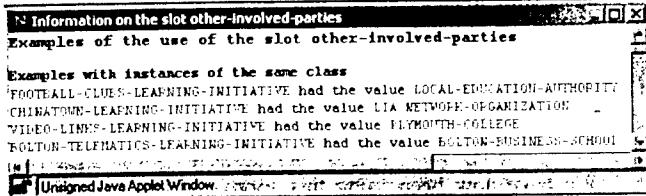
Many errors in semantic annotation occur because of errors in naming existing entities and in selecting the class of new instances [5]. The forms in WebOnto seek to alleviate this by prompting users with the names of relevant knowledge items.

Figure 7. A screen snapshot showing an automatically generated learning-community instance edit form.

An example of an automatically generated form for editing an instance of a learning community is shown in figure 7. Each slot is displayed as a row. The slot name is a button

34

which displays examples of the values that have been given to the slot within other instances. Figure 8 shows the result of selecting the 'other-involved-parties' button.
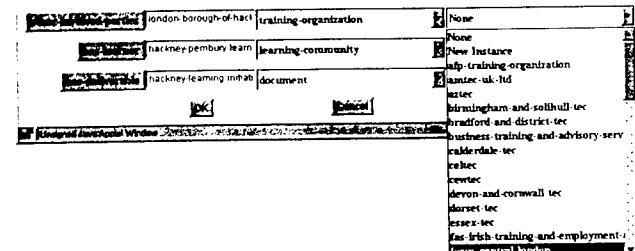
Figure 8. A screen snapshot of the help given when . selecting the other-involved-parties button of the form shown in figure 7.

The second column is a simple text field into which the name of a value can be entered. Within our underlying knowledge modelling language OCML [15] slots can be typed using a class or a combination of classes (e.g. (or organization person)). These classes and all of their descendants appear in alphabetical order the third column of the form. Figure 9 shows a user selecting the training-organization class for the other-involved-parties slot. When a class is selected the instances of the class appear in the menu in the fourth column. Figure 10 shows a user selecting the focus-central-london instance.

Figure 9. A screen snapshot showing a user selecting the training-organization class for the other-involved-parties slot of a learning-initiative instance.

Figure 10. A screen snapshot showing a user selecting the focus-central-london instance for the other-involved-parties slot of a learning-initiative instance.

The forms here are in some respects similar to the forms provided in Protégé-II [8]. The key difference is that instance forms in WebOnto are generated directly from the ontology whereas the forms in Protégé-II use an extra set of form specific definitions. The extra information means that the generated forms can use non-trivial layouts but require an extra compilation cycle. Within WebOnto any changes to the ontology are immediately reflected within the forms.

Knowledge Items from Web Pages

As with the majority of our application domains a proportion of the elements referred to in the observatory knowledge base appear within web documents, specifically, within the entries within the Good Practice database. To aid in the generation of knowledge items from web documents WebOnto contains an interface to a *named entity recognizer*. Named entity recognizers are used to extract items of a pre-specified type from grammatical text. We currently use Marmot [17] to tokenize the text (identifying the nouns) and Badger [17] extract the named entities. We also use a regular expression matcher (written in Perl) because Badger relies on the input text being composed of grammatical sentences (nouns, verbs and prepositions) and this is not always the case for the learning initiatives.

The interface between OCML and the entity recognizer is implemented with two types of constructs: pattern definers and templates. A pattern definition consists of the name of an OCML class or instance and a set of strings which represents patterns using the using the standard notation for regular expressions. The pattern for a college is:

```
(def-pattern college
"(capital_word)* College"
"(capital_word)* College of (capital_word)*")
```

Within the observatory case we have created patterns to identify organizations, ethnic groups, peoples' names and dates.

Templates are used to create new OCML structures from the results of the entity recognizer. Currently three types of template are used:

- *New class instance* – this specifies how text can be used to create a new instance of a class.

- *New class subclass* – this specifies how subclasses of a class can be created.

- *Fill instance* – specifies how an existing instance is filled.

A template consists of the name of a class or instance, a list of variables and the template body. Within the template body variables are denoted by the prefix '$', and, $class-name and $instance-name are special variables which represent the name of the class and instance respectively. The template used to create the hackney-community-college instance was:
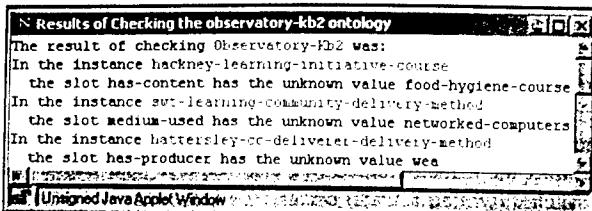
```
(def-new-instance-template organization (name)
(def-instance $name $class-name))
```

Other examples of how we have combined our knowledge modelling infrastructure with information extraction technologies can be found in [21].

Automatic type checking

The late 80s and 90s saw a considerable effort into creating tools for validating and verifying knowledge bases [14]. We have found that even relatively simple tools can aid ontology populators. OCML contains a general purpose real-time

35

constraint checker. The output of checking the observatory knowledge base is shown in figure 11. Any of the instances or relations shown in figure 11 can be inspected by simply clicking on them.

Figure 11. The result of carrying out consistency checking on the observatory knowledge base. Items within the knowledge base are highlighted using colour and can be selected and inspected. Colour is used to distinguish between instances and relations.

RELATED WORK

The KA^2 initiative [1] shares a number of commonalities with our work. As with the case described here the aim of KA^2 is to allow a community to build a knowledge base collectively, by populating a shared ontology. The knowledge base is constructed by annotating web pages with special tags, which can be read by a specialised search engine cum interpreter, Ontobroker [6]. In this paper we have described and approach which learns from the early problems reported in that initiative [5].

A number of tools such as the CEDAR toolkit [10] and OntoAnnotate [19] provide support based on a web browser integrated with a view of an ontology. The CEDAR annotation tool allows segments of text from web pages to be associated with OCML structures stored on a WebOnto server. Within OntoAnnotate text can be selected from a web page and dragged to fill in the value of an instance. OntoAnnotate also contains mechanisms for managing annotations after an ontology is altered, a text pattern matcher similar to the one described here and links to an ontology based information extraction system. Both the CEDAR annotation tool and OntoAnnotate are designed to use ontologies to annotate web pages whereas goal of the technologies described here are to facilitate the population of ontologies. Hence, rather than creating a separate tool we elected to extend WebOnto thus tightly coupling the ontology development and resource description activities.

In terms of the underlying architecture, as we stated earlier the main difference between our approach and the above approaches to adding semantic information to web pages is that we decouple the web pages from the knowledge model. We should state however that the WebOnto server is now able to export knowledge models in OIL RDF syntax [7]. This facility was used to incorporate parts of our library into an OIL based ontology server as part of a dynamic link service (see [12] for more details).

CONCLUSIONS

In this paper we have described how ontologies can support knowledge sharing within communities of practice. To be successful it is important that all stakeholders are able to participate in the ontology development process and that this process is ongoing and integrated with ontology population. Moreover, ontology population requires support from a mixture of technologies and as far as possible should be integrated into existing working practices.

We have now been using this approach over a number of years in a variety of projects, in domains ranging from managing best practice in the aerospace industry, to supporting the application of medical guidelines. Our experience to date suggests that our approach appears to provide both the technology and the methodological framework required to minimize risk and ensure the participating community's acceptance.

REFERENCES

1. Benjamins, R., Fensel, D. and Gomez Perez A. Knowledge Management through Ontologies. In U. Reimer (editor), *Proceedings of the Second International Conference on Practical Aspects of Knowledge Management.* Basel, Switzerland, 29-30 October, 1998.

2. Bowker, G. C., and Star, S. L. *Sorting Things Out: Classification and its Consequences.* MIT Press: Cambridge, Mass., 1999.

3. Domingue, J. Tadzebao and WebOnto: Discussing, Browsing, and Editing Ontologies on the Web. In B. Gaines and M. Musen (editors), *Proc 11th Knowledge Acquisition for Knowledge-Based Systems Workshop,* Banff, Canada, April, 1998.

4. Eisenstadt, M., Domingue, J., Rajan, T. and Motta, E. Visual Knowledge Engineering. *IEEE Transactions on Software Engineering* Special Issue on Visual Programming, 16(10), 1164-1177, October, 1990.

5. Erdmann, M., Maedche, A., Schnurr, H.P., Staab, S. From Manual to Semi-Automatic Semantic Annotation: About Ontology-based Text Annotation Tools. *COLING-2000 Workshop on Semantic Annotation and Intelligent Content,* Centre Universitaire, Luxembourg, 5-6 August, 2000

6. Fensel, D. Decker, S., Erdmann, M. and Studer, R. Ontobroker: The very high idea. *Proc 11th Annual Florida Artificial Intelligence Research Symposium (FLAIRS-98)*.

7. Fensel, D., Horrocks, I., van Harmelen, F., Decker, S., Erdmann, M. and Klein, M. OIL in a Nutshell, *Proc. 12th Int'l Conf. Knowledge Engineering and Knowledge Management*, Lecture Notes in Computer Science, vol. 1937, 2000, Springer Verlag, New York, 1-16

8. Grosso, W. E., Eriksson, H., Fergerson, R. W, Gennari, J. H., Tu, S. W., and Musen, M. A. Knowledge Modeling at the Millennium (The Design and Evolution of Protege-2000). *Proc 12th Knowledge Acquisition Workshop*, Banff, Alberta, Canada, October, 1999.

9. Gruber, T. R. A Translation Approach to Portable Ontology Specifications. *Knowledge Acquisition*, 5(2). 1993.

10. Hatala, M., and Hreno, J. Annotation of documents with knowledge model concepts. *Enrich Report ID44-O-K*, 2000. (Available at http://kmi.open.ac.uk/projects/ enrich/id44.pdf).

11. Heflin, J. Hendler, J., and S. Luke. Applying Ontology to the Web: A Case Study. In *Proc of the International Work-Conference on Artificial and Natural Neural Networks, IWANN'99*.

12. Kalfoglou, Y. Domingue, J., Carr, L., Motta, E., Vargas-Vera, M. and Buckingham Shum, S. On the integration of technologies for capturing and navigating knowledge with ontology-driven services. *KMi Technical Report No. 106*, April, 2001.

13. Lave, J. and Wegner, E. *Situated Learning: Legitimate Peripheral Participation*, Cambridge University Press, Cambridge, UK, 1991.

14. Meseguer, P. and Preece, A. Verification and validation of knowledge-based systems with formal specifications. *The Knowledge Engineering Review*, 10(4), 331-343, 1995.

15. Motta E. *Reusable Components for Knowledge Models*. IOS Press, Amsterdam, 1999.

16. Motta, E., Buckingham Shum, S. and Domingue, J. Ontology-Driven Document Enrichment: Principles, Tools and Applications. *International Journal of Human Computer Studies*. 52(5), 1071-1109, 2000.

17. Riloff, E. An Empirical Study of Automated Dictionary Construction for Information Extraction in Three Domains. *The AI Journal*, 85, 101-134, 1996.

18. Riva, A. and Ramoni, M. LispWeb: a Specialized HTTP Server for Distributed AI Applications, *Computer Networks and ISDN Systems*, 28 (7-11), 953-961, 1996.

19. Staab, S., A. Mädche, S. Handschuh. An Annotation Framework for the Semantic Web. In: S. Ishizaki (ed.), *Proc. of The First International Workshop on MultiMedia Annotation*. January, 30 - 31, 2001. Tokyo, Japan.

20. Tennison, J. and Shadbolt, N. R. APECKS: a Tool to Support Living Ontologies. *Proc. of the 11th Banff Knowledge Acquisition Workshop*. Banff, Alberta, Canada, April 18-23, 1998.

21. Vargas-Vera, M., Domingue, J., Kalfoglou, Y., Motta, E. and Buckingham-Shum, S. Template-driven information extraction for populating ontologies. *Proc of the IJCAI'01 Workshop on Ontology, Learning*. Seattle, WA, USA 2001.

22. Wenger, E. *Communities of Practice: Learning, Meaning, and Identity*. Cambridge University Press: Cambridge, 1998

Representing Roles and Purpose

James Fan
Ken Barker
Bruce Porter
Department of Computer Sciences
University of Texas at Austin
Austin, TX 78712 USA
{jfan, kbarker, porter}@cs.utexas.edu

Peter Clark
Knowledge Systems
Boeing Math and Computing Technologies
m/s 7L66, PO Box 3707, Seattle, WA 68124 USA
peter.e.clark@boeing.com

Abstract

Ontology designers often distinguish *Entities* (things that are) from *Events* (things that happen). It is not obvious how this division admits *Roles* (things that are, but only in the context of things that happen). For example, *Person* might be considered an Entity, while *Employee* is a Role. A Person remains a Person independent of the Events in which he participates. Someone is an Employee only by virtue of participating in an Employment Event. The problem of how to represent Roles is not new, but there is little consensus on a solution. In this paper, we present an ontology that finds a place for Roles as well as a representation that allows Roles to be related to Entities and Events to express the teleological notion of purpose.

Keywords
roles; ontologies; teleology

Background
One of the challenge problems in DARPA's Rapid Knowledge Formation project requires subject matter experts (SME's) with little training in knowledge engineering to build a knowledge base of information from a college-level textbook on cell biology. The knowledge base will be evaluated on its ability to answer a large set of questions drawn from standard test banks, such as the GRE subject exam (a graduate school admissions test) and questions from the end of book chapters. Our goal is to develop ways to help SME's succeed.

One of our chief concerns for this challenge problem is developing good ways to represent the wide variety of types of knowledge expressed in textbooks. Many knowledge engineering projects can focus on just a few types of knowledge – for example, building a knowledge base about aircraft might focus exclusively on structure and partonomy – because the questions they are intended to answer are relatively limited.

However, textbook knowledge and the questions we expect to be presented are quite varied.

There are many types of knowledge conveyed in textbooks, of course, and this paper focuses on just one – how to represent the roles and purposes of entities – which has been problematic for knowledge engineering. Although ontologies typically distinguish *Entities* (things that are) from *Events* (things that happen), it is not obvious how this division admits *Roles* (things that are, but only in the context of things that happen).

The source of the problem lies in the distinction between intrinsic and extrinsic features. **Intrinsic** features, such as *shape* and *size*, describe an entity in isolation. In contrast, **extrinsic** features describe an entity relative to other entities and events. For example, *used to strike nails* is an extrinsic feature of a hammer because it relates a hammer to nails and striking. Efforts to represent concepts using only intrinsic features have largely failed [11], especially for representing artifacts [3].

Although the distinction between intrinsic and extrinsic properties is more spectral than black-and-white, it is important to distinguish the many cases that fall into the uncontroversial extremes because they differ in significant ways. For example, an entity's intrinsic features (such as *age*) may change over time, but they are always applicable to the entity. In contrast, extrinsic features (such as the *salary* of a person) may become completely inapplicable. Moreover, unlike intrinsic features, an entity's extrinsic features may be contradictory, such as the *salary* of a person with multiple jobs. For these reasons, most psychological research on concept representation distinguishes between an entity's extrinsic and intrinsic features [11].

From these distinctions (and others we discuss later) we draw three conclusions. First, the distinction between intrinsic and extrinsic features is important; a knowledge-based system that ignores their differences might draw incorrect inferences. Second, the roles and purposes of an entity are necessarily extrinsic features, *i.e.* they relate an entity to other entities and events. Finally, roles should be reified in any

knowledge representation scheme. The representation of a role consists of those extensional features of an entity that are due to its participation in some event.

The Difference between Roles and Entities

There has been considerable research on roles in data and knowledge modeling, as we summarize below. The research offers two key insights. First, entities and roles are not related taxonomically, at least not in any simple way; "Neither the roles of the real world nor the entities of the real world are a subset of the other" [2]. Guarino offers two criteria for distinguishing roles from entities [6]: (1) a role is "founded" and (2) a role lacks "semantic rigidity". Something is founded if it is defined in terms of relationships to other things. Something is semantically rigid if its existence is tied to its class; that is, if in ceasing to be of kind X, it ceases to be. For example, the concept *food* is a role because it meets these criteria, as follows:

- *Food* is Founded: The properties of food, such as *eaten-by* and *nutritional-value*, are extrinsic properties of the entity filling the role of food – they relate that entity to others participating in the *eating* event, such as the *eater*, and they are applicable only in that context.
- *Food* lacks Semantic Rigidity: An entity that might fill the role of food retains its identity (i.e. its primary class membership) outside the context of the role. For example, a grasshopper is food when eaten by a bird, but when it is no longer considered food, it is still a grasshopper.

In contrast, these criteria tell us that *person* is an entity, and not a role, for the following reasons:

- *Person* is not Founded: The properties of a Person, such as *age* and *sex*, are intrinsic features. They are defined independently of other entities and events.
- *Person* has Semantic Rigidity: when a Person ceases to be a Person, she ceases to be.

Roles in Use

There would be little value in devising a complicated representation for roles if they do not occur frequently. To gauge how common roles are, we ran a simple experiment using English word lists.

We first extracted from a large online wordlist [1] nouns that end in "-ee", "-er", "-or" or "-ist". These endings, such as *employee*, *driver*, *actor* and *pianist*, are good cues for roles . We pruned this list to only those whose stems are also stems of base verb forms. The result was a list of more than 5,000 candidate role names. To determine how many of these might actually represent role concepts, we sampled 109 at random. Based on the tests of foundedness and lack of semantic rigidity, 101 of the sampled nouns represented role concepts. Given that there are 74,577 unique noun entries in the Collins wordlist, this experiment suggests that at least 6% of nouns may represent role concepts (at 95% con-

fidence). The suffix filter would miss many potential roles, making this number an underestimate of roles in use.

As a second experiment, we checked a list of the most frequently used nouns in the the British National Corpus [7]. 200 of the roughly 3,000 most frequent nouns represented role concepts, meaning that role concepts also account for 6% of the most common nouns. (Previous work [15] has established that there is considerable overlap among the more frequent words in different corpora).

A Knowledge Representation for Roles

Roles are easy to identify yet they are difficult to represent. They are not merely reified names for the participants in events. Rather, roles have their own characteristics which require that they be treated differently than entities in a knowledge representation scheme. Steimann [14] identified fifteen characteristics of roles, which we've distilled into these four:

1. Roles are created and destroyed dynamically. Because a role represents the extrinsic features of an entity due to its participation in an event, the role is created when the participation begins. If the entity stops participating, the role may cease to exist and all its properties may no longer hold.

2. A role can be transferred between entities. For example, the role of *manager* can be transferred from one person to another. Note that many of the role's features are transferred without change, while others must be re-computed in light of the new entity playing the role. For example, if a person earns a 20% bonus for being manager, then the *salary* feature must be recomputed should that role be transferred.

3. An entity may play different roles simultaneously, for example a *person* may be both an *employee* and an *employer*.

4. Entities of unrelated types can play the same role. For example, both a cracker and a grasshopper can play the role of *food*.

These four characteristics impose requirements on any knowledge representation scheme for roles. The next section assesses past approaches to representing roles in light of these requirements.
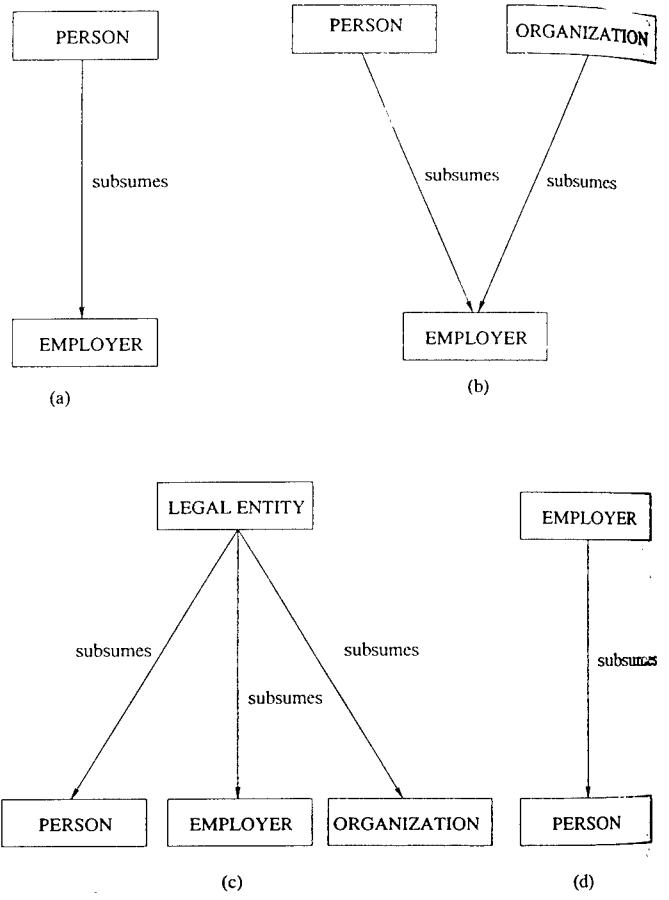
Previous Approaches to Representing Roles

According to Steimann [14], previous research produced three basic approaches to representing roles. The first approach represents a role as nothing but a label assigned to a participant in an event. For example, the *employer* role labels the agent of an *employ* event. This approach is simple, but it fails to reify roles as distinct from entities (instead combining intrinsic and extrinsic properties into a single representation of an entity), which is problematic as we discussed in Section *Background*.
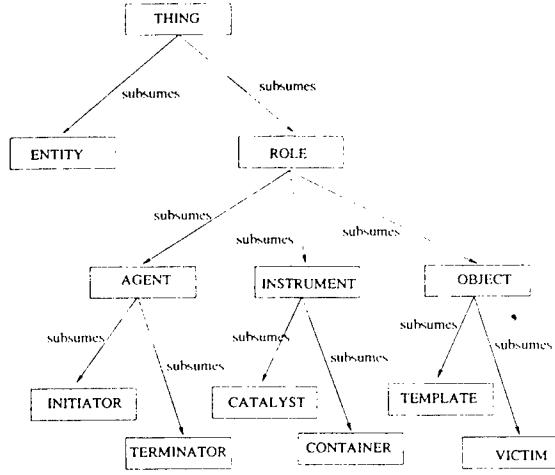
Assuming that these labels can be assigned and retracted dynamically (as entities play roles and later drop them), this approach meets the first requirement ("roles are dynamic") and the second requirement ("roles can be transferred"). The approach does not meet the third requirement ("entity can play multiple roles") because roles are not reified as predicates with arguments. Rather in this approach roles are simple propositions. Consequently the extrinsic features of an entity can clash due to the entity's participation in different events. For example, if a person has two jobs, then the two employee roles she plays will give her two different salary values. If a query about her salary is posed, then it is not clear which value should be returned. Finally, this approach meets the fourth requirement ("entities of different types can play the same role") as there are no constraints on assigning labels to entities.

The second approach, used by Sowa [13] and Uschold [8] , reifies roles and distinguishes them from entities (in that roles represent extrinsic features and entities represent intrinsic ones), then combines the two types of concepts into a single hierarchy. They can be combined in either of two ways; both are problematic [14]:

1. the roles are subtypes of entities. For example, the role *employer* would be a subtype of the entity *person*, as shown in Figure 1 (a). This becomes problematic when trying to meet the fourth requirement ("entities of different types can play the same role"). To illustrate, consider extending the hierarchy to assert that an *employer* may be either a *person* or an *organization*, as shown in Figure 1 (b). This taxonomic structure says that every *employer* is **both** a *person* and an *organization* – not what we intended. In an effort to represent the disjunction of person and organization, we create a new type, *legal-entity*, which subsumes *person* and *organization*, as shown in Figure 1 (c). Because an *employer* must be a *legal-entity*, *employer* must be a sibling of *person* and *organization*. This does not capture our original assertion that an *employer* is either a *person* or *organization*.

2. the roles are supertypes of the entities that play them. For example, *employer* would subsume *person*, as shown in Figure 1 (d). This is clearly wrong because not every person is an employer. Moreover, it fails to meet the first requirement ("roles are dynamic"), unless the subtype relationship between entities and roles is dynamic. To avoid this paradox, some knowledge representation schemes take exactly that approach [5]. (See also *qua-classes* of KL-ONE [4] and the *existence subclass* of SDM [9] and MERODE [12].) These schemes have the restriction that a role exists if and only if an entity is actually playing that role. This restriction makes it difficult to use role concepts to represent an entity's purpose, as we discuss in Section *Representing Purpose using Roles*.

Figure 1: **Taxonomy paradox. If roles and entities are combined into one hierarchy, none of the hierarchies above fully captures the intended information: an *employer* can be either a *person* or an *organization*.**

Figure 2: A partial listing of our role hierarchy. *Role* is a sibling of the top-level concept *Entity*, and it has several subtypes, such as *Agent*, *Instrument* and *Object*.

The third approach represents a role as an "adjunct instance" of an entity. An adjunct instance here is a distinct instance of a role class that is coupled with the instance of an entity; the role instance does not exist independent of that entity. We adopt this basic approach, as we discuss next.

Our Approach

We built a representation of roles using the adjunct instance approach to express what an entity is designed to do (its purpose), and what an entity actually does (its role). In our representation, roles are types independent of entities. An instance of a role is played by an instance of an entity; every instance of a role exists along with an instance of an instance of an entity. The role instances are connected with the entity instances through two composition methods described in Section *Role composition*. In order to retrieve values of properties that belong to a role, we need to first retrieve the role from the entity with which it is composed, and then we can retrieve the values of properties from the role.

In keeping role concepts separate from entities, the problem arises of where in the taxonomy role concepts belong. In order to avoid the taxonomy paradox described above, we make Role a sibling of the top-level Entity concept (see Figure 2).

For our project, we are mainly concerned with general role concepts. Examples of such general roles include:

- *Agent*: the role played by an entity performing or responsible for an event. More specific Agent roles include *Initiator*, *Terminator*, *Creator*, *Interpreter*.
- *Instrument*: the role played by an entity used in some event. More specific Instrument roles include *Container*, *Catalyst* and *Connector*.
- *Object*: the role played by an entity acted upon in an event.

More specific Object roles include *Template* (Object of a *Copy* event), *Idol*, *Input* and *Victim*.

Our solution is implemented in the KM language [10]. KM is a frame-based language with clear first-order logic semantics. To avoid issues of KM syntax, we will illustrate our solution with examples expressed in first-order logic.

Representing Purpose using Roles

The reification of roles (as distinct from entities) provides a convenient way to represent the teleological notion of purpose. We represent an entity's purpose as the default role(s) it plays. For example, the default role of *cereal* is *food* (i.e. to be the object eaten by people) and the default role of a *cup* is to *contain* (i.e. to be the instrument of containment). These entities are artifacts, which typically have a clear purpose, but natural entities are often ascribed a purpose, too. For example, one purpose of a human hand is to grip.

Role composition
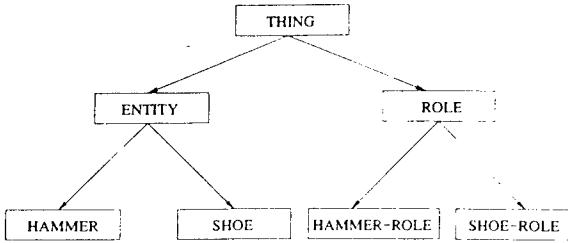
In our approach, roles are types and instances of roles are played by instances of entities; an instance of a role requires a corresponding entity. The correspondence is established with 2 relations: *played-by* and *purpose*. When an entity is related to a role with one of these relations, we say they are composed together. (In our knowledge representation scheme [10], such compositions have inferential ramifications, which are outside the scope of this paper.)

The *played-by* composition represents that an entity is actually participating in an event. (In our knowledge representation scheme, this can be asserted to hold in a temporally bounded state.) For example, when a hammer is participating in a hammering event, the instrument role for the hammering event is *played-by* the hammer. (Equivalently, the hammer *plays* the instrument role for the hammering event.)

The *purpose* composition represents a role that the entity is intended to play, but says nothing about whether it is actually doing so. For example, the purpose of a hammer is to be the instrument of a hammering event, which is true even when the hammer is not participating in any hammering event.

Both *played-by* and *purpose* are many-to-many relations, which means that an entity can play multiple roles and a role can be played by multiple entities. Both relations are fluent, which means that an entity can dynamically acquire and relinquish roles.

It is possible and common for an entity to play a role that is the purpose of another entity. For example, the purpose of a *hammer* is to be the instrument of a *hammering* event; a *shoe* might also "play the role" of a hammer – or more accurately, play the role that is the purpose of a hammer. In order to avoid this representational gymnastics, we could reify the purpose of a hammer as a *hammer-role* so that the shoe plays a hammer-role. Note that a shoe cannot "play" a hammer because hammer is an entity, not a role.

Figure 3: The duplication of the entity hierarchy in the role hierarchy caused by the promiscuous reification of the purpose of entities.

Although we *could* reify hammer-role, that leads to a potential problem. Reifying the purpose of entities promiscuously will result in duplication of the entity hierarchy in the role hierarchy (Figure 3). The duplication problem is inherit in any representation of purpose. In practice, however, it is not a serious issue because most roles need not be reified. Our criteria is to reify only those roles, such as *container*, that are likely to be played by many different kinds of entities, not just those entities whose purpose is to play the role.

Non-reified roles are specialized instances of generic roles, and they are left unnamed. Specialization is accomplished through the addition of properties or constraints on an instance of the generic roles. As an example of composition, the purpose of a hammer might be represented as follows:

$$\forall x\, isa(x, Hammer) \rightarrow$$
$$\exists y, z\; isa(y, Instrument) \land$$
$$isa(z, Hammering) \land purpose(x, y) \land in - event(y, z)$$

The Skolem variable y is an example of a non-reified role.

By using a combination of *purpose* composition and non-reified role concepts, we can avoid the problem of duplicating the entity hierarchy in the role concept hierarchy. For example, a representation of using my shoe as a hammer would be:

$$\exists p, h\; isa(myShoe, Shoe)$$
$$\land isa(h, Hammer) \land plays(myShoe, p) \land purpose(h, p)$$

The Skolem instance p is a non-reified role denoting the purpose of a hammer. It is used to express that $myShoe$ plays that role. That is, $myShoe$ plays the role which is the purpose of a hammer.

Conclusion

The distinction between *entities* (things that are) and *events* (things that happen) is clear and common in ontologies, but it's decidedly less clear how to handle *roles* (things that are, but only in the context of things that happen). Although roles are often confused with entities, and mixed together in a single hierarchy, we draw from the data modeling literature an operational distinction between them. Using this distinction we determine that roles are frequently used in English text, accounting for more that 6% of the most common nouns. We describe our representation in which roles are reified, and instances of roles are composed with the entities that participate in them. Finally, we show how this representation can be easily extended to include the teleological notion of the purpose of entities.

REFERENCES

1. *Collins English Dictionary*. William Collins Sons Co. Ltd., 1979.

2. C. Bachman and M. Daya. The role concept in data models. In *Proceedings of the 3rd International Conference on VLDB*, pages 464–476, 1977.

3. R. Barr and L. Caplan. Category representations and their implications for category structure. *Memory and Cognition*, 15:397–418, 1987.

4. R. Brachman and J. Schmolze. An overview of the KL-ONE knowledge representation system. *Cognitive Science*, 9:171–216, 1985.

5. J. Odell C. Bock. A more complete model of relations and their implementation: Roles. *Journal of Object-Oriented Programming*, 11:51–54, 1998.

6. N. Guarino. Attributes and arbitrary relations. *Data and Knowledge Engineering*, 8, 1992.

7. A. Kilgarriff. BNC database and word frequency lists.

8. S. Moralee M. Uschold, M. King and Y. Zorgios. The enterprise ontology. *The Knowledge Engineering Review*, 13, 1998.

9. D. McLeod and M. Hammer. Database description with SDM: A semantic database model. *ACM Transactions on Database Systems*, 6(3):351–386, 1981.

10. B. Porter and P. Clark. KM - the knowledge machine: Reference manual. Technical report, University of Texas at Austin, 1998.

11. E. Smith and D. Medin. *Categories and Concepts.* Harvard University Press, Cambridge, MA, 1981.

12. M. Snoeck and G. Dedene. Specialization and role in object oriented conceptual modeling. *Data and Knowledge Engineering*, pages 171–195, 1996.

13. J. Sowa. *Conceptual Structures: Information Processing in Mind and Machine.* AddisonWesley Publishing Company, New York, 1984.

14. F. Steimann. On the representation of roles in object-oriented and conceptual modelling. *Data and Knowledge Engineering*, 35(1):83–106, 2000.

15. S. Delisle T. Copeck, K. Barker and S. Szpakowicz. More alike than not-an analysis of word frequencies in four general-purpose text corpora. In *Proceedings of the Fourth Conference of the Pacific Association for Computational Linguistics*, pages 282–287, 1999.

Learning Hierarchical Task Models by Defining and Refining Examples

Andrew Garland and Kathy Ryall and Charles Rich
Mitsubishi Electric Research Laboratories
{garland,ryall,rich}@merl.com

Abstract

Task models are used in many areas of computer science including planning, intelligent tutoring, plan recognition, interface design, and decision theory. However, developing task models is a significant practical challenge. We present a task model development environment centered around a machine learning engine that infers task models from examples. A novel aspect of the environment is support for a domain expert to refine past examples as he or she develops a clearer understanding of how to model the domain. Collectively, these examples constitute a "test suite" that the development environment manages in order to verify that changes to the evolving task model do not have unintended consequences.

Keywords

programming by demonstration, knowledge acquisition

INTRODUCTION

Many fields of computer science — planning, intelligent tutoring, plan recognition, interface design, and decision theory to name a few — get a lot of leverage from applying general-purpose algorithms to domain-specific task models. This approach gives rise to the notorious *knowledge acquisition bottleneck*: developing an accurate domain model is a significant engineering obstacle. In this paper, we present a development environment that can ease the task model acquisition process. The environment combines direct model editing, machine learning based upon annotated examples, and model verification through regression testing. The learning techniques and most of the other major components of this environment are in place; however, the graphical front-end is still under development.

In this work, the problem of developing task models is considered in the context of the Collagen [18, 17] system. In Collagen, a collaborative interface agent engages in dialogs with a user to jointly achieve tasks. Collagen is an implementation of the SharedPlan theory of collaborative discourse [7], in which the agent's behavior is driven by general-purpose

algorithms for discourse interpretation [13], plan recognition [10], and action selection [11]. In order to apply these algorithms in a given domain requires constructing an explicit, declarative model of the underlying task structure.

Since Collagen task models are hierarchical, a domain expert must decide how to divide tasks into subtasks, which involves choosing the best abstractions to represent intermediate goals. The choice of intermediate goals is especially important for collaborative agents because the agent must be able to discuss how to accomplish tasks in a way that is intuitive to the user. Determining an appropriate set of intermediate goals (as well as the number and type of parameters for each) can be extremely difficult for a domain expert.

Our approach to acquiring task models is based on the conjecture that it is often easier for people to generate and discuss *examples* of how to accomplish tasks than it is to deal directly with task model abstractions. In a sense, we designed a kind of programming by demonstration [4, 12] system in which a domain expert performs a task by executing actions and then reviews and annotates a log of the actions.

In prior research, we developed machine learning techniques that infer hierarchical task models from a set of partially-annotated examples of task-solving behavior [5]. As a general tradeoff, an expert can provide minimal annotations about many examples or more exhaustive annotations about fewer examples. Also, as will be discussed below, certain types of annotations are more valuable to the learning engine.

We have integrated these machine learning techniques into a development environment that provides comprehensive support for experts to generate task models. This involves removing or refining past examples as well as defining new examples. In addition to learning from the collection of examples, the system can use them for regression testing to verify the behavior of the model throughout the development process. This technique detects more potential errors than simply checking the internal consistency of a model.

The design presented in this paper reflects the collective experience of the Collagen research group over the past several years "manually" developing task models. Typically, a model is constructed through an incremental development process, which is described in the following two paragraphs.

Initial versions of a task model are inferred from a small number of examples that show the most common solutions to key domain tasks. Both the model and the examples frequently undergo substantial revisions during this early stage. Next, the model will be generalized to cover additional examples that demonstrate solutions involving, for example, alternate orderings for actions, optional behavior, or alternate task decompositions. Occasionally, defining additional examples will spur the expert to re-conceptualize the entire domain, necessitating reworking many previous examples.

As the development process nears completion, there is less and less benefit to providing new examples. It is generally faster and easier to directly edit the model. Also, learned models, even when accurate, may need to be tweaked by the expert for other reasons. For example, as discussed in [5], the organization of a complete and accurate task model may be inappropriate for a collaborative agent. It is at this final stage of development that the ability to easily verify the behavior of the model using the collection of past examples is critical.

The next section of the paper describes how task models are inferred from the annotated examples of the expert; we also provide empirical results based on our implementation of this learning module. The third section of the paper presents the design of the model building environment in detail. The paper concludes with a discussion of related research.

MACHINE LEARNING FROM EXAMPLES

Within the task model development environment, there is a division of labor between the user and the computer: the user provides annotated examples so that the learning system can generalize the task model under development.

This section describes how a domain expert partially annotates examples. We describe the task model language first and then the different types of partial annotations. Empirical results are included that quantify how the different types of annotations influence the number of examples that need to be provided by the domain expert.

A task model is composed of actions and recipes. Actions are either primitive actions, which can be executed directly, or non-primitive actions (also called "intermediate goals" or "abstract actions"), which are achieved indirectly by achieving other actions. Each action has a type; each action type is associated with a set of parameters. Actions do not currently include an explicit representation for preconditions and effects.

Recipes are methods for decomposing non-primitive actions. Each recipe specifies a set of steps that are performed to achieve the non-primitive action that is the collective objective of the steps. All steps are assumed to be required unless they are labelled as optional. There may be several different recipes for achieving a single non-primitive action.

A recipe also contains constraints that impose partial temporal orderings on its steps, as well as various logical relations

among their parameters. For the purposes of this paper, the only logical relations we will consider are equalities. Equalities between a parameter of a step and a parameter of the objective of the recipe are called bindings, but are otherwise indistinguishable from constraints. Parameters and steps have a name as well as a type in order to allow for unambiguous references (in bindings and constraints) to multiple steps of the same type.

Figure 1 contains samples of this representation for a cooking domain that will be used throughout this paper as a running example. This domain was chosen over alternate Collagen task models because it is intuitive and can be easily varied in order to conduct empirical studies. A task model in the form of Figure 1 is the desired output of learning.

```
nonprimitive act PreparePasta
    parameter Pasta pasta

primitive act GetPasta
    parameter Pasta pasta

recipe PastaRecipe achieves PreparePasta
    steps       Boil boil
                CookPasta cook
                optional GetPasta get
    bindings    achieves.pasta = cook.pasta
    constraints get.pasta = cook.pasta
                boil.water = cook.water
                boil precedes cook
                get precedes cook
```

Figure 1: Collagen representations from a cooking domain (keywords are in bold).

Annotation Language

Informally, the input to the learning algorithm is a series of demonstrations; each one explicitly shows one correct way to perform a task and, via annotations, indicates other similar ways that are also correct. For example, if the sequence $[a, b, c]$ is correct and b is annotated as optional, then we know $[a, c]$ is also a correct example. We can also generalize from the annotated examples based on assumptions about the target model to be learned. For example, if the learner is told $[a, b, c]$ and $[c, b, a]$ are both correct and the target model represents partial ordering constraints on pairs of actions, all orderings of a, b, and c must be correct.

More precisely, each input to the learning engine is an annotated example e, where e is a five-tuple: $\langle \hat{e}, \mathcal{S}, optional, unordered, unequal \rangle$:

$\hat{e}$ is the temporally ordered list of actions $[p_1, \ldots, p_k]$ that constitute the *un*annotated example demonstrated by the expert. In most cases, each p_i will be a primitive action; however, p_i could also be an intermediate goal. The semantics of the latter case is that p_i is being used as a placeholder in lieu of fleshing out the example to include a segment that achieves p_i.

45

S is a segment, which is a pair $\langle segmentType, [s_1 \ldots, s_n] \rangle$. Each s_i, called a *segment element* or element for short, is either an action or a segment. Grouping elements together means that they collectively achieve a non-primitive act of type *segmentType*.

optional is a partial mapping from elements to boolean values. If the mapping is defined and is true, the expert is specifying that removing that segment element from the example would constitute another correct example from the domain.

unordered is a partial mapping from pairs of elements in the same segment to boolean values. If the mapping is defined and is true, the expert is specifying that switching the order of appearance of the pair of elements would constitute another correct example from the domain.

unequal is a partial mapping from pairs of action parameters to boolean values. If the mapping is defined and is true, the expert is specifying that another correct example with the same segmentation exists wherein these two parameters do not have the same value. This mapping does not convey information about inequality relations; i.e. this mapping cannot indicate that two parameters must never have the same value.

Figure 2 contains an example of how this formal notation is used to define an annotated example in the cooking domain. Each action is subscripted so that different instances of the same act type can be distinguished. The arguments of each primitive action are specific domain items.

$\hat{e} = [$ $\texttt{Boil}_1(\texttt{water}_9)$, $\texttt{GetPasta}_2(\texttt{spaghetti}_4)$,
$\texttt{CookPasta}_3(\texttt{spaghetti}_4, \texttt{water}_9)$, $\texttt{Boil}_4(\texttt{water}_9)$,
$\texttt{MakeSauce}_5(\texttt{marinara}_7)$, $\texttt{ServeDinner}_6(\texttt{kitchen}_5)$ $]$

$S = \langle$ $\texttt{MakeMeal}_0$, $[$
    $\langle$ $\texttt{PreparePasta}_7$, $[\texttt{Boil}_1, \texttt{GetPasta}_2, \texttt{CookPasta}_3]$ $\rangle$
    $\langle$ $\texttt{PrepareSauce}_8$, $[\texttt{Boil}_4, \texttt{MakeSauce}_5]$ $\rangle$
    $\texttt{ServeDinner}_6]$ $\rangle$

optional$(\texttt{GetPasta}_2) = true$
optional$(\texttt{ServeDinner}_6) = false$
unordered$(\texttt{PreparePasta}_7, \texttt{PrepareSauce}_8) = true$
unordered$(\texttt{Boil}_1, \texttt{CookPasta}_3) = false$
unequal$(\texttt{Boil}_1.\texttt{water}, \texttt{Boil}_4.\texttt{water}) = true$
unequal$(\texttt{GetPasta}_2.\texttt{pasta}, \texttt{CookPasta}_3.\texttt{pasta}) = false$

Figure 2: Sample annotated example, in formal notation.

Figure 2 defines one top-level segment of type $\texttt{MakeMeal}$, which is composed of two sub-segments ($\texttt{PreparePasta}$, $\texttt{PrepareSauce}$) and a primitive action ($\texttt{ServeDinner}$). The partial mappings at the bottom of the figure indicate that $\texttt{GetPasta}$ is optional and $\texttt{ServeDinner}$ is required. Also, the steps of type $\texttt{PreparePasta}$ and $\texttt{PrepareSauce}$ may appear in any order in general, while the $\texttt{Boil}$ step of $\texttt{PreparePasta}$ must always precede the $\texttt{CookPasta}$ step. Finally, the annotations indicate that the com-

mon water parameter value for $\texttt{Boil}_1$ and $\texttt{Boil}_4$ is a coincidence, but that the pasta parameter of $\texttt{GetPasta}$ and $\texttt{CookPasta}$ will always be the same.

A graphical interface, which is part of the development environment, allows experts to annotate this information in a more intuitive way, such as by marking certain nodes in a tree visualization. However, it should be clear from this partially annotated example that fully annotating examples would be quite burdensome regardless of the interface.

It is tempting to draw conclusions from the absence of certain annotations. For example, in Figure 2, many steps are not marked as optional. While one might interpret this to mean that the unmarked steps are required, it might just be the case that the expert is not sure if those steps are required or optional. To handle such cases, the learning techniques distinguish between positive evidence (e.g., annotating that a step is required) and the lack of negative evidence (e.g., the step appears in all defined examples involving this recipe).

A key feature of our learning algorithm is that it infers bindings, constraints, and parameters of non-primitive actions, which we will refer to collectively as *propagators*. The role of propagators is to enforce equality relationships among the parameter values of primitive actions. For example, in a task model for cooking spaghetti marinara, the cooked pasta must be the same pasta to which the marinara sauce is later added. In contrast, different knives can be used to cut, say, the tomatoes and the mushrooms. These equality relations cross the boundaries of many actions and recipes, i.e. they are not local to any particular recipe.

Empirical Results

Some experiments were run to better understand the tradeoff between how much information the expert provides in each example and how many examples must be provided to learn an accurate model of the domain. For testing purposes only, we simulate a human expert that provides varying types of annotations. This approach focuses the results on this tradeoff rather than the best way to elicit annotations from the expert. At present, we do not presume that there is a data base of unannotated examples that either the expert or the learner can access — examples are generated by the expert as needed.

In each experiment, we start with a target task model and use it to simulate the activities of a domain expert, both to generate unannotated examples and to annotate them. Segmentations and non-primitive action names are always provided by the simulated expert, but we varied which other annotations were provided. After each example is input to the learning engine, we determine if the generalized task model is equivalent to the target model. Also, we determine if each example was "useful," i.e. if it contained any information that altered the contents of the data structures used for inference; other examples are labeled "useless."

We ran experiments on two target task models. The first represents part of a sophisticated tool for building graphical user interfaces, called the Symbol Editor. The model was constructed in the process of developing an agent to assist novice users of the Symbol Editor. The model contains 29 recipes, 67 recipe steps, 36 primitive acts, and 29 non-primitive acts. A typical example contains over 100 primitive actions. The second test model was an artificial cooking world model designed to test the learning algorithm. The model contains 8 recipes, 19 recipe steps, 13 primitive acts, and 4 non-primitive acts. An example typically contains about 10 primitive actions. Both models have recursive recipes.

We ran all variations of possible combinations of annotation types, and report a subset in Table 1. In this table, O indicates that all ordering annotations are given, E indicates that all equality annotations are given, and P indicates that all propagators are given (propagator annotations subsume equality annotations). Optional steps are only annotated when all annotations are given (indicated by 'All' in the table). The reason for this is that optionality is the easiest aspect to learn because it does not involve relationships between steps. The data are the results of randomized sequences of examples — 100 trials for the cooking domain and 20 trials for the Symbol Editor. Also, the average and minimum are measured on useful examples.

Table 1: The kind of annotations provided influences the number of examples needed to learn task models.

Anno-tation	Cooking			Symbol Editor		
	Avg.	Min.	Useless	Avg.	Min.	Useless
All	5.3	3	9.9	1.9	1	0.1
OP	6.5	3	11.1	2.4	1	0.4
P	7.2	4	14.1	3.0	2	0.5
EO	7.2	3	10.4	14.2	3	47.0
E	8.1	4	13.1	14.4	3	46.9
O	38.3	15	404.3	53.0	37	118.7
None	38.3	15	404.2	53.1	37	118.6

The main surprise is that providing equality annotations dramatically reduces the number of required examples (from 38.3 to 8.1 for cooking). This is encouraging because it seems likely that it will be much less onerous for a human expert to indicate when apparent equalities in the example are coincidental, than to construct all the propagator information directly.

Another interesting result in Table 1 is that learning is strongly influenced by the order in which examples are processed. This is reflected both by the minimum number (which is roughly half the average number) of useful examples and the average number of useless examples (which is comparatively large). It is possible a human would provide diverse, useful examples so that the number of examples required in practice would be close to the minimum number of useful examples.

DESIGN FOR A MODEL DEVELOPMENT ENVIRONMENT

This section presents the design for our task model development environment. A novel aspect of the environment is support for a domain expert to refine past examples as he or she develops a clearer understanding of how to model the domain. Collectively, these examples constitute a "test suite" that the development environment manages in order to verify that changes to the evolving task model do not have unintended consequences.

Figure 3 shows an idealized sequence that a user would follow to develop a task model. In practice, a model would not be developed in such a straightforward path. For example, the model can be inferred, visualized, or manually edited by the user at arbitrary points during the development cycle.

- Define a starting set of actions (optional).
- Define examples.
- Generalize the model based on the examples.
- Visualize the resulting model.
- Use the model in subjective tests of quality.
- Refine prior examples.
- Manually edit the task model.
- Run regression tests with the collection of examples.

Figure 3: Typical steps in task model development

Most of the process described in Figure 3 is presently achievable through a single GUI application; however, certain aspects currently require a combination of command-line programs and text-file editing. Fully integrating the current capabilities into a single tool will reduce the burden for the domain expert, and will provide opportunities for providing more assistance.

Both Collagen and the model building tool are written in the Java programming language. This has two important beneficial consequences. First, it is easy to design the system to switch between alternate components (e.g., model viewers or learning engines) by using interfaces and Java Beans. Second, the task model language for Collagen is implemented as a superset of Java, which permits very specific refinements of task models for any particular domain.

The rest of this section is a "story board" that illustrates how a person might use our system to develop a task model for making a meal. As this task is something that occurs in the physical world, the user constructs examples by virtually walking through the process of making a meal — it is a mental walk-through, rather than an actual walk-through. In contrast, for a computer application with a graphical user interface that has already been built and implemented, a person could simply run the application, and annotate the resulting log. For an application that is being (re-)designed, an expert would use the virtual walk-through process to create examples.

Define a Starting Set of Actions (Optional) The first step the user may take is to generate an initial list of primitive and non-primitive acts, as shown in Figure 4.

Non-Primitives: MakeMeal

Primitives: Boil, CookPasta, PrepareSauce, ServeDinner

Figure 4: Initial working set of action types

This categorization may change over time, but helps to bootstrap the process. Our system does not assume the existence of a pre-defined hierarchy of actions (i.e., an ontology) for the domain — determining this hierarchy is a major part of task model development. While defining the primitives for an implemented GUI application may be straightforward, for activities in the real world the process is more difficult. In all non-trivial domains, identifying the correct set of abstract (non-primitive) actions is challenging.

Define Examples An example is an annotated list of instantiated actions (action type plus specific values for parameters) that constitute the achievement of a goal in the domain. The user may start by constructing an unannotated demonstration of how to make a meal, as shown in Figure 5 (for reading ease, actions in the examples of this section are not subscripted).

```
GetPasta
Boil
CookPasta
PrepareSauce
ServeDinner
```

Figure 5: First example, unannotated

In Figure 5, the user has not specified any parameter values. Also, the example includes an unknown action type (GetPasta), so the system may either ask the user if the action should be added to the working set as a primitive act or silently do so, depending on a settable option.

Working with this example, the user groups related actions into segments; for each segment, the user provides a name that describe the purpose of the segment. In the minimally annotated version shown in Figure 6, the elements of a segment are identified visually by the level of indentation — the purpose name for a segment, which precedes its elements, is surrounded by brackets. In this case, the user has grouped the first three steps into PreparePasta. The system recognizes that this act is not part of the working set and can ask if it should be added to the non-primitives. In practice, annotations don't have to be added during a second pass — they can be done at the same time that actions are added.

Generalize the Model Based on the Examples After annotating one or more examples, the user can invoke the infer-

---

```
[MakeMeal]
    [PreparePasta]
        GetPasta
        Boil
        CookPasta
    PrepareSauce
    ServeDinner
```

Figure 6: First example, minimally annotated

---

ence engine to generalize the task model to incorporate the examples. Even if the learning tool has only one example to process (i.e. this example is the first one submitted by the user), generalization may occur if the same non-primitive action-type appears more than once in the example.

The result of learning from the example in Figure 6 is given in Figure 7. A comparison with Figure 1 shows that the propagators are missing from this first version of the task model. Barring guidance from the domain expert, the automated techniques name each recipe based on the types of the required steps, sorted alphabetically, of the recipe. Also, step names are derived from the type of the step.

---

nonprimitive act PreparePasta

primitive act GetPasta

recipe Boil_CookPasta_GetPasta **achieves** PreparePast
   steps   Boil boil
                     CookPasta cookPasta
                     GetPasta getPasta
   constraints   getPasta **precedes** boil
                     boil **precedes** cookPasta

Figure 7: A portion of the task model after one example

An important issue is how the machine learning techniques will preserve manual edits to the task model. This issue arises in many environments where a human and a computer are collaborating — a person will often want to pin down some parts of the problem at hand so that the computer does not modify them when formulation a solution. The learning techniques currently allow an expert to pin down some parts of the system, but there are still some open issues regarding parameters for non-primitives.

As discussed in the previous section, the learning techniques free the domain expert from having to specify non-primitive parameters. Or, for those who are so inclined, an expert can specify the number and type of parameters for each non-primitive. However, mixing the two approaches requires either heuristics or dialogs with the expert or a combination of both. To see why, imagine an expert state that a non-primitive has a parameter of type X and that the learning engine infers the need for two slots of type X. Do either of these get mapped to the user-specified parameter and, if so, which one? There

are other ambiguous situations where the decision whether or not to re-use a propagator is unclear.

Define Additional Examples The expert iterates through this process until the task model is complete enough to test subjectively. After the user defines each additional example, inconsistencies between the current model and the new example need to be worked out. There are several ways in which the user and the system could interact to resolve inconsistencies; currently it is the sole responsibility of the domain expert.

As a result of resolving inconsistencies, sometimes the model will be changed and sometimes the example will be changed. For example, an action that was originally defined as a primitive action might appear as a segment purpose type. Sometimes the definition of the action needs to be changed and sometimes the expert makes an error. Another common source of inconsistencies is in the number and type of parameters for an action.

Figure 8 is an example of making linguini with clam sauce that might be provided as a second example to the learning system. In this figure, the user has added more detail by decomposing PrepareSauce and specifying parameters.

[*MakeMeal*]
    GoToKitchen(kitchen$_2$)
    [*PrepareSauce*]
        Boil(water$_3$)
        CookClams(clams$_8$,water$_3$)
        MakeClamSauce(clams$_8$)
    [*PreparePasta*]
        Boil(water$_3$)
        CookPasta(linguini$_{11}$,water$_3$)
    ServeDinner(kitchen$_2$)

Figure 8: Second example, minimally annotated

When processing this second example, the system adds new acts to the working set of primitives, moves PrepareSauce from the primitives to the non-primitives, and adds parameters to the definitions of actions that appear in this example. The system also determines that there are optional steps in two different recipes (going to the kitchen and getting the pasta) and that the PreparePasta and PrepareSauce steps are unordered. Finally, in this example, the water used in preparing the sauce and the pasta happen to be the same so the system infers a set propagators that will force this equality to always hold. Figure 9 shows the output of learning.

In future examples, new acts may be demonstrated. Or, additional recipes to achieve known acts may be shown (e.g., achieving MakeMeal by calling a take-out restaurant). Also, additional orderings will be learned for known recipes. Other examples will show that parameters are not tied to specific domain literals; likewise, the water used in preparing a sauce and the water used in preparing pasta are generally differ-

---

nonprimitive **act** PreparePasta
    parameter Water water

primitive **act** GetPasta

recipe Boil_CookPasta **achieves** PreparePasta
    steps   Boil boil
                   CookPasta cookPasta
                   optional GetPasta getPasta
    bindings   achieves.water = cookPasta.water
    constraints   boil.water = cookPasta.water
                   cookPasta.water = water$_3$
                   cookPasta.pasta = linguini$_{11}$
                   getPasta **precedes** boil
                   boil **precedes** cookPasta

Figure 9: A portion of the task model after two examples

---

ent. Future work includes investigating mechanisms for the learning system to indicate what types of examples would most benefit the learning process.

Visualize the Model After the inference techniques have generalized the model to account for this example, the expert can review the result to see if it matches his or her intention. In future work, additional bookkeeping by the learning techniques will enable the expert to find out what part(s) of which example(s) implied various pieces of the model (e.g. step optionality, ordering and equality constraints). Some pieces of the model will not be tied to examples — for example, if the model is manually edited.
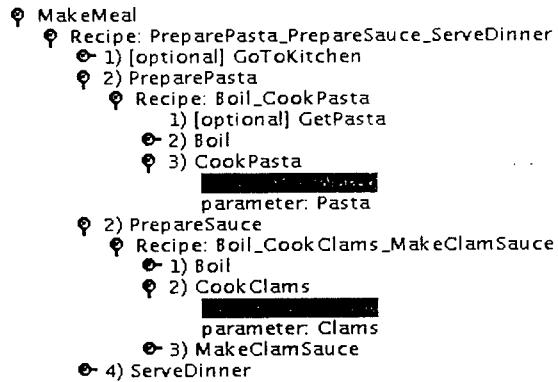
Figure 10: A graphical view of part of a task model.

Another feature of our system is that it supports multiple views of the task model. In Figure 1 (and throughout this section) we saw the textual representation of a portion of a task model. In Figure 10 we see a graphical view of the decompositions for a task in the task model, drawn as a tree.[1] The children of non-primitive actions are the recipes that achieve that act; a recipe expands to show the steps of that recipe; and

---

[1] To simplify exposition, some visual elements of the tool have been been suppressed or replaced by text (e.g. "[optional]").

the children of primitive acts are the action type's parameters. The numbering of recipe steps summarizes the precedence relations. Also, parameters that are constrained by the model to always be equal, i.e. the two Water parameters shown, are indicated by having the same background color (different colors are used for each set of propagators).

Use the Model in Subjective Tests of Quality   Ideally, the current task model can be evaluated by the domain expert by interacting with a collaborative agent for an existing GUI application.[2] Based on this interaction, the expert can identify weaknesses or errors in the model. For example, the expert can notice when the agent erroneously propagates a parameter value (as would be the case for the water used to boil clams and pasta). To improve the quality or accuracy of the model, the expert can create new examples, refine old examples, or manually edit the model .

Refine Prior Examples   After using the task model (or at other times), the domain expert may wish to refine prior examples. For example, showing the learning system that Get-Pasta has a pasta parameter with the same value as Cook-Pasta is easily done by adding parameter values to the first example. Figure 11 shows such a refinement of the first example.

[$MakeMeal$]
    [$PreparePasta$]
        GetPasta(ziti$_4$)
        Boil(water$_{12}$)
        CookPasta(ziti$_4$,water$_{12}$)
    [$PrepareSauce$]
    ServeDinner(kitchen$_1$)

Figure 11: First example, refined with parameters

In Figure 11, PrepareSauce is now marked as a place-holder non-primitive, i.e. a non-primitive that is not decomposed in this example. Alternately, the domain expert could have refined PrepareSauce in Figure 11.

Manually Edit the Task Model   The domain expert may wish to manually edit the task model, perhaps as a result of a subjective evaluation. For example, using the task model inferred from the two defined examples reveals that some dialogs flow unnaturally because PreparePasta does not have a parameter of type Pasta, even though that parameter is not needed for correctness. In addition, some experts may choose to replace the automatically generated recipe and step names, or may choose to take advantage of Collagen's flexible glossing (English text generation) mechanism.

Since the task model language for Collagen is a superset of Java, actions and recipes can include arbitrary Java code. In terms of system design, this means that there must be support

---

[2]Evaluation is always possible — Collagen can simulate the behavior of a collaborative agent in the absence of an existing application.

for experts to manually add Java code to the task model when needed.

Regression Testing   As task models become complex, it is easy to make changes that have unexpected consequences. Our current implementation incorporates a rigorous testing facility to identify when changes to the task model influences the analysis of stored examples. Regression testing is not as useful in the early stages of model development, when action and recipe definitions are undergoing rapid change.

As the model is developed and refined, older examples may become obsolete. Changes in the number and types of parameters for an action, for example, may cause an early example to no longer be consistent with the current version of the task model. Likewise, as actions are added and deleted, and potentially reclassified as primitives or non-primitives, earlier examples may no longer be valid. An important consideration for the development environment is how to alert the user that these examples exist and how to provide an explanation of why the example is no longer usable. In some cases, the user may choose to disregard an early example while at other times he or she may choose to update an earlier example so that it can continue to be used in regression testing.

RELATED RESEARCH

Tecuci *et al.* [19] present techniques for acquiring large numbers of hierarchical if-then task reduction rules through demonstration by and discussion with a human expert. In their system, the expert provides a problem-solving episode from which the system infers an initial task reduction rule, which is then refined through an iterative process in which the human expert critiques attempts by the system to solve problems using this rule. Tecuci *et al.* do not specifically address either the issue of building a model in the absence of a pre-defined ontology or the notion of regression testing to ensure that model updates preserve correctness.

Gil and Melz [6] and Kim and Gil [8] have reported on a wide range of issues related to building knowledge acquisition tools for developing databases of problem-solving knowledge. In contrast to our approach of inferring task models from annotated examples, they have focused on developing tools and scripts to assist people in editing and elaborating task models, including techniques for detecting redundancies and inconsistencies in the knowledge base, as well as making suggestions to users about what knowledge to add next.

Paternò [15, 16] presents a graphical tool for eliciting hierarchical task models, represented as ConcurTaskTrees, of cooperative activities. This tool provides support for converting informal scenarios into formal descriptions and then verifying the consistency of a ConcurTaskTree with saved scenarios. In addition to some key differences in representation language (such as alternate decompositions for abstract actions and the ability to enforce arbitrary constraints among parameters), this tool does not support inference.

Other research efforts have addressed aspects of the task model learning problem not addressed in this paper. Bauer [2, 3] presents techniques for acquiring non-hierarchical task models from unannotated examples for the purpose of plan recognition (i.e., inferring a person's intentions from her actions). OBSERVER [21] automatically learns the preconditions and effects of planning operators from unannotated expert solution traces and then refines the operators through practice. In a related approach, van Lent and Laird [20] present techniques to learn the preconditions and goal conditions for a hierarchy of operators (encoded as specialized Soar production rules) given expert-annotated performance traces.

Angros Jr. [1] presents techniques that learn recipes that contain causal links, to be used for the intelligent tutoring systems, through both demonstration and automated experimentation in a simulated environment. Masui and Nakayama [14] investigate learning macros from observation of or interaction with a computer user in order to assist the user with tasks that occur frequently or are inherently repetitive. Lau *et al.* [9], in one of the few formal approaches to learning macros, uses a version space algebra to learn repetitive tasks in a text-editing domain.

CONCLUSION

This paper presented an approach, which is implemented in a development environment, for constructing and maintaining a hierarchical task model from a set of annotated examples provided by a domain expert. The key pieces of the system are a machine learning inference engine and a facility for conducting regression testing in order to verify the consistency of a task model with previously defined examples. As a general tradeoff, the domain expert can either provide minimal annotations about many examples or more exhaustive annotations about fewer examples.

Future work for this project fall into two broad categories: extensions to the inference engine and improvements in the usability of the development environment. One area for future work that falls into both categories is to develop *constructive critics*, i.e. algorithms that propose possible annotations, and include a facility for users to easily manage the advice provided by them. For example, one constructive critic might analyze past usage logs (or annotated examples) to suggest the segment elements that should be marked optional in an as-yet unannotated example. The user should be able to: 1) review such suggestions at any time, 2) easily understand them, and 3) easily accept none, some, or all of the them.

REFERENCES

1. R. Angros Jr. *Learning What to Instruct: Acquiring Knowledge from Demonstrations and and Focussed Experimentation.* PhD thesis, University of Southern California, 2000.

2. M. Bauer. Acquisition of Abstract Plan Descriptions for Plan Recognition. In *Proc. 15th Nat. Conf. AI*, pages 936–941, 1998.

3. M. Bauer. From Interaction Data to Plan Libraries: A Clus-

tering Approach. In *Proc. 16th Int. Joint Conf. on AI*, pages 962–967, 1999.

4. A. Cypher, editor. *Watch What I Do: Programming by Demonstration.* MIT Press, Cambridge, MA, 1994.

5. A. Garland, N. Lesh, and C. Sidner. Learning Task Models for Collaborative Discourse. In *Proc. of Workshop on Adaptation in Dialogue Systems, NAACL '01*, pages 25–32, 2001.

6. Y. Gil and E. Melz. Explicit representations of problem-solving strategies to support knowledge acquisition. In *Proc. 13th Nat. Conf. AI*, pages 469–476, 1996.

7. B. Grosz and C. Sidner. Plans for discourse. In P. R. Cohen, J. Morgan, and M. E. Pollack, editors, *Intentions in Communication*, pages 417–444. MIT Press, Cambridge, MA, 1990.

8. J. Kim and Y. Gil. Acquiring problem-solving knowledge from end users: Putting interdependency models to the test. In *Proc. 17th Nat. Conf. AI*, pages 223–229, 2000.

9. T. Lau, P. Domingos, and D. S. Weld. Version space algebra and its application to programming by demonstration. In *Proc. 17th Int. Conf. on Machine Learning*, pages 527–534, 2000.

10. N. Lesh, C. Rich, and C. Sidner. Using Plan Recognition in Human-Computer Collaboration. In *Proc. of the 7th Int. Conf. on User Modeling*, pages 23–32, 1999.

11. N. Lesh, C. Rich, and C. Sidner. Collaborating with Focused and Unfocused Users under Imperfect Communication. In *Proc. 9th Int. Conf. on User Modeling*, pages 64–73, 2001.

12. H. Lieberman, editor. *Your Wish is My Command: Programming by Example.* Morgan Kaufmann, 2001.

13. K. E. Lochbaum. A Collaborative Planning Model of Intentional Structure. *Computational Linguistics*, 24(4):525–572, Dec. 1998.

14. T. Masui and K. Nakayama. Repeat and predict– two keys to efficient text editing. In *Conference on Human Factors in Computing Systems*, pages 118–123, 1994.

15. F. Paternò and C. Mancini. Developing task models from informal scenarios. In *Proc. ACM SIGCHI '99, Late-Breaking Results*, pages 228–229, 1999.

16. F. Paternò, G. Mori, and R. Galiberti. CTTE: An environment for analysis and development of task models of cooperative applications. In *Proc. ACM SIGCHI '01, Extended Abstracts*, pages 21–22, 2001.

17. C. Rich and C. Sidner. COLLAGEN: A Collaboration manager for Software Interface Agents. *User Modeling and User-Adapted Interaction*, 8(3/4):315–350, 1998.

18. C. Rich, C. Sidner, and N. Lesh. Collagen: Applying Collaborative Discourse Theory to Human-Computer Interaction. *AI magazine*, 22(4), 2001. To appear. http://www.merl.com/papers/TR2000-38/.

19. G. Tecuci, M. Boicu, K. Wright, S. W. Lee, D. Marcu, and M. Bowman. An integrated shell and methodology for rapid development of knowledge-based agents. In *Proc. 16th Nat. Conf. AI*, pages 250–257, 1999.

20. M. van Lent and J. Laird. Learning hierarchical performance knowledge by observation. In *Proc. 16th Int. Conf. on Machine Learning*, pages 229–238, 1999.

21. X. Wang. Learning by observation and practice: an incremental approach for planning operator acquisition. In *Proc. 12th Int. Conf. on Machine Learning*, pages 549–557, 1995.

Building and Exploiting Ontologies for an Automobile Project Memory

Joanna Golebiowska [1][2], Rose Dieng-Kuntz [1], Olivier Corby [1], Didier Mousseau [2]

1 INRIA, ACACIA Project, 2004 route des Lucioles, BP 93, 06902 Sophia-Antipolis Cedex, France

2 RENAULT, TPZ D12 138, DTSI/DTPU/KMPD, sce 18820 860 quai de Stalingrad, 92109 Boulogne, France

E-mail: {Joanna.Golebiowska, Rose.Dieng, Olivier.Corby}@sophia.inria.fr

Abstract

This paper describes SAMOVAR (Systems Analysis of Modelling and Validation of Renault Automobiles), aiming at preserving and exploiting the memory of past projects in automobile design (in particular the memory of the problems encountered during a project) so as to exploit them in new projects. SAMOVAR relies on (1) the building of ontologies (in particular, thanks to the use of a linguistic tool on a textual corpus in order to enrich a core ontology in a semi-automatic way), (2) the «semantic» annotations of the descriptions of problems relatively to these ontologies, (3) the formalisation of the ontologies and annotations in RDF(S) so as to integrate in SAMOVAR the tool CORESE that enables an ontology-guided search in the base of the problem descriptions.

1 Introduction

How to preserve and exploit the memory of past projects in automobile design (in particular the memory of the problems encountered during a project) so as to exploit them in new projects? The role of ontologies for knowledge management is more and more. They can play an important role for building a project memory, that is a specific kind of corporate memory [9,10]. Several researchers aim at proposing a methodology for building such ontologies, possibly from textual information sources [2]. Such a methodological framework is interesting for us, as there are several heterogeneous sources of information inside the company: different databases, official references, problem management systems and other specific bases in the departments; moreover, in addition to basic data which can be processed by traditional means, some bases contain important textual data.

After detailing our problematic and the concrete problem to be solved at Renault, we will present the approach adopted for SAMOVAR. Then we will detail our techniques for building the SAMOVAR ontologies, relying on both manual construction and semi-automatic construction thanks to the application of heuristic rules on the output of a linguistic tool applied on a textual corpus stemming from textual comments of a database. Then we will explain their exploitation and the use of the CORESE (Conceptual Resource Search Engine) tool [8] for

information retrieval about the descriptions of past problems encountered in vehicle projects. We will generalize our approach so as to propose a method for building a project memory in the framework of any complex system design. In our conclusion, we will compare SAMOVAR to related work.

2 The problematic

The field of SAMOVAR is the process of prototype validation during a vehicle project. This process is intrinsically complex and raises many problems. These problems frequently slow down the cycle due to the necessity of repeating validations: so, it increases both the delays and the costs of such projects.

A close observation of validation shows that part of the failure is due to loss of information and of experience gained. The objective of SAMOVAR is to improve the exploitation of this information and make it available for future projects. Useful data exist in the form of text. Therefore it is necessary to find suitable techniques and tools, such as for example linguistic techniques for exploiting the knowledge underlying such texts.

2.1 Context

The product development cycle of an automobile is made of numerous repetitive sub-cycles (design/ development / validation) - of short or long duration. The whole cycle is punctuated by milestones and prototype waves which mark the production of successive models and prototypes, more or less complex. During a vehicle project, validations are carried out: the testing department checks that the component-parts or the functions satisfy the requirements of the product specifications.

Thus, the quality of smoothness of the dashboard, the noise of a car door being shut, the behaviour of the car on cobble stones, or even its resistance to high or low temperatures are tested. These validations are spread throughout the vehicle project and done successively by the testing department, starting from the most elementary functions till the final synthesis test. The project begins with tests related to the engineering center according to the parts validated and ends with tests on performance, speed and crash.

These project validation phases often reveal discrepancies with respect to the specifications. From detection of a problem to its resolution, such problems are documented in a unique data management system called Problem Management System (PMS). This system uses a database including the information needed for the process of problem management: especially information on the actors involved in the project and above all, the descriptions and comments on the problems that arose.

2.2 Interest of exploiting the Problem Management System

The appearance of problems increases the additional costs and the project duration. Therefore solutions have been thought out. One possible solution would be to exploit the information contained in the PMS in order to use the PMS not only as a problem management system but also as a source of information.

The PMS can be considered as a huge source of information, thanks to the textual fields of the base which are particularly rich and under-exploited. The actors involved in the automobile design project express themselves freely for describing the problems detected, as well as the various solutions proposed, or the constraints for carrying out such or such solution. This base can therefore be considered as archives or even as constituting (a part of) the memory of a project, more precisely the memory of the problems encountered during the project.

Furthermore, in the company, there are other information sources, such as the official corporate referential or the numerous local bases of the testing department. It would be useful to exploit this information with the contents of the PMS.

Therefore our aim is to propose a means of retrieving, structuring and making reusable this wide quantity of information for the same project or for the other projects. The participants of current projects have expressed needs related to information search and retrieval useful during the validation phases. Their needs concerned especially the retrieval of similar incidents, detection of any correlation or dependency with other incidents and so the reuse of existing solutions within the same or even a different project.

Some pieces of information are relatively simple to retrieve. However, this is not the case for the textual data of PMS. The vocabulary used by the project participants in such comments is broad and varied: a given term (existing in the corporate official referential) frequently has different designations according to the department or even the phase reached in the project. Therefore, our objective was to detect a suitable semantic term, to classify it according to the validation process and to link it with all the variations encountered. So, we needed to extract the main terms of the domain (and the relations between them if possible) and to structure them in our ontology.

2.3 SAMOVAR's approach

A synthesis of tools dedicated to the extraction of terms and of relations from textual corpora is proposed in [3]. Several linguistic tools exist to extract candidate terms: Lexter [5], Nomino[1], Ana [11] [12]. With regard to the acquisition of semantic relations, several approaches enable to acquire them (based on the exploitation of syntactical contexts : [17], or the use of the lexical-syntactical patterns : [18], [19]). Few tools are offered such as Coatis [14] for causal relationships, Cameleon [28] [27] for hyponymy and meronymy relations.

The approach of SAMOVAR consists of structuring the knowledge contained in the PMS textual fields describing problems, and of enabling the user to carry out searches with the aim of finding similar problem-descriptions.

---

[1] http://www.ling.uqam.ca/nomino

As a starting point, we took directly the exploitable sources (i.e. the different databases of the company), and then we built up several ontologies offering different viewpoints on the validation process: problems, projects, services, components (i.e. parts). After having primed our base manually, we completed it progressively, with the elements from the PMS textual data using Natural Language Processing (NLP) tools – in particular, Nomino that was chosen as term extractor for availability reasons. This stage is automatic, however the support of an expert is necessary throughout the process. Then we annotated the problem descriptions automatically with instances of concepts of the ontologies. Finally we facilitated the access to the base of problem-descriptions thanks to the formalization in RDF(S) of the ontologies and of the annotations, enabling the use of the CORESE tool [8] to carry out ontology-guided searches through the such annotated base of problem-descriptions. The whole SAMOVAR approach is summarized in figure 6.

3 SAMOVAR ontologies

The SAMOVAR base is a multicomponent ontology composed of 4 ontologies, each dedicated to the description of a precise field :

- *Component* Ontology: it is based on the official company referential, corresponding to the functional segmentation of a vehicle into sub-components;

- *Problem* Ontology: it contains the problem types and it is built up semi-automatically from a manually-activated core from textual fields taken from the problem management system;

- *Service* Ontology: it corresponds to the services cross-referenced with the company organization (management and profession) and it is supplemented by PMS information. This ontology gives an additional overall point of view on the problems;

- *Project* Ontology: it reflects the structure of a project and it is made up of knowledge acquired during a project vehicle, according to the interviews carried out with different actors on the project.

Each ontology is a n-leveled hierarchy of concepts linked by the specialization link.

All the ontologies (or Samovar ontology components), apart from the Problem ontology, were built automatically, by an extraction of the PMS data base.

Remark: Instead of building several interconnected ontologies, we could have built one single ontology organized through several sub-ontologies. We chose to distinguish the different ontologies in order to enable their possible reuse independently from one another. The various constituents of our ontology correspond to the possible points of view concerning the the validations process. Even though, in fact, they constitute a single object, it is important to protect the possibility of various points of approach for validations.

3.1 Construction of the ontologies

The ontologies were built through two phases according to the data type and the means involved:

- a first extraction of the information contained in data bases,
- a second extraction, with specific techniques and tools for discovering the information « hidden » in texts.

The core of our ontology was primed manually, thanks to elements stemming from existing bases (see figure 1).
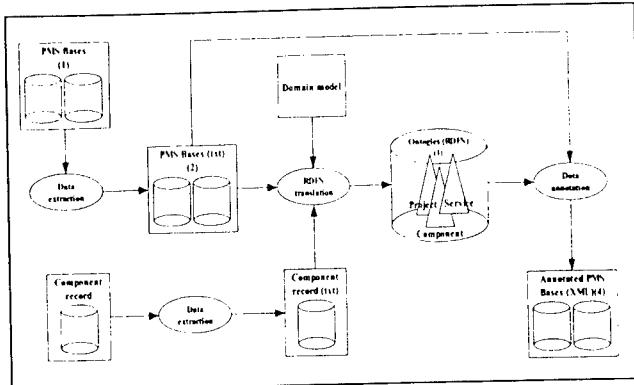
Figure 1: *Construction of ontologies for SAMOVAR – first data extraction*

A first extraction of the initial data (1) supplied a textual format (2) which was then translated in the form of an ontology, by respecting the RDFS format (as expected by CORESE). In parallel, another extraction was made from the Component referential in order to complete the previous data with additional information. In this way Component, Service and Project Ontologies are obtained, our ontological base (3). Then this base was used to annotate the data with the terms designating concepts of the ontologies. Thus we obtained the initial base annotated with annotations related to the concepts of the ontologies (4).

A second process deals with the textual data (the final goal being to enrich the result of the first extraction with the information stemming from the texts). To be able to deal with a text we needed a minimun of tools adapted to this type of data – the Natural Language Processing tools. We wanted to avoid heavy treatments requiring building the entire chain of treatment, for this reason we've reduced NLP treatments to the candidates terms.

This process exploits the output obtained after application of the linguistic tool Nomino on the textual corpus stemming from the textual comments contained in the problem management system (PMS). Nomino is a tool for extraction of nominal groups from a representative corpus in a domain. Nomino takes as input a textual corpus and produces as output a set of « lexicons » - lists of nouns, nominal complex units (NCU), additional nominal complex units (ANCU), verbs, adjectives, adverbs. The (A)NCU corresponds to the prepositional groups (PG) or the nominal groups (NG). The lexicons of the NCU are accessible in the form of graphs which illustrate the existing dependencies for a PG or a NG.

Then, we exploited the lexicons and the graphs produced by Nomino, in order to :

- detect the significant terms (i.e. corresponding to important validation points in the automobile design validation process),
- enrich the *Problem* ontology by means of the Nomino graphs, by exploiting the regularity of their structures.

3.1.1 Detection of significant terms

Firstly, we analysed the lexicons produced by Nomino in order to discover the most frequent terms, likely to be the most representative terms of the domain : *wiring, assembling, pipe, attachment, centring, component, installation, conformity, branch, hole, clip, screw, contact, maintains, tightening, paw, position, geometry, connecting.*

These structured terms allowed us to set up the *Problem* ontology. The initial structuring of this ontology was based on discussions with the experts. Figure 2 shows an extract of this *Problem* ontology.
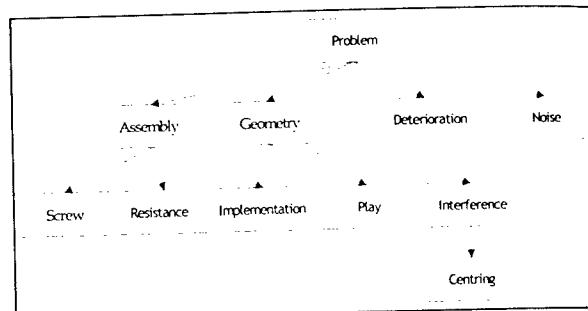
Figure 2: *Extract of the Problem Ontology*

The terms selected for the bootstrap were those which are exploitable as semantic clues for a problem type: for example, a problem of Centring can be discovered thanks to the presence of such clues as «indexage», coaxiality, «entraxe», etc.

Indeed the Nomino outputs can be sorted by frequence numbers. The most frequent words can be considered as relevant fr the processed domain and we exploit them as clues for the *Problem* ontology bootstrap.

The validity of the terms (i.e. the candidate terms for the bootstrap, and the clues exploited to find them) was confirmed with support of the experts.

Once the bootstrap of ontology was constituted, it needed to be enriched. For this purpose, we used the prepositional groups stemming from Nomino.

The extraction process implemented so far was applied to the enrichement of Problem Ontology. The other ontologies were constructed automatically from different data base fields, with help of interviews information. That is why most examples presented below concern only Problem Ontology. In the second phase we intend to reuse this method to enrich the Component ontology, notably to extract supplementary terminologie (synonyms, etc.)

3.1.2 Enrichment of the Problem ontology

Besides nouns, Nomino produces nominal and prepositional groups. We exploited the structures of the most frequent cases produced by Nomino.

The manual analysis of these NCU was performed by studying each Nomino output carefully so as to find some regularities in the NCU obtained by Nomino. This manual analysis, carried out with the support of the expert, supplied the structures which we exploited to build the SAMOVAR heuristic rules. For instance, we could find cases such as:

- (*DIFFICULTY EFFORT PROBLEM HARDNESS LACK RISK*) OF PROBLEM
- *DISCOMFORT* FOR PROBLEM OF PART
- *IMPOSSIBILITY* OF PROBLEM OF PART
- PROBLEM(*INCORRECT IMPOSSIBLE INSUFFICIENT DIFFICULT*)
- (*DAMAGE DISPLACEMENT LACK BREAK BREAKAGE*) OF PART

We exploited these structural regularities of Nomino outputs to build manually heuristics rules validated by the expert, heuristic rules which would enable the feeding of the ontology in a semi-automatic way.

These rules that reflected the existing structures in the corpus were determined manually, but once implemented and activated, they helped us to enrich the *Problem* ontology automatically by suggesting to attach a relevant new concept corresponding to a new term, at the right position in the ontology. Figure 4 shows examples of heuristic rules.

R1 : Noun [type=Problem,n=i] Prep[« of »]
Noun[type=Problem,n=i+1] ;

R2 :(difficulty||effort||hardness||lack||risk) Prep[« of »]
Noun[type=Problem]

R3 : impossibility Prep[« of »] **Noun[type=Problem]**
Prep[« of »] Noun[type=Component]

R4 : **Noun[type=Problem]** Prep[« of »||« on »||« under »]
Noun[type=Component]

Figure 3: *Examples of heuristic rules*

These rules represent the possible combinations between the elements of the *Component* and *Problem* ontologies as attested in the texts. A rule is presented as a series of categories, each one possibly decorated with a set of features (for example type=Problem to indicate that the element is part of the *Problem* ontology, type=Component for an element of *Component* ontology, etc.).

For example, the rule R1 authorizes a succession of terms consisted of noun, preposition and noun, where the first is a noun of Problem type, it is followed by a preposition "of" aanother noun of Problem type, which becames the son of the first noun.

The second rule R2, authorizes a succession of terms consisted of noun, preposition and noun, where the first can be "difficulty" ("effort", "hardness" or "lack"), followed by a preposition "of" and another noun of Problem type.

These rules were implemented in PERL.

3.1.3 Kinematic of the process

We enriched the *Problem* ontology gradually (see Figure 4). For that, the SAMOVAR system takes in entry the Nomino outputs, the *Component* ontology, *Problem* ontology bootstrap and the heuristic rule base. Then it analyses the nominal groups to see with which rule each of them can match.

Example of a Nominal Group and the corresponding rule:

NOISE OF RUBBING OF THE WHEEL DURING ITS HEIGHT ADJUSTMENT
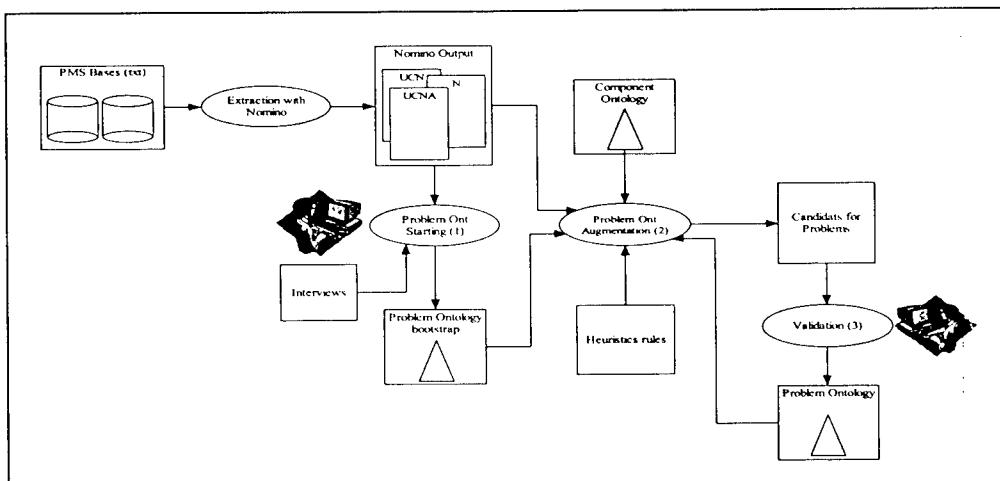
Noun[type=Problem,n=i] Prep[« of »] Nom[type=Problem,n=i+1]

The rule matches the nominal group, recognises the first term as a noise (that corresponds to an existing concept in the *Problem* ontology) and proposes to build a concept for the second noun and to insert it in the *Problem* ontology, as a son of the *Noise* concept. In the following case, the rule matches the name of the part and proposes to link the first term as a *Problem* :

JUDDERING OF THE REAR SWEEP ARM ON PPP3

Noun[type=Problem] Prep[« of »||« on »||« under »] Noun [type=Component]

The output provides the candidate terms to insert in the *Problem* ontology. The knowledge engineer (possibly with the support of the expert) validates each candidate and decides if the position proposed for insertion in the existing *Problem* hierarchy is correct. If yes, a concept corresponding to the term is inserted in the ontology. Such a concept – that was attested in the textual corpus - can be compared to a «terminological concept» if we use the terminology of Terminae [4].

Figure 3: *Process of enrichment of the ontology Problem*

To formalize our ontologies, we chose the RDF Schema (RDFS) language, which is recommended by W3C for description of resources accessible by the Web. RDFS allows to simply describe the ontology to which RDF annotations will be relative to. Such RDF annotations are quite relevant to describe resources within a company. We can consider the descriptions of the problems met in a vehicle project (i.e. problem descriptions contained in PMS) as resources being a part of the memory of this project.

Therefore, we developed a parser which, at the end of the process, generates a version of the ontology in RDF Schema (which is also the formalism required by the CORESE software). After RDF(S) generation, the annotations of the PMS problem-descriptions are automatically updated by SAMOVAR in the form of RDF statements.

4   Exploitation of the Ontologies

4.1   Use of the CORESE Tool

The ontologies set up were used to make annotations on the problem-descriptions from the PMS, considered as document elements. Their formalization in RDF Schema and the formalization of the annotations in RDF enabled to use the CORESE tool for information retrieval guided by such RDF(S) ontologies and annotations [8].

The CORESE tool implements a RDF(S) processor based on the conceptual graph (CG) formalism [30]. CORESE relies on RDF(S) to express and exchange metadata about documents. CORESE offers a query and inference mechanism based on the conceptual graph (CG) formalism. It may be compared to a search engine which enables inferences on the RDF statements by translating them into CGs.

CORESE translates the classes and properties of RDFS towards CG concept types and relation. CORESE also translates the base of RDF annotations into a base of CGs. This enables the user to ask queries to the RDF/CG base. A query is presented in the form of an RDF statement which is translated by CORESE into a query graph which is then projected on the CG base (using the projection operator available in CG formalism). The graphs results of this projection are then translated back into RDF for providing the user with the answers to his query. The projection mechanism takes into account the concept type hierarchy and the relation type hierarchy (obtained by translation of the RDF schemas).
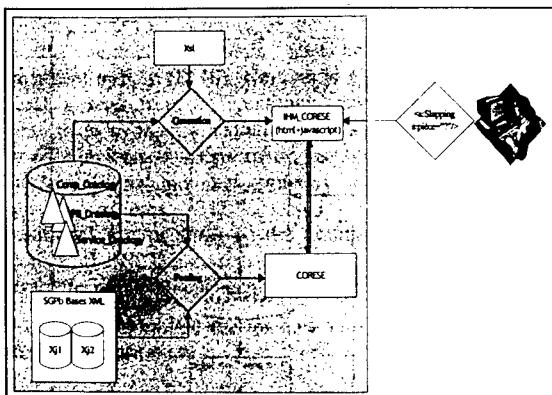
Figure 4: *Architecture of SAMOVAR*

To exploit CORESE, we formalised the SAMOVAR ontologies into RDFS. Then, we indexed the problem-descriptions of the PMS base with instances of concepts from these ontologies, while respecting the XML-based RDF syntax. After these two stages, the user could carry out information retrieval from the annotated problem-description base. The results of the user's query take into account not only the initial terms of the query but the links modeled in the different ontologies.

4.2   Examples of queries

Here are two examples in which we show that the problems extracted from texts and structured with hierarchical links allows us to find duplications of problem descriptions:
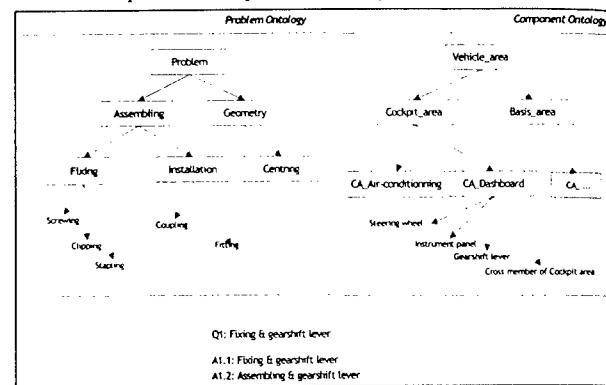
Figure 5: *Pathway for the ontologies to retrieve information*

In the first example, the user is looking for the problems of *fixing on the gearshift lever bellows*. A single answer is obtained:

T_Fixation	rdf:about=http://coco.tpz.tot.fr:8080/SAMOVARXML/MOXj1-02057.xml
libelle DIAMETRE DU SOUFFLET AU NIVEAU DU BOUTON PRESSION NON EN CONCORDANCE AVEC LE DIAMETRE DU POMMEAU DU SELECTEUR DE VITESSE **(VOIR PSXj2-00193)**	
piece SOUFFLET_DE_LEVIER_DE_VITESSE	

On the other hand, if the user extends her query to take into account more general concepts, following the ontological links (in our case - *assembling*), she will find a second case, which is effectively a similar problem-description.

Following a successive route through the ontologies thanks to the generalization and specialization links, the user can expand the query to find the subsuming concepts (cf. the fathers of the elements of the query) and the sibling concepts. In the example, the user can explore the *problems on gearshift lever*, level by level: from problems of fixing /connecting, she can go up to the father of this last concept (i.e. *Assembling*), and then go down to the other children concepts (e.g. *Installation*). The second case thus found is a similar problem-description to the first answer :

T_Montage rdf:about=http://coco.tpz.tot.fr:8080/SAMOVARXML/PSXj2-00193.xml
libelle BOUTON PRESSION DU SOUFFLET DE LEVIER DE VITESSE IMMONTABLE **(GEREE PAR MOXj1-02057)**
piece SOUFFLET_DE_LEVIER_DE_VITESSE

In the second example, the user would like to find the problems of centring on crossbar of cockpit area. The system

returns three cases among which two turn out to be problem-descriptions pointing mutually:

```
T_Centrage rdf:about= http://coco.tpz.tot.fr:8080/SAMOVARXML/MOXj1-00403.xm
    libelle FIXATIONS PDB : FIXATIONS LATERALE G ET COMPTEUR
DECENTRE SUR TRAVERSE.
    piece TRAVERSE_DE_POSTE_DE_CONDUITE
T_Centrage rdf:about=http://coco.tpz.tot.fr:8080/SAMOVARXML/MOXj1-02071.xml
    libelle FIXATION : SUPPORT CARMINAT SUR TRAVERSE DECENTREE.
(VOIR PSXj2-00023)
    piece TRAVERSE_DE_POSTE_DE_CONDUITE
T_Centrage rdf:about=http://coco.tpz.tot.fr:8080/SAMOVARXML/PSXj2-00023.xml
    libelle NON COAXIALITE DES TROUS DE FIXATION, SUPPORT
CALCULATEUR CARMINAT SUR TRAVERSE.(GEREE PAR MOXj1-02071)
    piece TRAVERSE_DE_POSTE_DE_CONDUITE
```

The browsing through the ontology lets the user browse the whole base of problem-descriptions, following the semantic axes modeled through links in the ontologies. This browsing helps the user to find similar problem-descriptions.

4.3   Evaluation of the ontologies for the search of similar problem-descriptions

The tests were made on the *Component* and *Problem* ontologies covering the corpus corresponding to an extract of the PMS base of a vehicle-project:

- a first step was concerning a specific perimeter (*Dashboard*) for 2 milestones,

- a second step processed the entire base of the project.

We created these ontologies taking the different information sources into account (official references cross-checked with items from the problem base). In professional terms the domain corresponds to the process of *assembling*. At present the *Dashboard* perimeter contains 118 concepts and 3 relations among which 22 components within 6 architectural areas, 12 sections and 3 levels reflecting the official *Component* referential. The *Problem* ontology contains about 43 types of problems. The *Service* ontology comprises 9 services extracted automatically from the base. These ontologies have been used to annotate around 351 problem-descriptions.

The whole base contains 792 concepts and 4 relations among which 467 components are structured in the same way, but updated with a typology of 39 component managers. The *Problem* ontology contains about 75 types of problems. The *Service* ontology contains about 38 types of services retrieved from base. These ontologies have been used to annotate around *4483* problem-descriptions.

4.3.1   *Discussion*
The first exploratory investigations on search of similar problem-descriptions have been proved to be interesting. All problem-descriptions mutually pointing have been found (in the case where problem-descriptions belong to the covered perimeter). Furthermore, there were less answers, but only the relevant ones.

So, we can conclude that good results are obtained thanks to the annotations of problem-descriptions with the instances of the problem types discovered from texts and structured in an ontology.

We can also notice that the modeling of the ontology is essential in this method. Test modifications in the *Problem*

ontology had more or less positive repercussions on the results. It is important to make sure of the validity of the ontology with the experts' support.

More generally, the method strongly depends on the corpus of the handled domain : if we reuse it for another domain, it will probably be necessary to update the heuristic rules allowing extraction of new concepts in order to cover the structures not processed. Indeed, the heuristic rules depend on the regularities found among the candidate terms extracted from the corpus.

Other « adjustments » were necessary during the process. For example, annotations with problems are at present performed by pattern matching : an annotation with a specific problem is activated as soon as the presence of some clues (for example Centring will be detected thanks to the presence of such clues as indexage, coaxiality, entraxe). According to the order of triggering of the rules, a problem-description can be annotated with instances of different ontology concepts. It would be interesting to order the rule triggering.

Besides, some other NLP tools (such as relation extractors [14] [28]) could help to refine furthermore the results of the *Problem* ontology construction.

As a further work, we intend to apply the same approach for building a *Solution* ontology (that would be connected to the *Problem* ontology). The same approach can be adopted: i.e. write heuristic rules from the manual analysis of the regularities of the candidate terms produced by Nomino and expressing possible solutions to the problems.

It would enable to index the problem-descriptions not only with instances of the concepts of the ontologies *Problem*, *Project*, *Service* and *Component*, but also with adequate instances of concepts of this *Solution* ontology.

5   Conclusions

5.1   Related Work
We have previously evoked several linguistic tools, dedicated to the extraction of terms and of relations from textual corpora. Among such tools, the choice of Nomino was due to both its relevance for our purposes and its availability. SAMOVAR can be compared to several approaches or tools integrating linguistic tools for extraction of candidate terms from a textual corpus.

Terminae [4] offers a methodology and an environment for building ontologies thanks to linguistic-based techniques of textual corpus analysis. The method is based on a study of the occurrences of terms in a corpus in order to extract the conceptual definitions and the environment helps the user in her modeling task by checking the characteristics of a new concept and by proposing potential family knot. Lexiclass [1] offers an interesting approach for building a regional ontology from technical documents. This tool enables the classification of syntagms extracted from a corpus, in order to help the knowledge engineer to discover important conceptual fields in the domain. Lexiclass coupled with Lexter, carries out a syntagm classification from Lexter according to the terminological context of the terms.

[3] describes a general method for building an ontology, method based on analysis of textual corpus using linguistic tools. The authors give the example of the Th(IC)2 project where they combine several tools for processing the textual corpus, each tool dedicated to a specific task (Lexter for terms extraction, Cameleon for relations, Terminae - for concept hierarchy construction) Our

method is situated in such a methodological framework: we use various specific tools in every step of the process, but with a corpus stemming from different origins (i.e. both interviews and textual data retrieved from existing databases). This variety characterizes the originality of our approach. [22], [23], [20] also present a general architecture for building an ontology from a textual corpus. [22], [23] exploit different linguistic tools so as to build a concept taxonomy and exploit a learning algorithm for mining non-taxonomic relations from texts.

The integration of CORESE in SAMOVAR and its ability to enable information retrieval thanks to annotations linked to the concepts of the ontologies thus build in a semi-automatic way is one originality of SAMOVAR. We must notice that SAMOVAR thus implements an approach for finding similar problems among past problem descriptions, which is a typical capability of case-based reasoning systems [26].

5.2 Further work

As noticed earlier, we will study heuristic rules for extraction of the *Solution* ontology from the textual corpus. Moreover, making explicit the links between the *Problem* and the *Solution* ontologies would enable to refine the indexing of the problem descriptions. Therefore, we will exploit a linguistic tool enabling the extraction of domain-dependent semantic relations, adapted to the automobile domain.

5.3 Towards a Method for Building a Project Memory

By finding information about similar problems processed during a given project, SAMOVAR began the process of capitalization in the company. It will be possible henceforth to spread it to wider scale - to exploit the incidents and the existing solutions between the various vehicle projects, to study problems and solutions within the same range or the same project. And in the longer term, exploit this capitalization to discover recurring problems in a company by re-showing weak spots "problems generators " to the engineering centres.

So SAMOVAR could enhance information sharing among the teams involved in the same or different vehicle projects.

We could exploit the SAMOVAR principles for other projects, provided that the right adaptations are carried out, especially at the level of the ontologies. We can thus generalize our approach to other domains than automobile design, for example to build and exploit a memory of the project of design or construction of any complex system, particularly regarding the memory of the problems encountered in such projects (e.g. incidents met during the design of a plane, a satellite, even a power plant, etc.). We propose a method relying on the following steps:

1. If there exists a database or a referential describing the components of this complex system, exploit it to build semi-automatically a *Component* ontology. Otherwise, use linguistic tools and method such as the ones described in [3] in order to build this Component ontology.
2. If there exists a description of a project characteristics in the considered company, exploit it to build a *Project* ontology. Otherwise, rely on interviews of the experts.
3. Establish a corpus of texts describing the problems met during one or several existing projects. It can involve texts

resulting from textual documents or from textual comments in databases.

4. Exploit some existing linguistic tools allowing the extraction of candidate terms (e.g. Lexter [5, 6] or Nomino).
5. Analyse manually (with the support of an expert) the regularities among the candidate terms which are liable to describe types of problems (resp. solutions). Then thanks to the regularities observed, write heuristic rules exploiting both these regularities and the *Component* and *Project* ontologies in order to suggest terms to include as concepts into the *Problem* (resp. *Solution*) ontology and even more to propose their position in this ontology. Validate such heuristic rules by the expert.
6. Use these heuristic rules and let an expert validate the propositions of the system obtained thanks to these heuristic rules.
7. Use the concepts of the *Problem, Solution, Component* and *Project* ontologies, so as to index automatically the elementary problem-descriptions (in the textual corpus) with instances of these concepts.
8. Exploit an RDFS generator for the ontologies and an RDF generator for the annotations, in order to be able to use the search engine CORESE to query the base annotated by the instances of problems.

The proposed methodology is generic. However the rules are constructed relying on the corpus: they reflect the existing structures of the corpus and are strongly connected to it. So, to apply the methodology for another domain it will be necessary to rebuild the heuristic rule base, so as to make it reflect the regularities observed in the corpus. This is typical of a methodology based on corpus analysis.

5.3.1 Acknowledgments

6 References

[1] Assadi H, Construction of a regional ontology form text and its use with a documentary system. In N. Guarino, ed. Proc. of the 1st Int. Conf. On Formal Ontology and Information Systems (FOIS'98), IOS Press, 1998.

[2] Aussenac-Gilles N, Biébow B, Szulman S, Corpus analysis for conceptual modelling, EKAW'2000 Workshop Ontologies and Texts, Juan-les-Pins, October 2-6, 2000 pages 13-20

[3] Aussenac-Gilles N, Biébow B, Szulman S, Revisiting Ontology Design : a Method Based on Corpus Analysis, In R. Dieng and O. Corby eds, Knowledge Engineering and Knowledge Management: Methods, Models and Tools, EKAW 2000, Juan-les-Pins, French Riviera, October 2-6, 2000, p. 172-188.

[4] Biébow B, Szulman S, Terminae : a linguistics-based tool for building of a domain ontology, In D. Fensel and R. Studer, eds, Knowledge Acquisition, Modeling and Management, Proc. of the 11th European Workshop (EKAW'99), LNAI 1621,. Springer-Verlag, 1999.

[5] Bourigault D, Lexter, un Logiciel d'Extraction de TERminologie, Application à l'acquisition des connaissances à partir de textes, PhD thesis, E.H.E.S.S, Paris, France, 1994

[6] Bourigault D, Lexter, a natural language processing tool for terminology extraction. Proc. of the 7th EURALEX Int. Congress, Goteborg, 1996.

[7] Brickley D. and Guha R.V. eds. Resource Description Framework (RDF) Schema Specification 1.0, W3C Candidate Recommendation 27 March 2000, http://www.w3.org/TR/rdf-schema

[8] Corby O, Dieng R , Hébert C, A Conceptual Graph Model for W3C Resource Description Framework, ICCS'2000, Springer-Verlag, Darmstadt, August 2000.

[9] Dieng R., Corby O., Giboin A. and Ribière M. Methods and Tools for Corporate Knowledge Management. In S. Decker and F. Maurer eds, International Journal of Human-Computer Studies, Special issue on Knowledge Management, 51:567-598, September 1999.

[10] Dieng R., Corby O., Giboin A., Golebiowska J., Matta N. and Ribière M. Méthodes et Outils pour la Gestion des Connaissances, Dunod, 2000.

[11] Enguehard C, ANA, Apprentissage Naturel Automatique d'un réseau sémantique, thèse de doctorat, UTC, 1992

[12] Enguehard C, and Pantera L. Automatic natural acquisition of terminology. Journal of Quantitative Linguistics, 2/1:27-32, 1995.

[13] D. Faure and C. Nédellec., In D. Fensel and R. Studer, editors, Proc. of the 11th European Workshop (EKAW'99), LNAI 1621,. Springer-Verlag, 1999.

[14] Garcia D, Analyse automatique des textes pour l'organisation causale des actions. Réalisation du système informatique COATIS, PhD thesis, Université de PARIS IV, Paris 1998

[15] Golebiowska J, SAMOVAR - Knowledge Capitalization in the Automobile Industry aided by Ontologies, PKAW 2000, Sydney, December 11-13, 2000.

[16] Golebiowska J, SAMOVAR - Setting up and Exploitation of Ontologies for capitalising on Vehicle Project Knowledge. In Aussenac-Gilles N., Biébow B., Szulman S., eds, Proc. of EKAW'2000 Workshop Ontologies and Texts, Juan-les-Pins, October 2000 pages 79-90

[17] Grefenstette G, Explorations in automatic thesaurus discovery, Kluwer Academic Publishers, Boston, 1994

[18] Hearst M, Automatic Acquisition of Hyponyms from Large Text Corpora, ICCL, COLING 92, Nantes July 25-28, 1992

[19] Jouis C, Contribution à la conceptualisation et à la Modélisation des connaissances à partir d'un analyse linguistique de textes. Réalisation d'un prototype : le système SEEK. Thèse de doctorat, 1993, EHESS.

[20] Kietz J.-U., Maedche A. and Volz R. A Method for Semi-Automatic Ontology Acquisition from a Corporate Intranet. In Aussenac-Gilles N., Biébow B., Szulman S., EKAW'2000 Workshop Ontologies and Texts, Juan-les-Pins, October 2-6, 2000 pages 37-50.

[21] O. Lassila and R. R. Swick eds. Resource Description Framework (RDF) Model and Syntax Specification, W3C Recommendation 22 February 1999, http ://www.w3.org/TR/REC-rdf-syntax

[22] Maedche A. and Staab S., Mining Ontologies from Texts. In Dieng R. and Corby O. eds, Knowledge Engineering and Knowledge Management: Methods, Models and Tools, EKAW 2000, Juan-les-Pins, French Riviera, October 2-6, 2000, p. 189-202.

[23] Maedche A. and Staab S., Discovering conceptual relations from text. Proc. of ECAI'2000, IOS Press, August 2000.

[24] Morin E. Acquisition de patrons lexico-syntaxiques caractéristiques d'une relation sémantique, TAL (Traitement Automatique des Langues), 1999

[25] Morin E Automatic acquisition of semantic relations between terms from technical corpora. Proc. of the 5th Int. Congress on Terminology and Knowledge Engineering (TKE'99), 1999.

[26] Moussavi M. - A Case-Based Approach to Knowledge Management, in Aha D.W. (Ed). Proc. of the AAAI'99 Workshop on "Exploring Synergies of Knowledge Management and Case-Based Reasoning". Juillet 1999; Orlando, FL. AAAI Press Technical Report WS-99-10.

[27] Séguéla P. and Aussenac-Gilles N.. Extraction de relations sémantiques entre termes et enrichissement de modèles du domaine. IC'99, pages 79-88, Paris, 1999.

[28] Séguela P, Adaptation semi-automatique d'une base de marqueurs de relations sémantiques sur des corpus spécialisés. Terminologies Nouvelles, 19:52-60, 1999.

[29] Séguéla P. Construction de modèles de connaissances par analyse linguistique de relations lexicales dans les documents techniques. PhD Thesis, Université de Toulouse, March 2001.

[30] Sowa J. F. Conceptual Graphs : Information Processing in Mind and Machine. Reading, Addison Wesley, 1984.

Ontology-Based Operators
for e-Business Model De- and Reconstruction

Jaap Gordijn
Vrije Universiteit - Vuture.net (Centre for e-business research)
Cisco Systems - Internet Business Solutions Group
De Boelelaan 1081a
1081 HV Amsterdam, The Netherlands
gordijn@cs.vu.nl

Hans Akkermans
Vrije Universiteit - Vuture.net (Centre for e-business research)
AKMC Knowledge Management
De Boelelaan 1081a
1081 HV Amsterdam, The Netherlands
hansakkermans@compuserve.com

Abstract
We define e-business models as conceptual models that show how a network of actors (a value constellation) creates, exchanges and consumes objects of value by performing value adding activities. In this paper we present a semi-formal ontology-based representation of e-business models that is useful in carrying out a preliminary business and requirements analysis. In particular, we show that a small set of generic 'model deconstruction' operators is able to generate design variations on a given e-business model, so that upfront analysis of the characteristics and consequences of a range of alternative e-business models becomes possible. We illustrate our ontology-based e^3-value approach by a commercial project on Internet news services.

Keywords
e-business model, reconstruction, ontology, e^3-value

INTRODUCTION
Successful e-business information systems are often characterized by *innovative* ways of doing business. This is usually called the *e-business model*. We define an e-business model as a conceptual model that shows how a network of actors creates, exchanges and consumes objects of *value* by performing value adding activities.

Finding such an e-business model is a creative task. We can, however, support this task by (1) an insightful way of representing e-business models, and (2) a way of finding and analyzing 'design' variations on such models.

To find variations on an initial e-business model, and consequently to assist in the elicitation of such a model, e^3-value defines e-business model *deconstruction* operators (inspired by [7, 3, 8]). These operators are part of an e-business model deconstruction and reconstruction process, during which we *de-assign* activities from their performing actors, try to find

alternative, and/or more activities by de-constructing existing ones, and re-assign newly found activities to executing actors. Because we assume that activities are profitable for at least one actor, re-assignment is possible. Essentially, to clarify discussions between stakeholders, we split the reconstruction process into two questions: (1) which value adding activities exist, and (2) which actors are to perform these activities?

In previous work we have introduced an ontology-based general representation of e-business models ([4], see recent publications at http://www.cs.vu.nl/~gordijn/research.htm). Based on this e^3-value ontology, we discuss in the present paper three generic operators for e-business model deconstruction: (1) the *value activity* deconstruction operator, which breaks an activity into smaller ones, but leaves the products/services offered or requested by the original activity to its environment unchanged, (2) the *value port* deconstruction operator, which breaks a service/product offered or requested by a value activity into smaller ones, and (3) the *value interface* deconstruction operator, which breaks combinations of value objects offered *and* counter-compensations requested into smaller pieces.

We illustrate e-business model de- and reconstruction by one of the e-business projects where we successfully applied our approach. The project at hand is about the provisioning of a value-added news service. With respect to such a service, a regular newspaper called the *Amsterdam Times* (a fictitious name, but based on an actual commercial e-business project) wants to offer to all its subscribers a service to read articles online using the Internet, but such that it will make hardly any additional costs. Therefore, the idea is to finance the execution of this business idea by the telephone connection revenues, which originate from the reader who has to set up a telephone connection for Internet connectivity.

This paper first introduces in brief the core concepts of our e^3-value methodology, which we use to formalize e-business models. Then we discuss both the general theory and an application of e-business model de- and reconstruction, and finally we present our conclusions.

E³-VALUE CORE CONCEPTS

To represent an e-business model, we use a *lightweight* ontology consisting of interrelated core concepts, and we utilize a well known *lightweight* scenario technique, called Use Case Maps [1]. This allows us to communicate e-business models easily to intended users such as business consultants, and CxO's. Moreover, the agility of e-business projects (the need to define, explore, and execute a business idea fast [5]) asks for an lightweight approach. Below, we discuss the ontological concepts and the UCM scenario concepts briefly (see Fig. 1 for an example). More information can be found in [1], [4].

The e³-value ontology

Actor. An actor is perceived by its environment as an independent economic (and often also legal) entity. By carrying out *value activities* (see below) an actor makes profit or increases its utility. In a sound and viable e-business model *every* actor should be capable of making a profit.

Value Object. Actors exchange value objects. A value object is a service, product or even a consumer experience. The important point here is that a value object is *of economic value* to one or more actors.

Value Port. An actor uses a value port to show to its environment that it wants to provide or request value objects. The concept of port (a notion adopted from engineering systems theory) is important, because it enables to abstract away from the internal business processes, and to focus only on how external actors and other components of the e-business value model can be 'plugged in'.

Value Interface. Actors have one or more value interfaces. A value interface groups individual value ports. It shows the value object(s) an actor is willing to exchange *in return for* other value object(s) via its ports. Such willingness is expressed by a decision function on the value interfaces, which shows on what conditions an actor wants to exchange a value object for another value object. The exchange of value objects is atomic at the level of the value interface. Either *all* exchanges occur as specified by the value interface or *none* at all.

Value Exchange. A value exchange is used to connect two value ports with each other. It represents one or more *potential* trades of value objects between actors.

Value Offering. A value offering is a set of value exchanges. It shows which value objects are exchanged via value exchanges *in return for* other value objects. A value offering should obey the semantics of the connected value interfaces: that is values are exchanged via a value interface on *all* its ports, or *none* at all.

Market segment. In the marketing literature [6], a market segment is defined as a concept that breaks a market (consisting of actors) into segments that share common properties.

Accordingly, our concept *market segment* shows a set of actors that share for value interfaces an equal decision function. We realize that in practice all actors behave differently and consequently cannot have equal decision functions. However, to be able to design understandable e-business models, we assume (as in marketing theory is done) that some groups of actors constitute equivalence classes with respect to their decision functions.

Value Activity. A value activity is *performed by* an actor and increases profit or utility *for* such an actor. The value activity is included in the ontology to discuss the *assignment* of value activities to actors. Value activities can be de-constructed into smaller value activities, but the requirement is that these still should be profitable or increase utility for the performing actor.

Use Case Maps

Scenario path. A scenario path consists of one or more *segments*, related by *connection elements* and *start-* and *stop stimuli*. It represents via *which* value interfaces objects of value must be exchanged, as a result of a start stimulus, *or* as result of exchanges via *other* value interfaces. Thus a scenario path shows causal relations between value interfaces.

Stimulus. A scenario path starts with a **start stimulus**. A start stimulus represents an event, possibly caused by an actor. If an actor causes an event, the start stimulus is drawn within the box representing the actor. The last segment(s) of a scenario path is connected to a **stop stimulus**. A stop stimulus indicates that the scenario path ends.

Segment. A scenario path has one or more segments. Segments are used to relate value interfaces with each other, possibly via connection elements, to show that an exchange on one value interface causes an exchange on another value interface. Using connection elements, sophisticated causal relations can be represented.

Connection. Connections are used to relate individual segments. An **AND fork** splits a scenario path into two or more sub path, while the **AND join** collapses sub path into one path. An **OR fork** models a continuation of the scenario path into one direction, to be chosen from a number of alternatives. The **OR join** merges two or path into on path. Finally, the **direct** connection interconnects two individual segments.

E-BUSINESS MODEL RECONSTRUCTION IN E³-VALUE

The e-business model reconstruction process consists of the following steps, which we discuss in the following sections in detail:

1. Identification of an initial e-business model.

2. Deconstruction of the initial e-business model.

3. Reconstruction of alternative e-business models.

The initial e-business model

The process of e-business model reconstruction starts with a representation of an initial e-business model. We assume the existence of an innovative e-business idea. Consequently, the goal of this step is to articulate that idea more precisely, so that stakeholders all have a common understanding about the idea.

The idea for the e-business model in this paper is to use a *termination* fee to finance a *news article online* service for subscribers on a regular newspaper. *Termination* means that if someone tries to set up a telephone connection by dialing a telephone number, another actor must pick up the phone, that is, *terminate* the connection. If someone is willing to *cause* termination of a large quantity of telephone calls, most telecommunication operators are willing to pay such an actor for that (the *termination fee*). Because the newspaper has a large subscriber base, it is capable to generate a large number of terminations for an *article online* service. This idea is formalized by an initial e-business model (see Fig. 1).

e-Business model deconstruction

For deconstruction, we de-assign actors from value activities, but leave value exchanges between value activities intact. Then, we repeatedly apply one of the three deconstruction operators. As we will show, it is possible to apply operators a number of times on an e-business model. The next sections discuss the three operators, along with their business rationale, and an example.

Value Activity deconstruction

Business rationale. Can we split a value activity, which initially is viewed as being performed as a whole by one actor, into smaller activities, together behaving as the original one, whereby each smaller activity potentially can be performed by individual actors?

Focus. The value activity de-constructor focusses on the *internal* structure of a value activity while keeping its value interfaces to the environment the same. It breaks down a value activity into smaller ones, for instance to allow specialized actors to perform one of these value activities.
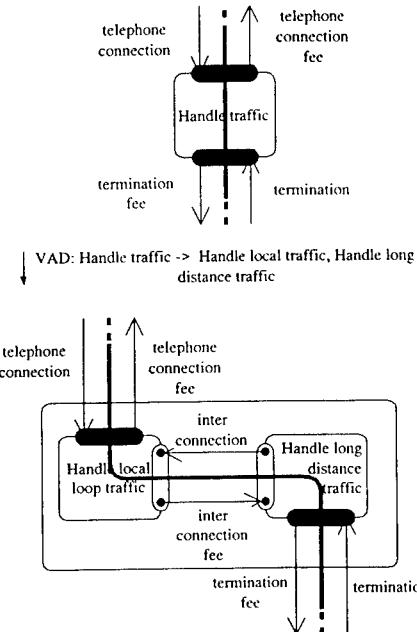
Operator $VAD : a \rightarrow a_1, ..., a_n$.

1. De-construct a value activity a with value interfaces $i_1, ...i_n$ into value activities $a_1, ..., a_n$.

2. Assign each value interfaces $i_1, ...i_n$ to one or more of the de-constructed value activities.

3. Add, if necessary, extra value interfaces to the de-constructed value activities, and relate these by value exchanges. Extra value interfaces and exchanges can be necessary to ensure that the de-constructed activities $a_1, ..., a_n$ are from an environment perspective equivalent to a.
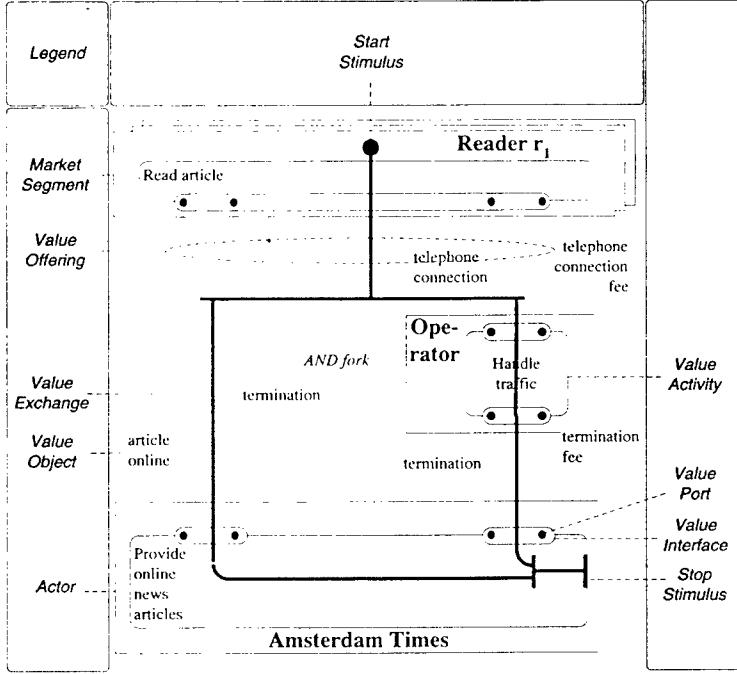
4. Reconsider scenario segments, which hit the value interfaces of value activity a.

It is possible that for a value activity a multiple alternative deconstructions exist.

Example: De-construct the *Handle traffic* value activity into two other value activities Fig. 2 de-constructs the *Handle traffic* value activity into two smaller value activities, which each can be potentially performed by a single (different) actor. The two value interfaces of *Handle traffic* can be found at the two smaller value activities, thereby providing the same interfaces to their environment as the original value activity. The value activity *Handle local traffic* offers end-to-end connectivity to a reader and gets paid for this, while it only exploits the local loop: the last miles from a local telephone switch to the reader. Consequently, this activity should 'buy' interconnection from the *Handle long distance traffic* activity, and pays for this in return. The latter activity exploits a telecommunication network between local telephone switches, and a web server for hosting news articles. Buying interconnection is shown by adding value interfaces and value exchanges between *Handle local traffic* and *Handle long distance traffic*. The scenario path is changed but hits the same value interfaces as was the case for the *Handle traffic* value activity.

Figure 2: Deconstruction of the value activity *handle traffic* into two value activities *handle local loop traffic* and *handle long distance traffic*.

Figure 1: The initial e-business model showing that the Amsterdam Times funds its service by a termination fee offered by a telecommunication operator. The reader offers a termination opportunity and a telephone connection fee and requests in return an article online and a telephone connection. The ports requesting/offering these value objects are grouped into one value interface from a reader's perspective because these objects are only of value in combination to the reader. By following the scenario path, it can be seen that the Amsterdam Times resells the termination to a telecommunication operator. This operator also receives a fee for a telephone connection, as result from reading an article. For each actor, initially one value activity is assumed that describes its value adding process at best.

Example: De-construct the *Provide Online news articles* **value activity into two other value activities** The deconstruction shown in Fig. 3 essentially separates the content creation (news) from the technical infrastructure needed to deliver content to the reader. It can be seen as out sourcing Internet service provisioning from a news provisioning perspective. Again we need to add value interfaces and value exchanges to represent that the *Provide news articles* value activity must acquire facilities for Internet service provisioning. Note that the scenario path for the de-constructed value activities hits the same value interfaces as the original value activity. However, internally, the scenario path splits to show that as a result of a termination/article online exchange, also a termination/termination fee *and* an Internet service provisioning/fee is necessary.

Value Port deconstruction
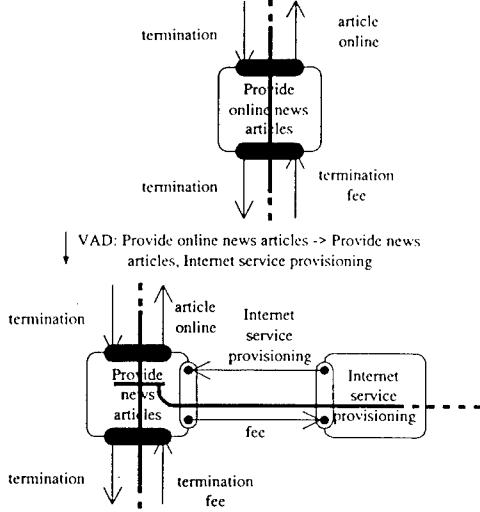
Business rationale. Can we split products, services or combinations into smaller products/services, which each can be delivered and consumed by individual actors?

Focus. Focus is to untangle offered or requested value objects, which still are of value for actors. These objects can

potentially be offered by multiple value activities rather than one, and thus by multiple actors. Because we change the value port, we change the value interface of a value activity to the environment.

Operator $VPD : p \rightarrow p_1, ... p_n.$

1. For each value port p in a value interface:

2. Consider deconstruction of value port p with value object o into value ports $p_1, ..., p_n$ with value objects $o_1, ... o_n$.

3. If deconstruction is possible, de-construct also the peer-ports of p. *Peer* ports are the ports p_i, which are connected by value exchanges to value port p. Note that a value port p can be connected to multiple other value ports p_i, representing that a value activity containing port p can exchange objects with multiple other value activities.

   (a) Dis-connect value exchanges connecting value port p and value ports p_i.

   (b) De-construct value ports p_i into ports $p_{1_i}, ..., p_{n_i}$ in the same way as p was de-constructed.
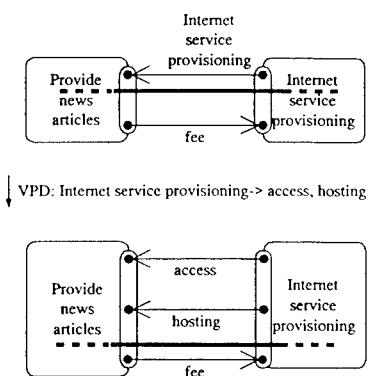
Figure 3: Deconstruction of value activity *provide online news articles* **into two value activities** *provide news articles* **and** *internet service provisioning.*

(c) Re-connect ports $p_1, ..., p_n$ using value exchanges with ports $p_{1_i}, ..., p_{n_i}$.

Example: De-construct the value object *Internet service provisioning* **into two other value objects** Fig. 4 deconstructs the value port *Internet service provisioning* into two different ports/value objects: (1) Internet *hosting* provisioning, e.g. hosting a web site, and (2) Internet *access* provisioning, e.g. exploiting a modem pool to offer access to the Internet.

Value Interface deconstruction

Figure 4: Deconstruction of the value object *Internet service provisioning* **into two value ports** *access* **and** *hosting.*

Business rationale. A value interface models the notion of *one good turn deserves another*, consisting of value objects offered and value objects requested in return. It is sometimes possible to split up a value interface in more interfaces, for (1) de-bundling, and (2) smaller value activities. Bundling refers to the business notion that an actor believes that if two or more products are offered as a whole, more money can be earned than offering these products seperately (see e.g. [2]). De-bundling refers to the opposite mechanism. We can also apply value interface deconstruction to split up the value activity associated with the interface at hand. Essentially, we split up an interface into smaller ones, whereby each value interface can be associated with a new value activity.

Focus. The focus is to find smaller value interfaces, that is value interfaces with a smaller number of value ports.
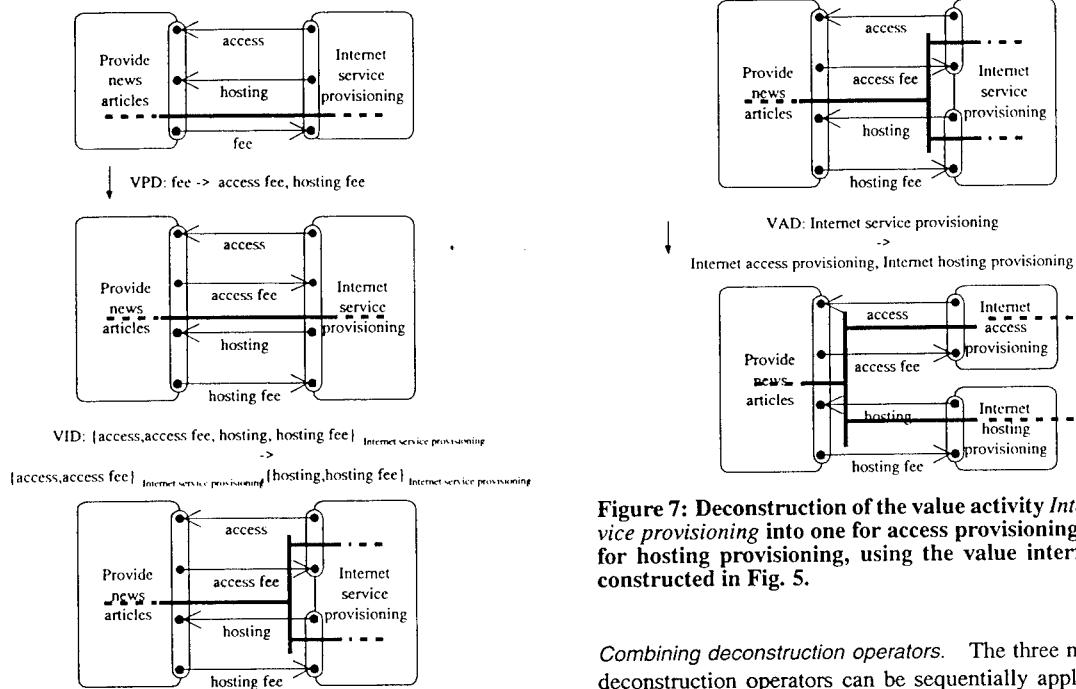
Operator $VID : i \rightarrow i_1, ..., i_n$.

1. For each value interface i with value ports $p_1, ..., p_n$ of a value activity a:

2. Find (alternative) value interfaces $i_1, ..., i_n$ grouping value ports $p_1, ..., p_n$.

3. Reconsider scenario segments.
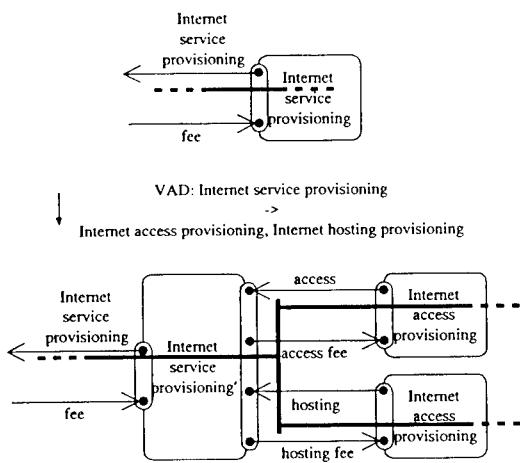
Example: An access and hosting value interface Fig. 5 introduces two separate value interfaces for the Internet service provisioning activity: one for offering Internet access and one for offering hosting services. Creation of these interfaces takes two steps. First we have to de-construct the *fee* port into two ports: the *access fee* and *hosting fee*. This is necessary due to the definition of value interface. A value interface models objects of value offered to the environment *and* the objects requested in return. We therefore need ports who receive the objects requested in return for offering *access* and *hosting* value objects. Second, we create two value interfaces, representing *hosting* and *access* services.

Note we do *not* split the value interface of the *Provide news articles* value activity. This value interface models that, for offering articles online, we need *both* hosting and access for each scenario occurrence.
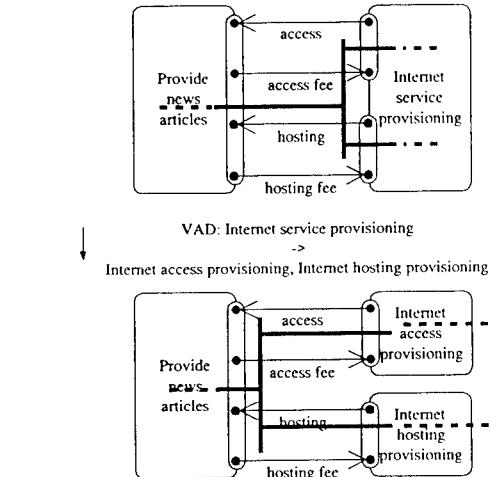
Example: Access and hosting via value *activity* **deconstruction** It also possible to split up the *Internet service provisioning* value activity into *Internet access provisioning* and *Internet hosting provisioning* (see Fig. 6), but there is an important difference compared to the previous example. Fig. 6 still shows a value activity called *Internet service provisioning'* (although smaller than the original one). This activity is profitable by offering a bundle of access and hosting services, but must buy-in access and hosting from another service. In contrast, in Fig. 5, the value activity *Provide news articles* is responsible for acquiring both access and hosting.

Figure 5: **Deconstruction of the value interface with four ports into two value interfaces with each two ports.**

Figure 6: **Deconstruction of the value activity** *Internet service provisioning* **into one for access provisioning and one for hosting provisioning. In contrast to Fig. 5, the** *Internet service provisioning* **ensures that their exist still one bundle of Internet service provisioning, while in Fig. 5 an actor who wants** *access* **and** *hosting* **must compose the bundle him/herself.**

Figure 7: **Deconstruction of the value activity** *Internet service provisioning* **into one for access provisioning and one for hosting provisioning, using the value interfaces deconstructed in Fig. 5.**

Combining deconstruction operators. The three mentioned deconstruction operators can be sequentially applied. The following three cases appear regularly:

- A number of sequential value activity deconstruction operations. In this case, we try to break up a value activity into (alternative) smaller ones, but do not change anything visible to the outside world.
- Value port deconstructions, followed by value interface deconstructions, and finally value activity deconstructions. In this case, we try to find smaller value objects which can be offered by separate value activities, which can be performed by individual actors. Fig. 7 is an example of this. First we de-construct the value interface of *Internet service provisioning* into two smaller ones for access and hosting (see Fig. 5), and then we de-construct the value activity into two smaller ones.
- De-bundling, a number of value port deconstructions, followed by value interface deconstructions. Fig. 5 can be seen as a case of de-bundling: we allow that the services *hosting* and *access* are sold separately rather than as a whole. Note that a value interface means that if a value object is exchanged via one of its ports, value objects on all its other ports must be exchanged too, so after de-bundling, access and hosting can be obtained as separate services rather than as a whole.

e-Business Model reconstruction

Deconstruction of an e-business model means de-assigning value activities and actors, and generating new value activities. During e-business model reconstruction, we study the re-assignment of value activities to performing actors.

First, we generate value activities *configurations*. These are connected value activities, by means of value exchanges, which represent an e-business model, *without* their performing actors. Because in this case study, we did not consider alternative deconstructions, so we have only one such a configuration (essentially Fig. 8 with omitted actors.).

Second, we re-identify actors, who are potentially interested in executing one or more value activities. Actors are potentially interested, if they expect to make a profit, or to increase utility by performing the value activity. Re-identification means that we consider new actors, which were not identified during development of the initial e-business model. It is reasonable to expect that by finding new, more specialized value activities, other actors than the ones already found are interested to perform these.

Third, we make an *actor-value activity assignment* matrix (see Table 1). This matrix shows actors, which are potentially interested in performing value activities of a specific configuration.

Finally, using the actor-value activity assignment matrix, alternative e-business models can be extracted and represented using our graphical technique. Fig. 8 shows one possible e-business model. Other models are possible by choosing other assignments of value activities to actors.

CONCLUSION

Finding innovative e-business models is a creative task. However, finding varations on such an e-business model can be facilitated by e-business model de- and reconstruction. The starting for this is to separate (1) which value adding activities exist from (2) which actors are performing these.

To find e-business model variations, we have defined three deconstruction operators, which all have a clear business rationale. The value activity deconstruction (*VAD*) operator helps in finding smaller value activities, which all can be profitably performed by at least one actor. We keep the value interface invariant using this operator, and only focus on the partitioning of a value activity over a number of actors rather than one actor.

A value interface models that an actor, or value activity, offers something of value to its environment, *and* wants something in return for that. The value interface deconstruction (*VID*) operator splits such interfaces into smaller ones. This may be done for two reasons. First, splitting can be done for unbundling reasons: the offering of value objects separately rather than as a bundle. Second, de-constructed value interfaces can be used to de-construct a value activity associated with these interfaces into smaller activities.

Finally, the value port deconstruction (*VPD*) operator assists in identifying new value ports/objects, based on an initial one, which each can de delivered or requested by individual actors. Mostly, the *VPD* operator is followed by the *VID*

operator to address unbundling, or by the *VAD* operator, to distribute the offering of the original value object over a number of actors.

Also, we have shown how these operators work out in a practical, non-trivial e-business modeling project. The representation proposed in this paper of e-business models appeared valuable in the project to illustrate complicated concepts such as call termination and interconnection to stakeholders, while the presented de-·and reconstruction process proved important to find new value activities, and to renegotiate assignment of these activities with the performing actors.
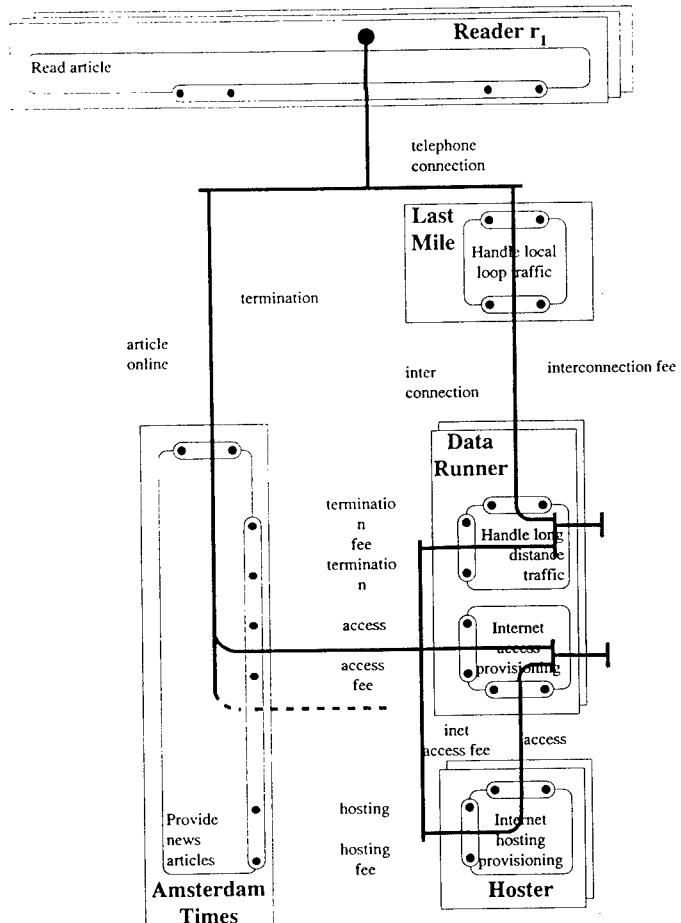
REFERENCES

1. R. Buhr. Use case maps as architectural entities for complex systems. *IEEE Transactions on Software Engineering*, 24(12):1131-1155, 1998.

2. S.-Y. Choi, D. O. Stahl, and A. B. Whinston. *The economics of doing business in the electronic marketplace*. Macmillan Technical Publishing, Indianapolis, IN, 1997.

3. P. Evans and T. S. Wurster. *Blown to Bits - How the New Economics of Information Transforms Strategy*. Harvard Business School Press, Boston, MA, 2000.

4. J. Gordijn, J. Akkermans, and J. van Vliet. What's in an electronic business model. In R. Dieng and O. Corb, editors, *Knowledge Engineering and Knowledge Management - Methods, Models, and Tools, 12th International Conference (EKAW 2000)*, volume LNAI 1937, pages 257–273, Berlin, D, 2000. Springer Verlag. Also available from http://www.cs.vu.nl/~gordijn.

5. A. Hartman, J. Sifonis, and J. Kador. *Net Ready - Strategies for Success in the E-conomy*. McGraw-Hill, New York, NY, 2000.

6. P. Kotler. *Marketing management: analysis, planning, implementation and control*. Prentice Hall, Englewood Cliffs, NJ, 1988.

7. D. Tapscott, D. Ticoll, and A. Lowy. *Digital Capital - Harnessing the Power of Business Webs*. Nicholas Brealy Publishing, London, UK, 2000.

8. P. Timmers. *Electronic Commerce: Strategies and Models for Business-to-Business Trading*. John Wiley & Sons Ltd., Chichester, UK, 1999.

Table 1: Actor - Value activity matrix showing which actors can potentially perform which value activity, *and* while creating profit, or increasing utility by doing an activity.

Value activity	Actor				
	Reader	Last Mile	Data Runner	Hoster	Amsterdam Times
Read article	x				
Handle local loop traffic		x	x		
Handle long distance traffic		x	x		
Provide internet access		x	x	x	x
Hosting		x	x	x	x
Provide news articles					x

Figure 8: A re-constructed e-business model by assigning newly discovered value activities to actors.

Joint Knowledge Capture for Grammars and Ontologies

Udo Hahn   Kornél G. Markó
Ⓒⓕ Text Knowledge Engineering Lab
Albert-Ludwigs-Universität Freiburg
D-79098 Freiburg, Germany
http://www.coling.uni-freiburg.de

Abstract

We introduce a methodology for automating the maintenance and growth of domain-specific concept taxonomies and grammatical class hierarchies simultaneously, based on knowledge capture from natural language texts. The assimilation process is centered around the linguistic and conceptual 'quality' of various forms of evidence underlying the generation, assessment and on-going refinement of lexical and concept hypotheses. On the basis of the strength of evidence, hypotheses are ranked according to plausibility, and the most reasonable ones are selected for assimilation into the given lexical class hierarchy and domain ontology.

INTRODUCTION

Intelligent systems require knowledge-rich resources to reason with. As their creation is usually delegated to human experts who are slow and costly, these systems face the often deplored knowledge acquisition bottleneck. The knowledge supply challenge is even more pressing when *various* knowledge sources have to be provided within the framework of a single system, all at the same time. This is typically the case for knowledge-intensive natural language processing (NLP) systems which require simultaneous feeding with a lexical inventory, morphological and syntactic rules or constraints, and semantic as well as conceptual knowledge.

Each of these subsystems embody an enormous amount of specialized component knowledge on its own. Much emphasis has already been put on providing machine learning support for single of these components — morphological [4], lexical [17, 20], syntactic [3, 9, 12], semantic [5, 11, 16] and conceptual knowledge [10, 22, 8, 13]. But only Cardie [2] has made an attempt so far to combine these isolated streams of linguistic knowledge acquisition within a common approach, i.e., to learn *different* types of relevant NLP knowledge *simultaneously*.

We also propose such an integrated approach for learning lexical/syntactic and conceptual knowledge. New concepts are acquired and positioned in the concept taxonomy, as well as the grammatical status of their lexical correlates is learned taking two knowledge sources into account. *Domain knowledge* provides a concept and role taxonomy which serves as a comparison scale for judging the plausibility of newly derived concept descriptions in the light of that prior knowledge. *Grammatical knowledge* contains a type hierarchy of lexical classes which make increasingly restrictive grammatical constraints available for linking an unknown word with its corresponding word class. Our model makes explicit the kind of qualitative reasoning that is behind these multi-threaded learning processes [21, 8].

A LEARNING SCENARIO

Consider a learning scenario as depicted in Figure 1 from a grammatical perspective and in Figure 2 from a conceptual one. Suppose, your knowledge of the information technology domain tells you nothing about *Itoh-Ci-8*. Imagine, your favorite technology magazine features an article starting with *"The Itoh-Ci-8 has a size of ... "*. Has your knowledge increased? If so, what did you learn from just this phrase?

The learning process starts upon the reading of the unknown word *"Itoh-Ci-8"*. In this initial step, the corresponding hypothesis space incorporates all the top level concepts available in the ontology for the new lexical item *"Itoh-Ci-8"*. So, the concept ITOH-CI-8 may be an OBJECT, an ACTION, a DEGREE, etc. (cf. Figure 2). Similarly, from a grammatical viewpoint (cf. Figure 1), the lexical item *"Itoh-Ci-8"* can be hypothesized as being an instance of one of the top-level part-of-speech categories, e.g., a NOMINAL, an ADVERB or a VERBAL.[1] Due to grammatical constraints, however, (*"Itoh-Ci-8"* directly follows *"The"*) the VERBFINITE hypothesis and more specialized ones can be immediately rejected (cf. the darkly shaded box in Figure 1).

While processing the noun phrase *"The Itoh-Ci-8"* as the subject of the verb *"has"*, the ADVERB hypothesis, as well as

---

[1] While the distinction between NOMINAL and VERBAL should be obvious, the prominent role of ADVERB at the top level of word class categories might not be. NOMINAL as well as VERBAL carry grammatical information such as case, gender, number, or tense, mood, aspect, respectively, none of which is shared by ADVERBs. As class hierarchies derive from the principle of property inheritance, and ADVERBs lack common features with other word classes, they form an independent class on their own. This explains the prominent role of ADVERB at this high level of abstraction.
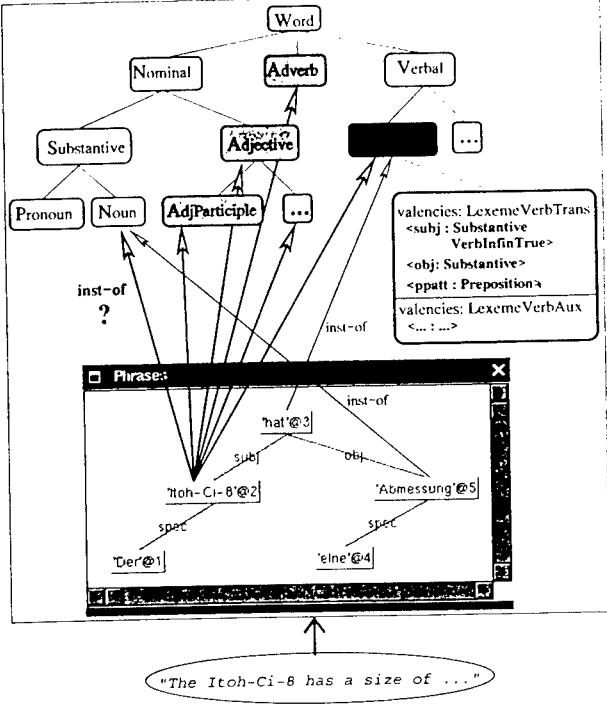
Figure 1: Sample Scenario — Grammatical Learning

Figure 2: Sample Scenario — Conceptual Learning

specialized NOMINAL hypotheses such as ADJECTIVE, including participles acting as adjectives, ADJPARTICIPLE (cf. the grey shaded boxes in Figure 1), become invalid (none of the instances of any of these word classes must intervene a determiner and a finite verb directly), still leaving the SUBSTANTIVE hypothesis intact, which is also derived from NOMINAL. Additional supportive evidence for the latter comes from the part-of-speech constraints imposed by the subj (or, alternatively, by the obj) dependency relation (cf. the valency requirements attached to the verb *"has"* in Figure 1). The equally possible VERBINFIN alternative, however, is ruled out due to violating syntactic evidence. Since *"Itoh-Ci-8"* is not a pronoun (it does not match this closed list), we hypothesize it to be a NOUN, finally.

From a semantical perspective (cf. Figure 2), the concept ITOH-CI-8, at this stage of analysis, is related via a specialized AGENT role to the ACTION concept POSSESSION, the concept denoted by *"has"* (lexical ambiguities, e.g., for the verb *"has"* as an auxiliary or full verb, lead to the creation of alternative hypotheses). Since POSSESSION requires its AGENT to be an OBJECT, ACTION and DEGREE are no longer valid concept hypotheses for ITOH-CI-8. Their cancellation (cf. the darkly shaded boxes in Figure 2) yields already a significant reduction of the huge initial concept search space. The learner then aggressively specializes the remaining single hypothesis to the immediate subordinates of OBJECT, e.g., PHYSICALOBJECT and TECHNOLOGY, in order to test more restrictive hypotheses which – due to more specific constraints – are easier falsifiable.

Just as subject was mapped to the AGENT role, the semantic constraints for the verb *"has"* also indicate that the grammatical direct object relation is to be interpreted in terms of a conceptual PATIENT role. Accordingly, the phrase *" ... has a size of ... "* is processed such that $size.1$ is the PATIENT of the POSSESSION relationship. Subsequent interpretation steps combine the fillers of the AGENT and PATIENT roles so that the following terminological expressions are asserted:

(P1) $size.1$: PHYSICALSIZE
(P2) $Itoh\text{-}Ci\text{-}8.1$ HAS-SIZE $size.1$

Assertion (P1) indicates that $size.1$ is an instance of the concept class PHYSICALSIZE and (P2) relates $size.1$ and $Itoh\text{-}Ci\text{-}8.1$ via the binary relation HAS-SIZE. Given the conceptual roles attached to PHYSICALSIZE, the system recognizes that all specializations of PRODUCT can be related to the concept PHYSICALSIZE (via the role SIZE-OF), while for TECHNOLOGY no such relation can be established. So, we favor the conceptual reading of ITOH-CI-8 as a kind of a PRODUCT and suppress the TECHNOLOGY hypothesis (cf. the grey-shaded box in Figure 2).

At this initial stage, we, finally, come up with two preliminary assumptions – grammatically, we consider the lexical item *"Itoh-Ci-8"* as a NOUN, while conceptually, we interpret ITOH-CI-8 as a PRODUCT.
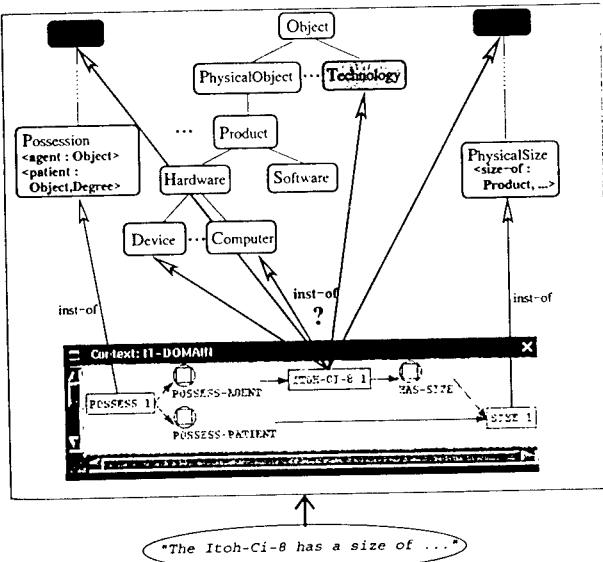
THE LEARNING MODEL

The system architecture for elicitating conceptual and grammatical knowledge from texts is summarized in Figure 3. It depicts how linguistic and conceptual evidence are generated and combined to continuously discriminate and refine the set of word class and concept hypotheses (the unknown item yet to be learned is characterized by the black square).
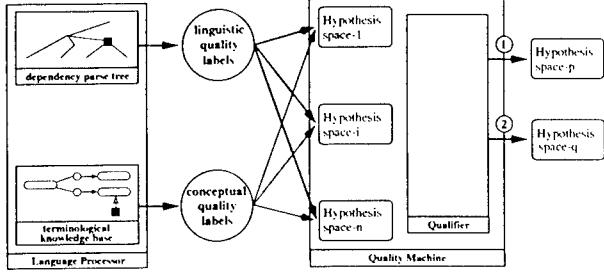
Figure 3: Architecture for Joint Ontology and Grammar Learning

Grammatical knowledge for syntactic analysis is based on a fully lexicalized dependency grammar [7]. Such a grammar captures binary valency constraints between a syntactic head (e.g., a noun) and possible modifiers (e.g., a determiner or an adjective). These include restrictions on word order, compatibility of morphosyntactic features and semantic integrity conditions. For a dependency relation $\delta \in \mathcal{D} := \{$specifier, subject, dir-object, ...$\}$ to be established between a head and a modifier, all valency constraints must be fulfilled.

In this object-oriented approach, lexeme specifications form the leaf nodes of a lexical class hierarchy. Grammatically related lexical items are grouped together in terms of specific word classes which are further abstracted in terms of increasingly general word class specifications. Different levels of generality in this hierarchy reflect stronger or weaker constraints these classes embody. This word class taxonomy consists of word class names $\mathcal{W} := \{$VERBAL, VERBFINITE, SUBSTANTIVE, NOUN, ...$\}$ and a subsumption relation $isa_{\mathcal{W}}$ = $\{$(VERBFINITE, VERBAL), (NOUN, SUBSTANTIVE), ...$\}$ $\subset \mathcal{W} \times \mathcal{W}$, which characterizes specialization relations between word classes.

The *language processor* [6] yields structural dependency information from the grammatical constructions in which an unknown lexical item occurs in terms of the corresponding *parse tree*. Whenever a parse tree incorporating an unknown word is generated, an implicit word class hypothesis for this item is made, since after committing to this word class all its valency constraints are met. All the different syntactic constructions (e.g., genitive, apposition, comparative), in which unknown lexical items appear, are recorded and assessed later on relative to the credit they lend to a particular hypothesis.

Conceptual knowledge is expressed in terms of a KL-ONE-like knowledge representation language [24]. A domain ontology consists of a set of concept names $\mathcal{F} := \{$COMPANY, HARD-DISK, ...$\}$ and a subsumption relation $isa_{\mathcal{F}}$ = $\{$(HARD-DISK, STORAGEDEVICE), (IBM, COMPANY), ...$\}$ $\subset \mathcal{F} \times \mathcal{F}$. Concepts are linked by conceptual relations. The corresponding set of relation names $\mathcal{R} := \{$HAS-PART, DELIVER-AGENT, ...$\}$ contains the labels of conceptual relations which are also organized in a subsumption hierarchy $isa_{\mathcal{R}}$ = $\{$(HAS-HARD-DISK, HAS-PHYSICAL-PART), (HAS-PHYSICAL-PART, HAS-PART), ...$\}$.

The semantic interpretation of dependency parse trees [19] involving unknown lexical items and their conceptual correlates in the *terminological knowledge base* forms the basis for the derivation of concept hypotheses within alternative *hypothesis spaces*. Each hypothesis they contain is further enriched by conceptual annotations reflecting structural patterns of consistency, mutual justification, analogy, etc. This kind of initial evidence, in particular its predictive "goodness" for the learning task, is represented by corresponding sets of linguistic and conceptual quality labels.

Linguistic quality labels reflect structural properties of phrasal patterns in which unknown lexical items occur — we assume that the type of grammatical construction exercises a particular interpretative force on the unknown item and, at the same time, yields a particular level of credibility for the hypotheses being derived therefrom. Appositive constructions (*"the laser printer X"*), e.g., constrain the conceptual status of the unknown item much more than, e.g., genitives (*"X's price"*). Hence, the linguistic quality label APPOSITIVE ranks higher than GENITIVE.

Conceptual quality labels capture the degree of structural similarity, compatibility, etc. when the representation structures of a concept hypothesis are compared with those of alternative concept hypotheses or a priori representation structures in the underlying domain knowledge base. The closer the match with given knowledge, the more credit is lent to a hypothesis. For instance, a very positive conceptual quality label, M-DEDUCED, is assigned to multiple derivations of the *same* concept hypothesis in *different* hypothesis (sub)-spaces, a definitely negative one is INCONSISTENT, which annotates contradictory hypotheses.

Multiple concept hypotheses which represent alternative readings of each unknown lexical item are organized in terms of corresponding *hypothesis spaces*, each one holding a different or a further specialized concept hypothesis. The *quality machine* estimates the overall credibility of single concept hypotheses by taking the assembled set of quality labels for each hypothesis into account. The final computation of a preference order for the entire set of competing hypotheses takes place in the *qualifier*, a terminological classifier extended by an evaluation metric for quality-based selection criteria. The output of the quality machine is a ranked list of plausible concept hypotheses (for a formal specification of the underlying quality calculus, cf. [21]).

THE LEARNING SCENARIO REVISITED

Depending on the type of the syntactic construction in which the unknown lexical item occurs, different hypothesis generation rules may fire. Genitives, such as *"The switch of the Itoh-Ci-8..."*, place by far fewer constraints on the item to be learnt than, say, appositives like *"The laser printer Itoh-Ci-8 ..."*. In the following, let *target* be the unknown item (*"Itoh-Ci-8"*) and *base* be the known item (*"switch"*). The main conceptual constraint for genitives requires the target concept to fill one of the n roles attached to the base concept. Since without additional evidence the correct role cannot yet be decided upon, n alternative, equally likely hypotheses have to be posited (unless additional constraints apply). Following on that, the target concept is assigned as a tentative filler of the i-th role of the base concept in the corresponding i-th hypothesis space. The classifier then immediately derives a suitable concept hypothesis by specializing the target concept according to the concept class restriction of the base concept's i-th role (cf. [10] for a similar constraint propagation mechanism). The hypothesis generation rule also assigns a syntactic quality label to the i-th hypothesis indicating the type of syntactic construction in which target and base co-occur (here, GENITIVE).

After the processing of *"The Itoh-Ci-8 has a size of..."*, the target ITOH-CI-8 is already predicted as a PRODUCT. Prior to continuing with the phrase *"The switch of the Itoh-Ci-8 ..."*, consider a fragment of the conceptual representation for SWITCHes:

(P3) SWITCH-OF $\doteq$ SWITCH|PART-OF|HARDWARE

(P4) SWITCH $\doteq$
$\forall$HAS-PRICE.PRICE $\sqcap$
$\forall$HAS-WEIGHT.WEIGHT $\sqcap$
$\forall$SWITCH-OF. $\left(\begin{array}{c} \text{OUTPUTDEV} \sqcup \text{INPUTDEV} \sqcup \\ \text{STORAGEDEV} \sqcup \text{COMPUTER} \end{array} \right)$

The relation SWITCH-OF is defined by (P3) as the set of all PART-OF relations which have their domain restricted to SWITCH and their range restricted to HARDWARE. In addition, (P4) reads as "all fillers of HAS-PRICE, HAS-WEIGHT, and SWITCH-OF roles must be concepts subsumed by PRICE, WEIGHT, and the disjunction (OUTPUTDEV $\sqcup$ INPUTDEV $\sqcup$ STORAGEDEV $\sqcup$ COMPUTER), respectively". So, three roles have to be considered for relating the target ITOH-CI-8, as a tentative PRODUCT, to the base concept SWITCH. Two of them, HAS-PRICE and HAS-WEIGHT, are ruled out due to the violation of a simple integrity constraint (PRODUCT does not denote a unit of measure). Therefore, only the role SWITCH-OF must be considered. Given the definition of SWITCH-OF in (P3), ITOH-CI-8 is immediately specialized to HARDWARE by the classifier (cf. also Figure 2). Since the classifier aggressively pushes hypothesizing to be maximally specific, four distinct hypotheses are immediately created due to the range restriction of the role SWITCH-OF as expressed in (P4), *viz.* OUTPUTDEV, INPUTDEV, STORAGEDEV and COMPUTER, and they are managed in four

distinct hypothesis spaces, h_1, h_2, h_3, and h_4, respectively. Within h_1, h_2, and h_3, DEVICE, their common superconcept, is *multiply* derived by the classifier, too. Accordingly, this hypothesis is assigned a high degree of confidence by issuing the conceptual quality label M-DEDUCED.

EVALUATION

The knowledge base on which we performed our experiments contained approximately 3,000 concepts and relations from the information technology (IT) domain, the grammatical class hierarchy was composed of 80 word classes. We randomly selected 48 texts from a corpus of IT magazines, with approximately 9,000 word tokens. So, 48 descriptions of new products were to be learnt.

Grammar Learning

The task of grammar learning is to predict the most specific word class for an unknown lexical item, given a hierarchy which covers all relevant word classes for a particular natural language. Most relevant for the learning task are valence specifications of the word classes and various word order constraints. Furthermore, morphosyntactic feature constraints have to be incorporated, when possible. These constraints are checked in a top-down manner, i.e., (cf. our discussion related to Figure 1) we start from pretty general word class hypotheses which are continuously refined as more discriminatory evidence comes in.

Following previous work on evaluation measures for learning systems in the framework of NLP [10], we distinguish here the following parameters:

- **Hypotheses** denote the set of grammatical class hypotheses derived by the system as the final result of the text understanding process, for each target item;
- **Correct** denotes the number of cases in which **Hypotheses** contain the correct grammatical class description for the target item;
- **OneCorrect** denotes the number of cases in which **Hypotheses** is a singleton set, i.e., contains only the correct grammatical class description;
- **AllHypos** denote the number of word class hypotheses generated by the system for all target items.

The data in Table 1 indicate that the system dealt with 75 instances of unknown lexical items. This number includes cases of word class ambiguities, as well as instances of word classes other than SUBSTANTIVEs (but excluding occurrences of VERBALs). In 73 of the 75 cases word class hypotheses could be generated (in two cases data was so weak that no hypothesis could be created). In 66 of the 73 cases the set of word class hypotheses for an unknown lexical item contained the correct prediction, while for 37 lexical items there was only one and correct hypothesis.

For determining precision and recall, we faced the following problem. At the end of the full learning cycle various word class hypotheses may still remain valid for one unknown lex-
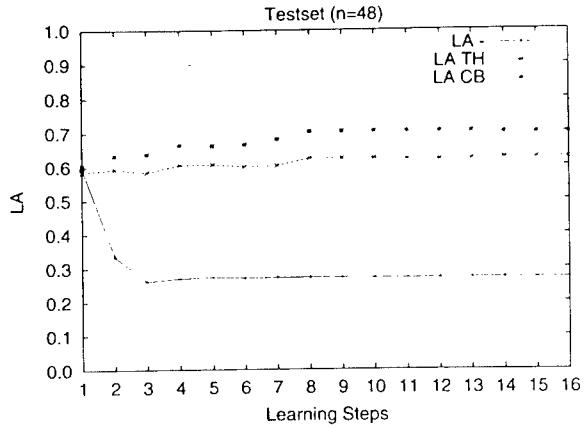
Figure 8: Learning Accuracy (LA) under Optimal Conditions

Figure 9: Number of Hypotheses (NH) under Optimal Conditions

The almost dramatic decrease in learning accuracy for the pure terminological learner (LA-) can be explained by the observation that as the number of learning steps increase the number of hypothesis spaces (they contain alternative readings for a concept only one of which may be the correct one) increase as well (cf. Figure 9). Since only the quality calculus prunes the growing hypothesis spaces by eliminating implausible ones, there is no wonder that in a pure terminological approach learning accuracy falls below 30%. As can be seen from the considerable difference between **LA-** on the one hand, and **LA CB/LA TH** on the other, as well as from the small corridor between **LA CB** and **LA TH**, grammatical evidence is the driving factor for learning accuracy, while conceptual criteria only slightly improve the final results. The same holds for the realistic scenario, too, indicating the influence of seemingly general regularities.

Under optimal learning conditions the hypothesis spaces reveal a quasi-logarithmic growth behavior (cf. Figure 9). More specifically, the pure terminological learner generates almost double as many hypothesis spaces (7.3 on the average) as the quality calculus operating with linguistic evidence only (3.7 on the average). A significant selection of hypothesis spaces, however, results from the application of the full quality calculus (1.3 on the average), i.e., the incorporation of conceptual criteria. The lower average number in the optimal case is due to a more precise analysis, whereas in the realistic scenario corrupted syntactic parses have to be considered, too. In any case, the data reveals the superior discrimination power contained in the full quality calculus when it comes to the choice between concept alternatives.

DISCUSSION AND CONCLUSIONS

Knowledge-based systems provide powerful means for reasoning, but it takes a lot of effort to equip them with the knowledge they need, usually by manual knowledge engineering. In this paper, we have argued for an alternative solution. It is based on an automatic learning methodology in which concept and grammatical class hypotheses emerge as
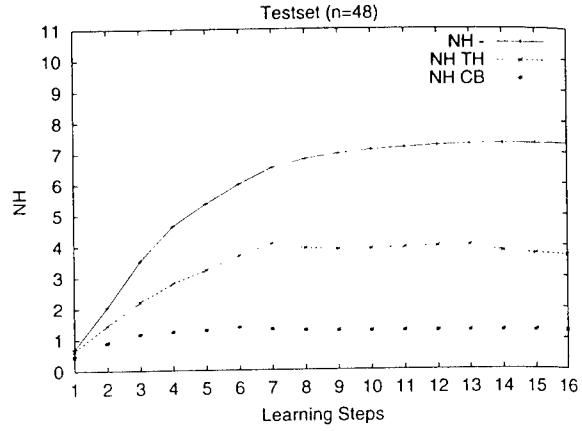
a result of the incremental assignment and evaluation of the quality of linguistic and conceptual evidence related to unknown words. No specialized learning algorithm is needed, since learning is a (meta)reasoning task carried out by the classifier of a terminological reasoning system [21].

This distinguishes our methodology from Cardie's case-based approach [2] which also combines conceptual and grammatical learning, but where the actual learning task is delegated to the C4.5 decision tree algorithm. Cardie's approach also requires some supervision (interactive grammatical encoding of the context window surrounding the unknown word), while our method operates entirely unsupervised. We share with her the view, however, that learning should encompass several linguistic dimensions simultaneously (parts of speech, semantic and conceptual encodings) within a unified approach, and should also avoid any explicit hand-coding heuristics to drive the acquisition process.

The work closest to ours with respect to the ontology learning problem has been carried out by Rau et al. [15] and Hastings and Lytinen [10]. They also generate concept hypotheses from linguistic and conceptual evidence. Unlike our approach, their selection of hypotheses depends only on an ongoing discrimination process based on the availability of this data but does not incorporate an inferencing scheme for reasoned hypothesis selection. The crucial role of quality considerations becomes obvious when one compares plain and quality-annotated terminological reasoning for the learning task. In the light of our evaluation study (cf. Figure 6 and 8, final learning step) the difference amounts to 23% under realistic experimental conditions, and 43% under optimal conditions, respectively, distinguishing between **LA -** (plain terminological reasoning) and **LA CB** values (terminological metareasoning based on the quality calculus).

REFERENCES

1. E. Brill. Transformation-based error-driven learning and natural language processing: A case study in part-of-speech tagging. *Computational Linguistics*, 21(4):543–565, 1995.

2. C. Cardie. A case-based approach to knowledge acquisition for domain-specific sentence analysis. In *AAAI'93 – Proceedings of the 11th National Conference on Artificial Intelligence*, pages 798-803. Washington, D.C., July 11-15, 1993. Menlo Park, CA & Cambridge, MA: AAAI Press & MIT Press, 1993.

3. E. Charniak. Tree-bank grammars. In *AAAI'96//IAAI'96 – Proceedings of the 13th National Conference on Artificial Intelligence & 8th Innovative Applications of Artificial Intelligence Conference*, volume 2, pages 1031-1036. Portland, Oregon, August 4-8, 1996. Menlo Park, CA & Cambridge, MA: AAAI Press & MIT Press, 1996.

4. B. Daille. Morphological rule induction for terminology acquisition. In *COLING 2000 – Proceedings of the 18th International Conference on Computational Linguistics*, volume 1, pages 215-221. Saarbrücken, Germany, 31 July - 4 August, 2000. San Francisco, CA: Morgan Kaufmann, 2000.

5. F. Gomez, C. Segami, and R. Hull. Determining prepositional attachment, prepositional meaning, verb meaning and thematic roles. *Computational Intelligence*, 13(1):1-31, 1997.

6. U. Hahn, N. Bröker, and P. Neuhaus. Let's PARSETALK: Message-passing protocols for object-oriented parsing. In H. Bunt and A. Nijholt, editors, *Advances in Probabilistic and Other Parsing Technologies*, volume 16 of *Text, Speech and Language Technologies*, pages 177-201. Dordrecht, Boston: Kluwer, 2000.

7. U. Hahn, S. Schacht, and N. Bröker. Concurrent, object-oriented natural language parsing: The PARSETALK model. *International Journal of Human-Computer Studies*, 41(1/2):179-222, 1994.

8. U. Hahn and K. Schnattinger. A text understander that learns. In *COLING/ACL'98 – Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics & 17th International Conference on Computational Linguistics*, volume 1, pages 476-482. Montréal, Canada, August 10-14, 1998. San Francisco, CA: Morgan Kaufmann, 1998.

9. M. Haruno, S. Shirai, and Y. Ooyama. Using decision trees to construct a practical parser. In *COLING/ACL'98 – Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics & 17th International Conference on Computational Linguistics*, volume 1, pages 505-511. Montréal, Canada, August 10-14, 1998. San Francisco, CA: Morgan Kaufmann, 1998.

P. M. Hastings and S. L. Lytinen. The ups and downs of lexical acquisition. In *AAAI'94 – Proceedings of the 12th National Conference on Artificial Intelligence*, volume 1, pages 754-759. Seattle, Wash., July 31 - August 4, 1994. Menlo Park, CA: AAAI Press & MIT Press, 1994.

. M. A. Hearst. Automated discovery of WORDNET relations. In C. Fellbaum, editor, WORDNET: *An Electronic Lexical Database*, pages 131-151. Cambridge, MA: MIT Press, 1998.

. J. C. Henderson and E. Brill. Bagging and boosting a Treebank parser. In *NAACL 2000 – Proceedings of the 1st Meeting of the North American Chapter of the Association for Computational Linguistics*, pages 34-41. Seattle, Wash., USA, April 29 - May 4, 2000. San Francisco, CA: Morgan Kaufmann, 2000.

13. A. Maedche and S. Staab. Discovering conceptual relations from text. In W. Horn, editor, *ECAI 2000 – Proceedings of the 14th European Conference on Artificial Intelligence*, pages 321-325. Berlin, Germany, August 20-25, 2000. Amsterdam: IOS Press, 2000.

14. M. P. Marcus, B. Santorini, and M. A. Marcinkiewicz. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313-330, 1993.

15. L. Rau, P. Jacobs, and U. Zernik. Information extraction and text summarization using linguistic knowledge acquisition. *Information Processing & Management*, 25(4):419-428, 1989.

16. S. D. Richardson, W. B. Dolan, and L. Vanderwende. MIND-NET: Acquiring and structuring semantic information from text. In *COLING/ACL'98 – Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics & 17th International Conference on Computational Linguistics*, volume 2, pages 1098-1102. Montréal, Canada, August 10-14, 1998. San Francisco, CA: Morgan Kaufmann, 1998.

17. E. Riloff. An empirical study of automated dictionary construction for information extraction in three domains. *Artificial Intelligence*, 85(1/2):101-134, 1996.

18. M. Romacker and U. Hahn. An empirical assessment of semantic interpretation. In *NAACL 2000 – Proceedings of the 1st Meeting of the North American Chapter of the Association for Computational Linguistics*, pages 327-334. Seattle, Washington, USA, April 29 - May 4, 2000. San Francisco, CA: Morgan Kaufmann, 2000.

19. M. Romacker, K. Markert, and U. Hahn. Lean semantic interpretation. In *IJCAI'99 – Proceedings of the 16th International Joint Conference on Artificial Intelligence*, volume 2, pages 868-875. Stockholm, Sweden, July 31 - August 6, 1999. San Francisco, CA: Morgan Kaufmann, 1999.

20. B. Schiffman and K. R. McKeown. Experiments in automated lexicon building for text searching. In *COLING 2000 – Proceedings of the 18th International Conference on Computational Linguistics*, volume 2, pages 719-725. Saarbrücken, Germany, 31 July - 4 August, 2000. San Francisco, CA: Morgan Kaufmann, 2000.

21. K. Schnattinger and U. Hahn. Quality-based learning. In W. Wahlster, editor, *ECAI'98 – Proceedings of the 13th European Conference on Artificial Intelligence*, pages 160-164. Brighton, U.K., August 23-28, 1998. Chichester: John Wiley, 1998.

22. S. Soderland and W. Lehnert. WRAP-UP: A trainable discourse module for information extraction. *Journal of Artificial Intelligence Research*, 2:131-158, 1994.

23. A. Voutilainen. A syntax-based part-of-speech analyser. In *EACL'95 – Proceedings of the 7th Conference of the European Chapter of the Association for Computational Linguistics*, pages 157-164. Dublin, Ireland, March 27-31, 1995. Association for Computational Linguistics, 1995.

24. W. A. Woods and J. G. Schmolze. The KL-ONE family. *Computers & Mathematics with Applications*, 23(2/5):133-177, 1992.

CREAM — Creating relational metadata with a component-based, ontology-driven annotation framework

Siegfried Handschuh
Insitute AIFB
University of Karlsruhe
76128 Karlsruhe, Germany
sha@aifb.uni-karlsruhe.de

Steffen Staab
Insitute AIFB
University of Karlsruhe
76128 Karlsruhe, Germany
sst@aifb.uni-karlsruhe.de

Alexander Maedche
FZI Research Center for
Information Technologies
76131 Karlsruhe, Germany
maedche@fzi.de

"The Web is about links; the Semantic Web is about the relationships implicit in those links."
Dan Brickley

Abstract

Richly interlinked, machine-understandable data constitutes the basis for the Semantic Web. Annotating web documents is one of the major techniques for creating metadata on the Web. However, annotation tools so far are restricted in their capabilities of providing richly interlinked and truely machine-understandable data. They basically allow the user to annotate with plain text according to a template structure, such as Dublin Core. We here present CREAM (Creating RElational, Annotation-based Metadata), a framework for an annotation environment that allows to construct *relational metadata*, i.e. metadata that comprises class instances and relationship instances. These instances are not based on a fix structure, but on a domain ontology. We discuss some of the requirements one has to meet when developing such a framework, e.g. the integration of a metadata crawler, inference services, document management and information extraction, and describe its implementation, *viz.* Ont-O-Mat a component-based, ontology-driven annotation tool.

Keywords

Metadata, Markup, Annotations, Ontology, DAML+OIL, RDF, Semantic Web

Introduction

Research about the WWW currently strives to augment syntactic information already present in the Web by semantic metadata in order to achieve a Semantic Web that human and software agents alike can understand. RDF(S) or DAML+OIL are languages that have recently advanced the basis for extending purely syntactic information, *e.g.* HTML documents, with semantics. Based on these recent advancements one of the the most urgent challenges now is a knowledge capturing problem, *viz.* how one may turn existing syntactic resources into interlinked knowledge structures that represent relevant underlying information. This paper is about a framework for facing this challenge, called CREAM[1], and about its implementation, Ont-O-Mat.

The origin of our work facing this challenge dates back to the start of the seminal KA2 intiative [1], *i.e.* the initiative for providing semantic markup on HTML pages for the knowledge acquisition community. The basic idea then was that manual knowledge markup on web pages was too error-prone and should therefore be replaced by a *simple* tool that should help to avoid syntactic mistakes.

Developing our CREAM framework, however, we had to recognize that this knowledge capturing task exhibited some intrinsic difficulties that could not be solved by a *simple* tool. We here mention only some challenges that immediately came up in the KA2 setting:

- **Consistency:** Semantic structures should adhere to a given ontology in order to allow for better sharing of knowledge. For example, it should be avoided that people confuse complex instances with attribute types.
- **Proper Reference:** Identifiers of instances, *e.g.* of persons, institutes or companies, should be unique. For instance, in KA2 metadata there existed three different identifiers of our colleague Dieter Fensel. Thus, knowledge about him could not be grasped with a straightforward query.
- **Avoid Redundancy:** Decentralized knowledge provisioning should be possible. However, when annotators collaborate, it should be possible for them to identify (parts of) sources that have already been annotated and to reuse previously captured knowledge in order to avoid laborious redundant annotations.
- **Relational Metadata:** Like HTML information, which is spread on the Web, but related by HTML links, knowledge markup may be distributed, but it should be semantically related. Current annotation tools tend to generate template-like metadata, which is hardly connected, if at all. For example, annotation environments often support Dublin Core [12], providing means to state, e.g., the name of authors, but not their IDs[3].

---

[1]CREAM: Creating RElational, Annotation-based Metadata.
[2]The reader may see similar effects in bibliography databases. E.g., query for James (Jim) Hendler at the — otherwise excellent — DBLP:
http://www.informatik.uni-trier.de/~ley/db/.
[3]In the web context one typically uses the term 'URI' (uniform resource,

- **Maintenance**: Knowledge markup needs to be maintained. An annotation tool should support the maintenance task.
- **Ease of use**: It is obvious for an annotation environments to be useful. However, it is not trivial, because it involves intricate navigation of semantic structures.
- **Efficiency**: The effort for the production of metadata is a large restraining threshold. The more efficiently a tool support the annotation, the more metadata will produce a user. These requirement stand in relationship with the ease of use. It depends also on the automation of the annotation process, e.g. on the pre-processing of the document.

CREAM faces these principal problems by combining advanced mechanisms for inferencing, fact crawling, document management and — in the future — information extraction. Ont-O-Mat, the implementation of CREAM, is a component-based plug-in architecture that tackles this broad set of requirements.[4]

In the following we first sketch two usage scenarios (Section: Scenarios for CREAM). Then, we explain our terminology in more detail, derive requirements from our principal considerations above and explain the architecture of CREAM (Section: Design of CREAM). We describe our actual tool, Ont-O-Mat, in Section Implementation. Before we conclude, we contrast CREAM with related work, namely knowledge acquisition tools and annotation frameworks.

Scenarios for CREAM

We here only summarize two scenarios, two knowledge portals, for annotation that have been elaborated in [21]:

The first scenario extends the objectives of the seminal KA2 initiative. The KA2 portal provides a view onto knowledge of the knowledge acquisition community. Besides of semantic retrieval as provided by the original KA2 initiative, it allows comprehensive means for navigating and querying the knowledge base and also includes guidelines for building such a knowledge portal. The potential users provide knowledge, e.g. by annotating their web pages in a decentralized manner. The knowledge is collected at the portal by crawling and presented in a variety of ways.

The second scenario is a knowledge portal for business analysts that is currently constructed at Ontoprise GmbH. The principal idea is that business analyst review news tickers, business plans and business reports. A considerable part of their work requires the comparison and aggregation of similar or related data, which may be done by semantic queries like"Which companies provide B2B solutions?", when the knowledge is semantically available. At the Time2Research portal they will handle different types of documents, annotate them and, thus, feed back into the portal to which they may ask questions.

Design of CREAM

In this section we explain basic design decisions of CREAM, which are founded on the general problems sketched in the

introduction above. In order to provide a clear design rationale, we first provide definitions of important terms we use subsequently:

- **Ontology**: An ontology is a formal, explicit specification of a shared conceptualization of a domain of interest [8]. In our case it is constituted by statements expressing definitions of DAML+OIL classes and properties [7].
- **Annotations**: An annotation in our context is a set of instantiations attached to an HTML document. We distinguish *(i)* instantiations of DAML+OIL classes, *(ii)* instantiated properties from one class instance to a datatype instance — henceforth called attribute instance (of the class instance), and *(iii)* instantiated properties from one class instance to another class instance — henceforth called relationship instance.
  Class instances have unique URIs. Instantiations may be attached to particular markups in the HTML documents, *viz.* URIs and attribute values may appear as strings in the HTML text.
- **Metadata**: Metadata are data about data. In our context the annotations are metadata about the HTML documents.
- **Relational Metadata**: We use the term relational metadata to denote the annotations that contain relationship instances.
  Often, the term "annotation" is used to mean something like "private or shared note", "comment" or "Dublin Core metadata". This alternative meaning of annotation may be emulated in our approach by modeling these notes with attribute instances. For instance, a comment note "I like this paper" would be related to the URL of the paper via an attribute instance 'hasComment'.
  In contrast, relational metadata contain statements like 'student Siegfried cooperates with lecturer Steffen', *i.e.* relational metadata contain relationships between class instances rather than only textual notes.
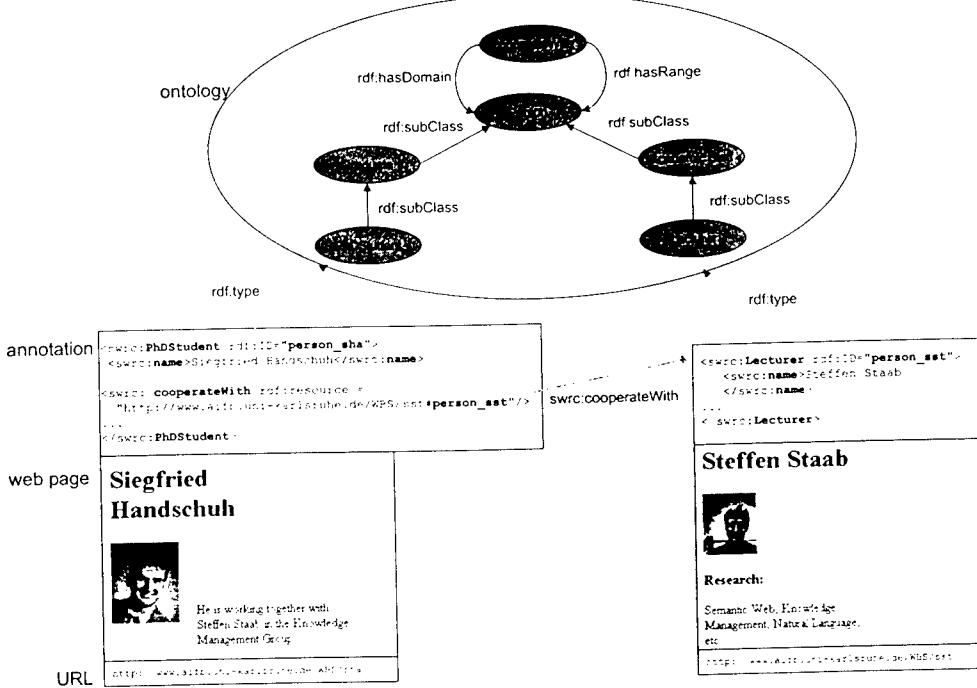
Figure 1 illustrates our use of the terms "ontology", "annotation" and "relational metadata". It depicts some part of the SWRC[5] (semantic web research community) ontology. Furthermore it shows two homepages, viz. pages about Siegfried and Steffen (http://www.aifb.uni-karlsruhe.de/WBS/sha and http://www.aifb.uni-karlsruhe.de/WBS/sst, respectively) with annotations given in an XML serialization of RDF facts. For the two persons there are instances denoted by corresponding URIs (person_sha and person_sst). The swrc:name of person_sha is "Siegfried Handschuh". In Addition, there is a relationship instance between the two persons: they cooperate. This cooperation information 'spans' the two pages.

Requirements for CREAM

The difficulties sketched in the introduction directly feed into the design rationale of CREAM. The design rationale links the challenges with the requirements. This results in a N:M mapping (neither functional nor injective). An overview of the matrix is given in Table 1. It shows which modules (requirements) are mainly used to answer challenges set forth in the introduction, *viz.*:

---

identifier) to speak of 'unique identifier'.

[4] The core Ont-O-Mat can be downloaded from:
http://ontobroker.semanticweb.org/annotation.

[5] http://ontobroker.semanticweb.org/ontos/swrc.html

Figure 1: Annotation example

- **Document Viewer**: The document viewer visualizes the web page contents. The annotator may easily provide annotations by highlighting text that serves as input for attribute instances or the definition of URIs. The document viewer must support various formats (HTML, PDF, XML, etc.).
- **Ontology Guidance**: The annotation framework needs guidance from the ontology. In order to allow for sharing of knowledge, newly created annotations must be consistent with a community's ontology. If annotators instantiate arbitrary classes and properties the semantics of these properties remains void. Of course the framework must be able to adapt to varying ontologies in order to reflect different foci of the annotators.
  Furthermore, the ontology is important in order to guide annotators towards creating relational metadata. We have done some preliminary experiments and found that subjects have more problems with creating relationship instances than with creating attribute instances (cf. [22]). Without the ontology they would miss even more cues for assigning relationships between class instances.
  Both ontology guidance and document viewer should be easy to use: Drag'n'drop helps to avoid syntax errors and typos and a good visualization of the ontology can help to correctly choose the most appropriate class for instances.
- **Crawler**: The creation of relational metadata must take place *within* the Semantic Web. During annotation annotaters must be aware of which entities exist in the part of the Semantic Web they annotate. This is only possible if a crawler makes relevant entities immediately available. So, annotators may look for proper reference, i.e. decide whether an entity already has a URI (e.g. whether the entity named "Dieter Fensel" or "D. Fensel" has already been identified by some other annotators) and thus only annotators may recognize whether properties have already been instantiated (e.g. whether "Dieter Fensel" has already be linked to his publications). As a consequence of annotators' awareness relational metadata may be created, because class instances become related rather than only flat templates are filled.
- **Annotation Inference Server**: Relational metadata, proper reference and avoidance of redundant annotation require querying for instances, i.e. querying whether and which instances exist. For this purpose as well as for checking of consistency, we provide an annotation inference server in our framework. The annotation inference server reasons on crawled and newly annotated instances and on the ontology. It also serves the ontological guidance, because it allows to query for existing classes and properties.
- **Document Management**: In order to avoid redundancy of annotation efforts, it is not sufficient to ask whether instances exist at the annotation inference server. When an annotator decides to capture knowledge from a web page, he does not want to query for all single instances that he considers relevant on this page, but he wants information, whether and how this web page has been annotated before. Considering the dynamics of HTML pages on the web, it is desirable to store annotated web pages together with their

78

Table 1: Design Rationale — Linking Challenges with Required Modules

Requirement General Problem	Document Viewer	Ontology Guidance	Storage			
			Replication		Document Management	Information Extraction
			Crawler	Annotation Inference Server		
Consistency		X		X		
Proper Reference			X	X		
Avoid Redundancy			X	X	X	
Relational Metadata		X	X	X		
Maintenance					X	X
Ease of use	X	· X				X
Efficiency	X	X	X	X	X	X

annotations. When the web page changes, the old annotations may still be valid or they may become invalid. The annotator must decide based on the old annotations and based on the changes of the web page.

A future goal of the document management in our framework will be the semi-automatic maintenance of annotations. When only few parts of a document change, pattern matching may propose revision of old annotations.

- **Information Extraction**: Even with sophisticated tools it is laborious to provide semantic annotations. A major goal thus is semi-automatic annotation taking advantage of information extraction techniques to propose annotations to annotators and, thus, to facilitate the annotation task. Concerning our environment we envisage two major techniques: First, "wrappers" may be learned from given markup in order to automatically annotate similarly structured pages (cf., e.g., [16]). Second, message extraction like systems may be used to recognize named entities, propose co-reference, and extract some relationship from texts (cf., e.g., [20]).

Besides of the requirements that constitute single modules, one may identify functions that cross module boundaries:

- **Storage**: CREAM supports two different ways of storage. The annotations will be stored inside the document that is in the document management component, but it is also stored in the annotation inference server.
- **Replication**: We provide a simple replication mechanism by crawling annotations into our annotation inference server.

Architecture of CREAM

The architecture of CREAM is depicted in Figure 2. The complete design of CREAM comprises a plug-in structure, which is flexible with regard to adding or replacing modules. Document viewer and ontology guidance module together constitute the major part of the graphical user interface. Via plug-ins the core annotation tool, Ont-O-Mat, is extended to include the capabilities outlined above. For instance, a plug-in for a connection to a document management system provides document management and retrieval capabilities that show the user annotations of a document he loads into his browser. This feature even becomes active when the user does not actively search for already existing

annotations. Similarly, Ont-O-Mat provides extremely simple means for navigating the taxonomy, which means that the user can work without an inference server. However, he only gets the full-fledged semantics when the corresponding plug-in connection to the annotation inference server is installed.

Implementation: Ont-O-Mat

This section describes Ont-O-Mat, the implementation of our CREAM framework. Ont-O-Mat is a component-based, ontology-driven markup tool. The architectural idea behind CREAM is a component-based framework, thus, being open, flexible and easily extensible.

In the following subsection we refer to the concrete realization and the particular technical requirements of the components. In subsection we describe the functionality of Ont-O-Mat based on an example ontology for annotation that is freely available on the web.
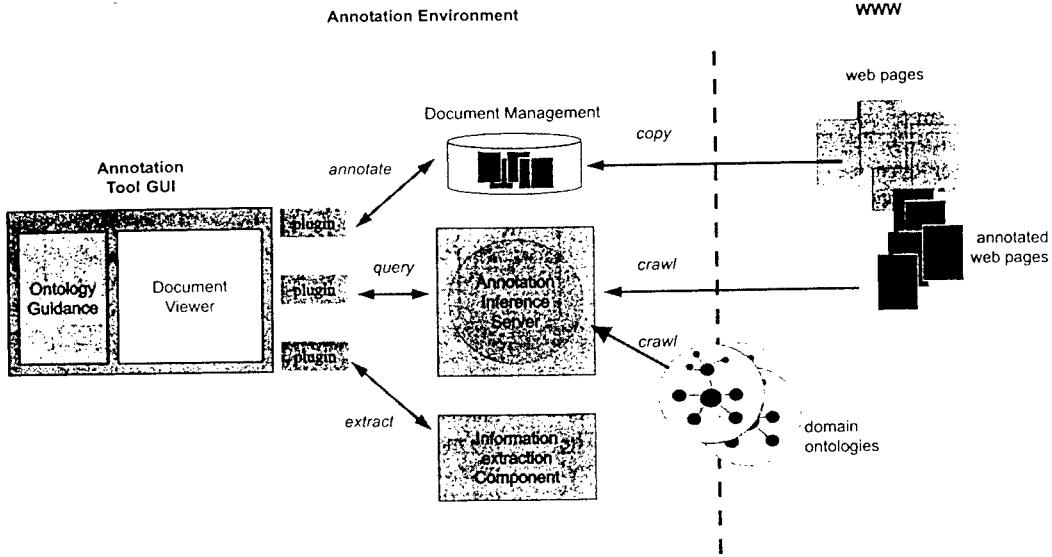
Ont-O-Mat services and components

The architecture of Ont-O-Mat provides a plug-in and service mechanism. The components are dynamically plug-able to the core Ont-O-Mat. The plug-in mechanism notifies each installed component, when a new component is registered. Through the service mechanism each component can discover and utilize the services offered by another component [9]. A service represented by a component is typically a reference to an interface. This provides among other things a de-coupling of the service from the implementation and allows therefore alternative implementations.

The Ont-O-Mat services have been realized by components according to the requirements listed in subsection . So far we have realized the following components: a comprehensive user-interface, component for document-management, an annotation inference-server and a crawler:

- **Document Viewer and Ontology Guidance**: There are various ways how the gained knowledge database can be visualized and thus experienced. On the one hand, the system can be used as a browser. In the annotated web pages, the extracted text fragments are then highlighted and an icon after each fragment is visible. By clicking on the icon, the name of the assigned class or attribute will be shown.

Figure 2: Architecture of CREAM.

Figure 2: Architecture of CREAM.

On the other hand, the user can browse the ontology and retrieve for one class all instances or for one instance all attributes.

The underlying data model used for Ont-O-Mat has been taken from the comprehensive ontology engineering and learning system ONTOEDIT / TEXT-TO-ONTO (see [18]). Ont-O-Mat works currently in "read-only-mode" with respect to the ontology and only operates on the relational metadata defined on top of the given ontology.

- **Document Management**: A component for document management is required in order to avoid duplicate annotations and existing semantic annotations of documents should be recognized. In our current implementation we use a straight forward file-system based document management approach. Ont-O-Mat uses the URI to detect the re-encounter of previously annotated documents and highlights annotations in the old document for the user. Then the user may decide to ignore or even delete the old annotations and create new metadata, he may augment existing data, or he may just be satisfied with what has been previously annotated. In order to recognize that a document has been annotated before, but now appears under a different URI, Ont-O-Mat computes similarity with existing documents by simple information retrieval methods, e.g. comparison of the word vector of a page. If thereby a similarity is discovered, this is indicated to the user, so that he can check for congruency.
- **Annotation Inference Server**: The annotation inference server reasons on crawled and newly annotated instances and on the ontology. It also serves the ontological guidance, because it allows to query for existing classes and properties. We use Ontobroker's [3] underlying F-Logic [14] based inference engine SilRI [2] as annotation inference server. The F-Logic inference engine combines ordering-

independent reasoning in a high-level logical language with a well-founded semantics.

- **RDF Crawler**: As already mentioned above, the annotation must take place right within the Semantic Web and not isolated. Therefore, we have built a RDF Crawler[6], a basic tool that gathers interconnected fragments of RDF from the Web and builds a local knowledge base from this data. In general, RDF data may appear in Web documents in several ways. We distinguish between *(i)* pure RDF (files that have an extension like "*.rdf"), *(ii)* RDF embedded in HTML and *(iii)* RDF embedded in XML. Our RDF Crawler relys on Melnik's RDF-API[7] that can deal with the different embeddings of RDF described above. One general problem of crawling is the applied filtering mechanism: Baseline document crawlers are typically restricted by a predefined depth value. Assuming that there is an unlimited amount of interrelated information on the Web (hopefully this will soon hold about RDF data as well), at some point RDF fact gathering by the RDF Crawler should stop. We have implemented a baseline approach for filtering: At the very start of the crawling process and at every subsequent step we maintain a queue of all the URIs we want to analyze. We process them in the breadth-first-search fashion, keeping track of those we have already visited. When the search goes too deep, or we have received sufficient quantity of data (measured as number of links visited or the total web traffic or the amount of RDF data obtained) we may quit.
- **Information Extraction**: This component has not yet been integrated in our Ont-O-Mat tool. Actually, we are near finishing an integration of a simple wrapper approach [15],

---

[6]RDF Crawler is freely available for download at:
http://ontobroker.semanticweb.org/rdfcrawler.
[7]http://www-db.stanford.edu/~melnik/rdf/api.html

80

but we have not yet the message extraction approach for Ont-O-Mat that suggests relevant part of the texts for annotation.

Using Ont-O-Mat — An Example

Our example is based on the freely available SWRC (Semantic Web Research Community)[8] ontology , the successor of the KA2 ontology. The SWRC ontology models the semantic web research community, its researchers, topics, publications, tools, etc. and properties between them. It is available in the form of DAML+OIL classes and properties, in pure RDF-Schema and in F-Logic. The general idea behind SWRC is that the SW research community creates relational metadata according to the SWRC ontology to enable semantic access to their web pages. In the following we shortly explain how Ont-O-Mat may be used for creating relational metadata based on the SWRC ontology.

The annotation process is started either with an annotation inference server or the server process is fed with metadata crawled from the web and the document server. Figure 3 shows the screen for navigating the ontology and creating annotations in Ont-O-Mat. The right pane displays the document and the left panes show the ontological structures contained in the ontology, namely classes, attributes and relations. In addition, the left pane shows the current semantic annotation knowledge base, i.e. existing class instances, attribute instances and relationship instances created during the semantic annotation.

1. First of all, the user browses a document by entering the URL of the web document that he would like to annotate. This step is quite familiar from existing browsers.

2. Then the user selects a text fragment by highlighting it and takes a look on the ontology which fits in the topic and is therefore loaded and visible in ontology browser.

3. There are two possibilities for the text fragment to be annotated: as an instance or as a property. In the case of an instance, the user selects in the ontology the class where the text fragment fits in, e.g. if he has the text fragment "Siegfried Handschuh", he would select the class "PhD Student". By clicking on the class, the annotation gets created and thus the text fragment will be shown as an instance of the selected class in the ontology at the ontology browser.

4. To each created instance, literal attributes can be assigned. The choice of the predefined attributes depends on the class the instance belongs to, e.g. the class "PhD Student" has the attributes name, address, email, and telephone number. The attributes can be assigned to the instance by highlighting the appropriate text fragment of the web document and dragging it to the related property field.

5. Furthermore, the relationships between the created instances can be set, e.g. the PhD Student Siegfried Handschuh

"works at" the OntoAgent project and "is supervised" by Rudi Studer. Ont-O-Mat preselects class instances according to the range restrictions of the chosen relation, e.g. the "works at" of a PhD Student must be an Project. Therefore only Projects are offered as potential fillers to the "works at" relation of Siegfried.

Comparison with Related Work

CREAM can be compared along three dimensions: First, it is a framework for mark-up in the Semantic Web. Second, it can be considered as a particular knowledge acquisition framework vaguely similar like Protégé-2000[6]. Third, it is certainly an annotation framework, though with a different focus than ones like Annotea [13].

Knowledge Markup in the Semantic Web

We know of three major systems that intensively use knowledge markup in the Semantic Web, viz. SHOE [10], Ontobroker [3] and WebKB [19]. All three of them rely on knowledge in HTML pages.

They all started with providing manual mark-up by editors. However, our experiences (cf. [5]) have shown that text-editing knowledge mark-up yields extremely poor results, viz. syntactic mistakes, improper references, and all the problems sketched in the introduction.

The approaches from this line of research that are closest to CREAM is the *SHOE Knowledge Annotator*[9].

The SHOE Knowledge Annotator is a Java program that allows users to mark-up webpages with the SHOE ontology. The SHOE system [17] defines additional tags that can be embedded in the body of HTML pages. The Knowledge Annotater is less user friendly compared with our implementation Ont-O-Mat. It shows the ontology in some textual lists, whereas Ont-O-Mat gives a graphical visualization of the ontologies. Furthermore, in SHOE there is no direct relationship between the new tags and the original text of the page, i.e. SHOE tags are not annotations in a strict sense.

Comparison with Knowledge Acquisition Frameworks

The CREAM framework is specialized for creating class and property instances and for populating HTML pages with them. Thus, it does not function as an ontology editor, but rather like the instance acquisition phase in the Protégé-2000 framework [6]. The obvious difference of CREAM to the latter is that Protege does not (and does not intend to) support the particular web setting, *viz.* managing and displaying web pages.

Comparison with Annotation Frameworks
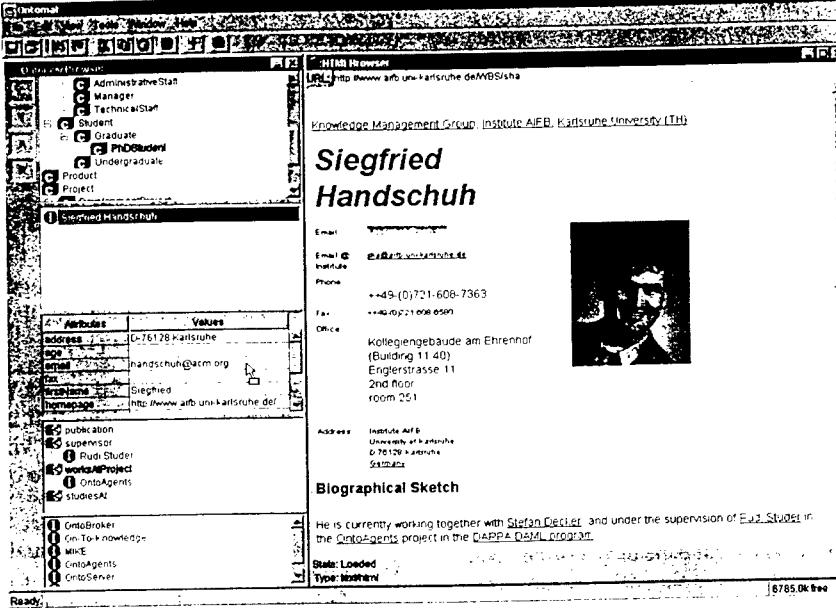
There are a lot of — even commercial — annotation tools like ThirdVoice[10], Yawas [4], CritLink [23] and Annotea (Amaya) [13].

These tools all share the idea of creating a kind of user comment on the web pages. The term "annotation" in these frame-

Figure 3: Ont-O-Mat Screenshot.

works is understood as a remark to an existing document. As mentioned before, we would model such remarks as attribute instances only in our framework. For instance, a user of these tools might attach a note like "A really nice professor!" to the name "Studer" on a web page.

Annotea actually goes one step further. It allows to rely on an RDF schema as a kind of template that is filled by the annotator. For instance, Annotea users may use a schema for Dublin Core and fill the author-slot of a particular document with a name. This annotation, however, is again restricted to attribute instances. The user may also decide to use complex RDF descriptions instead of simple strings for filling such a template. However, he then has no further support from Amaya that helps him providing syntactically correct statements with proper references.

To summarize, CREAM is used to generate really machine-understandable data and addresses all the problems that come from this objective: relational metadata, proper reference and consistency.

Conclusion and Future Plans

CREAM is a comprehensive framework for creating annotations, relational metadata in particular — the foundation of the future Semantic Web. The framework comprises inference services, crawler, document management system, ontology guidance, and document viewers.

Ont-O-Mat is the reference implementation of CREAM framework. The implementation supports so far the user with the task of creating and maintaining ontology-based DAML+OIL markups, i.e. creating of class, attribute and relationship in-

stances. Ont-O-Mat include an ontology browser for the exploration of the ontology and instances and a HTML browser that will display the annotated parts of the text. Ont-O-Mat is Java-based and provide a plugin interface for extensions for further advancement.

Our goal is a constant advancement of Ont-O-Mat and the CREAM framework in order to answer basic problems that come with semantic annotation.

We are already dealing with many different issues and through our practical experiences we could identify problems that are most relevant in our scenario/settings, KA2 and Time2Research. Nevertheless our analysis of the general problem is far from being complete. Some further important issues we want to mention here are:

- **Information Extraction**: We have done some first steps to incorporate information extraction. However, our future experiences will have to show how and how well information extraction integrates with semantic annotation.
- **Multimedia Annotation**: This requires considerations about time, space and synchronization.
- **Annotation + Authoring**: Knowledge capturing and annotation is a process of marking up existing HTML with semantic data. We are also interested in supporting the inverse process of HTML authoring from semantic data.
- **Changing Ontologies**: Ontologies on the web have characteristics that influence the annotation process. Heflin & Hendler [11] have elaborated on changes that affect annotation. Future annotation tools will have to incorporate solutions for the difficulties they consider.
- **Active Ontology Evolvement**: Annotation should feed back

82

into the actual ontologies, because annotators may find that they should consider new knowledge, but need revised ontologies for this purpose. Thus, annotation affects ontology engineering and ontology learning.

Our general conclusion is that providing semantic annotation, relational metadata in particular, is an important complex task that needs comprehensive support. Our framework CREAM and our tool Ont-O-Mat have already proved very successful in leveraging the annotation process. They still need further refinement, but they are unique in their design and implementation.

REFERENCES

1. R. Benjamins, D. Fensel, and S. Decker. KA2: Building Ontologies for the Internet: A Midterm Report. *International Journal of Human Computer Studies*, 51(3):687, 1999.

2. S. Decker, D. Brickley, J. Saarela, and J. Angele. A Query and Inference Service for RDF. In *Proceedings of the W3C Query Language Workshop (QL-98), http://www.w3.org/TandS/QL/QL98/, Boston, MA, December 3-4*, 1998.

3. S. Decker, M. Erdmann, D. Fensel, and R. Studer. Ontobroker: Ontology Based Access to Distributed and Semi-Structured Information. In R. Meersman et al., editors, *Database Semantics: Semantic Issues in Multimedia Systems*, pages 351–369. Kluwer Academic Publisher, 1999.

4. L. Denoue and L. Vignollet. An annotation tool for web browsers and its applications to information retrieval. In *In Proceedings of RIAO2000*, Paris, April 2000. http://www.univ-savoie.fr/ labos/syscom/Laurent.Denoue/riao2000.doc.

5. M. Erdmann, A. Maedche, H.-P. Schnurr, and Steffen Staab. From manual to semi-automatic semantic annotation: About ontology-based text annotation tools. In *P. Buitelaar & K. Hasida (eds). Proceedings of the COLING 2000 Workshop on Semantic Annotation and Intelligent Content*, Luxembourg, August 2000.

6. H. Eriksson, R. Fergerson, Y. Shahar, and M. Musen. Automatic generation of ontology editors. In *Proceedings of the 12th Banff Knowledge Acquisition Workshop, Banff, Alberta, Canada*, 1999.

7. Reference description of the daml+oil (march 2001) ontology markup language, March 2001. http://www.daml.org/2001/03/reference.html.

8. T. R. Gruber. A Translation Approach to Portable Ontology Specifications. *Knowledge Acquisition*, 6(2):199–221, 1993.

9. Siegfried Handschuh. Ontoplugins – a flexible component framework. Technical report, University of Karlsruhe, May 2001.

10. J. Heflin and J. Hendler. Searching the web with shoe. In *Artificial Intelligence for Web Search. Papers from the AAAI Workshop. WS-00-01*, pages 35–40. AAAI Press, 2000.

11. J. Heflin, J. Hendler, and S. Luke. Applying Ontology to the Web: A Case Study. In *Proceedings of the International Work-Conference on Artificial and Natural Neural Networks, IWANN'99*, 1999.

12. Dublin Core Metadata Initiative. http://purl.oclc.org/dc/, April 2001.

13. J. Kahan, M. Koivunen, E. Prud'Hommeaux, and R. Swick. Annotea: An Open RDF Infrastructure for Shared Web Annotations. In *Proc. of the WWW10 International Conference. Hong Kong*, 2001.

14. M. Kifer, G. Lausen, and J. Wu. Logical foundations of object-oriented and frame-based languages. *Journal of the ACM*, 42, 1995.

15. J. Klotzbuecher. Ontowrapper. Master's thesis, University of Karlsruhe, to appear 2001.

16. N. Kushmerick. Wrapper Induction: Efficiency and Expressiveness. *Artificial Intelligence*, 118(1), 2000.

17. S. Luke, L. Spector, D. Rager, and J. Hendler. Ontology-based Web Agents. In *Proceedings of First International Conference on Autonomous Agents*, 1997.

18. A. Maedche and S. Staab. Ontology learning for the semantic web. *IEEE Intelligent Systems*, 16(2), 2001.

19. P. Martin and P. Eklund. Embedding Knowledge in Web Documents. In *Proceedings of the 8th Int. World Wide Web Conf. (WWW'8), Toronto, May 1999*, pages 1403–1419. Elsevier Science B.V., 1999.

20. *MUC-7 — Proceedings of the 7th Message Understanding Conference*. http://www.muc.saic.com/, 1998.

21. S. Staab and A. Maedche. Knowledge portals — ontologies at work. *AI Magazine*, 21(2), Summer 2001.

22. S. Staab, A. Maedche, and S. Handschuh. Creating metadata for the semantic web: An annotation framework and the human factor. Technical Report 412, Institute AIFB, University of Karlsruhe, 2001.

23. Ka-Ping Yee. CritLink: Better Hyperlinks for the WWW, 1998. http://crit.org/ ping/ht98.html.

Capturing Analytic Thought

John D. Lowrance
Ian W. Harrison
Andres C. Rodriguez
Artificial Intelligence Center
SRI International
333 Ravenswood Avenue, Menlo Park, CA 94404
{lowrance, harrison, rodriguez} @ai.sri.com

Abstract

The survival of an enterprise often rests upon its ability to make correct and timely decisions, despite the complexity and uncertainty of the environment. Because of the difficulty of employing and scaling formal methods in this context, decision makers typically resort to informal methods, sacrificing structure and rigor. We are developing a new methodology that retains the ease of use, the familiarity, and (some of) the free-form nature of informal methods, while benefiting from the rigor, structure, and potential for automation characteristic of formal methods. Our approach records analysts' thinking in a corporate knowledge base consisting of *structured arguments*. The foundation of this knowledge base is an ontology of arguments that includes two main types of formal objects: *argument templates* and *arguments*. An argument template records an analytic method as a hierarchically structured set of interrelated questions, and an argument instantiates an argument template by answering the questions posed relative to a specific situation. This methodology emphasizes the use of simple inference structures as the foundation of its argument templates, making it possible for analysts to independently author new templates. When authoring an argument template, the analyst can choose to embed *discovery tools*, which are recommended methods of acquiring information pertaining to the questions posed. An analyst wanting to record an argument selects an appropriate template, uses the discovery tools to retrieve potentially relevant information, selects that information to retain as *evidence* and records its *relevance*, *answers* the questions, and records the *rationale* for the answers. The result is a recorded line of reasoning that breaks down the problem, bottoming out at the documents and other forms of information that were used as evidence to support the answers. The resulting collection of arguments and

templates constitutes a corporate memory of analytic thought that can be directly exploited by analysts or automated methods.

Keywords

Structured arguments, evidential reasoning, analysis, knowledge management.

INTRODUCTION

Understanding the world and facing the different alternatives it presents to us is crucial in any effort. Different studies and formalisms of *argumentation* have come out of different fields such as philosophy [11, 14, 15, 19] decision analysis [17] and artificial intelligence [9, 16, 4]. These formalisms attempt to deal with the uncertainty inherently present in the world. Behind every decision, though, there is an argument supporting it, and arguments range from rhetorical explanations to mathematical proofs. Argumentation theory leverages problem solving under uncertainty by supporting qualitative and quantitative approaches.

Analysis, on the other hand, deals with the examination and separation of a complex situation, its elements, and its relationships. More often than not, the situation is full of unknowns, uncertainties, and deliberate misinformation. The analyst is confronted not only with the facts, but also with his or her knowledge about the facts and assumptions, others' possible knowledge, the hypotheses that can be drawn from those facts, and the evidence supporting and contradicting those hypotheses (Heuer 1999).
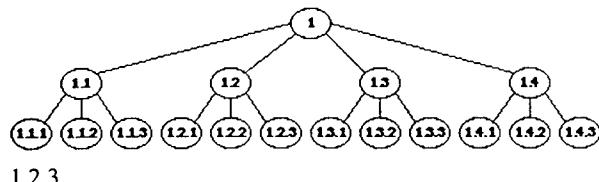
Under the sponsorship of the Defense Advanced Research Projects Agency (DARPA) of the U.S. Department of Defense, SRI International is developing SEAS, the Structured Evidential Argumentation System also known as the SRI Early Alert System (Lowrance, Harrison, and Rodriguez 2000). This work builds upon an earlier effort (Stokke et al. 1994) that developed the first SEAS prototype applied to the problem of early warning for project management. In our current work, SEAS is being generalized and applied to the problem of crisis warning for national security. Our goal is to construct a system capable of aiding intelligence analysts in leveraging analytic products and methods developed for past situations or by

other analysts addressing the same or similar contemporary problems. These analytic products take the form of arguments: given a framework of assumptions, some conclusions or statements can be reached. While national security analysis is the focus of this work, we believe that the tools and methods being developed have broad application outside of the national security arena. We believe that these tools and methods can be effectively applied to any problem where regular assessments must be made, based upon evidence from multiple sources, within a complex and uncertain environment.

CAPTURING ANALYTIC METHODS

Our approach is based on the concept of a *structured argument*. A structured argument is based on a hierarchically organized set of questions (a tree) that is used to assess whether an opportunity or threat of a given type is imminent. This hierarchy of questions is called the *argument's template* (as opposed to the *argument*, which is an instantiation of the template). This hierarchy of questions supporting questions may go a few levels deep before bottoming out in questions that must be directly assessed and answered. These are multiple-choice questions, with the different answers corresponding to discrete points or subintervals along a continuous scale, with one end of the scale representing strong support for a particular type of opportunity or threat and the other end representing strong refutation. Leaf nodes represent primitive questions, and internal nodes represent derivative questions. The links represent support relationships among the questions. A derivative question is supported by all the derivative and primitive questions below it. Figure 1 illustrates a seventeen-question argument template, with twelve primitive questions and five derivative questions. Note that question 1 is answered based upon the answers to 1.1, 1.2, 1.3, and 1.4, and 1.2 is answered based upon the answers to 1.2.1, 1.2.2, and

Figure 1: An example argument skeleton

1.2.3.

An inference method completes an argument template. It is used to automatically answer some questions based upon the answers to other questions. The analyst answers the primitive questions in the question hierarchy, and the answers to the derivative questions are automatically calculated. In so doing, our approach emphasizes the use of simple and regular inference structures. These structures are captured by *argument skeletons* and associated *inference methods*. The same argument skeleton and inference methods are typically used to support multiple argument templates over widely differing

topics. A typical inference method might take the maximum answer as the conclusion when combining several questions assessed along a continuous scale. The idea is that if the argument template author fully understands the structure of the interrelated questions that constitute the argument skeleton and the propagation scheme implemented by the inference method, then the author can write the argument template questions and answers to fit. The simpler the argument skeletons and inference methods, the easier it is for the author.

The use of regular argument skeletons is encouraged – that is, skeletal trees where all branches are identically structured. Regular structures help to encourage that equal time and emphasis are placed on all aspects of an analysis. Likewise, the use of uniform or regular inference methods is encouraged. A uniform inference method, where every derivative question's answer is derived using the same fusion method, makes for the easiest arguments to understand and lines of reasoning to follow. A regular inference method, one that employs the same fusion method across all questions at the same depth in the skeletal tree, is the next easiest to understand and follow.

Our philosophy is directly opposed to that of most uncertain reasoning systems. In most systems, the author begins by determining what questions might be asked and then interrelates them through a complex set of interconnections, typically annotated with conditional probabilities. As a result, the updating scheme is often complex and difficult to follow for those not versed in probability theory. While this "strong model" approach can be very effective when properly applied, we believe that the "weak model" approach emphasized here is easier to understand and use. Its effectiveness is directly related to the author's ability to adapt to these simple and regular inference structures, writing questions and answers that properly function within these constraints. Thus, knowledge is entered via text editing, without the use of probabilities or weights, making knowledge entry easy.

The challenge in authoring an argument template is to break the problem down into a hierarchically structured set of questions that matches the selected argument skeleton and whose interrelationships among the answers follow the inference method. Therefore, it is critical that the author understands the structure of the argument skeleton and the effect of the inference method, before beginning to fashion the questions and answers that will be posed by the argument template. See Figure 2 for an example argument template question hierarchy.

Each derivative question is represented by two text strings: a *topic* and the *question* itself. Primitive questions also include a question *amplification* string and five *multiple-choice answers*. The amplification states the question in more detail, reminding the user of the range of things to consider when answering the question.

To facilitate the rapid comprehension of arguments, we use a traffic light metaphor; relating answers to colored lights along a linear scale, from green to red. The questions in a template are yes/no or true/false; the multiple-choice answers for primitive questions partition this range, associating an answer with each colored light. Typically, a five-light scale is used (green, yellow-green, yellow, orange, red). Here green might correspond to true,

red to false, and the other three to varying degrees of certainty. Ideally, the multiple-choice answers are as concrete as possible and directly and unambiguously observable, making it easier for the user to recognize the answer that fits the situation being analyzed. No multiple-choice answers are associated with derivative questions; within arguments, their answers are strictly summarized by lights indicating their degree of certainty.

There are two distinct ways of approaching the structuring of an argument template: top-down and bottom-up. Using the top-down approach, one starts with the central question and attempts to break it down into a small set of supporting questions, each of approximately the same significance; then one breaks down each of those questions, attempting to break each into the same number of equally significant questions. This procedure continues until questions are produced that can be directly answered or until the number of overall questions has become too numerous to include in a single template. In this latter case, the author might elect to limit the depth of the original template and then capture those elements that fell below that depth limit in their own templates; each of these *cascaded templates* would share its root question with one of the primitive questions in the original template. The relationship of these cascaded templates to the original template can be captured by adding these to the original template as *discovery tools* (more on this below). As such, an analyst who is developing an argument based upon the original template, and is confronted with one of its primitive questions, can either elect to directly answer the stated question or invoke one of these discovery tools to further break down the question. The advantage of this approach is that the analyst determines which of these discovery tools to employ, thus choosing where and where not to spend time.

Using the bottom-up approach, one starts by enumerating the detailed conditions that should lead to warning. Once these are enumerated, one begins to cluster these into coherent collections of roughly equal size and significance. One then clusters the clusters, again striving for clusters of equal size and significance, and continues this process until a single cluster remains. Each cluster should give rise to a question in the resulting template, with the nesting of the clusters captured as supporting questions.

In practice, neither the top-down nor bottom-up approach is employed in its pure form. Instead, both are typically employed at different times, one after the other, until a satisfactory result is achieved. Once the overall skeletal structure has been established, then the author's attention should turn to writing the detailed questions and answers for the template.

In general, *discovery tools* are recommended methods for acquiring information relevant to answering questions in

---

Figure 2: An example argument template question hierarchy

1. **POLITICAL**: Is this country headed for a political crisis?

  1.1. **POLITICAL INSTABILITY**: Is political instability increasing?

    1.1.1 **INCREASINGLY UNSTABLE/WEAK GOVERNMENT**: Is the government becoming increasingly unstable or weak?

    1.1.2 **INCREASING CONFLICT OVER POLICY/ISSUE AREA**: Is increasing conflict over policy/issue areas having a destabilizing effect?

    1.1.3 **DECREASING PUBLIC CONFIDENCE**: Is decreasing public confidence in the leadership or government policies having a destabilizing effect?

  1.2. **POWER STRUGGLE**: Is there a government power struggle with potentially destabilizing consequences?

    1.2.1. **FACTIONALISM**: Is there evidence of growing factionalism within the government, bureaucracy, or legislature that is leading to or exacerbating a power struggle?

    1.2.2. **OPPOSITION CHALLENGE**: Is there a significant political opposition challenge to the government that is leading to or exacerbating a power struggle?

    1.2.3. **SUBNATIONAL GROUP INFLUENCE**: Are powerful subnational groups contributing to a government power struggle by influencing or backing specific government officials/factions?

  1.3. **GOVERNMENT RESPONSE TO SOCIO-POLITICAL DISCORD**: Is the government resorting to increasingly stringent measures in response to socio-political discord with potentially destabilizing consequences?

    1.3.1. **REPRESSION OF POLITICAL OPPOSITION**: Is government repression of the political opposition or dissident groups occurring/increasing?

    1.3.1 **REPRESSION OF SOCIAL/RELIGIOUS GROUPS**: Is government repression of social/religious groups occurring/increasing?

    1.3.2 **NTERNAL SECURITY MEASURES**: Is the government instituting or strengthening internal security measures in response to armed (guerrilla/insurgency/separatist) movements or terrorist/criminal activity?

  1.4. **STRUCTURAL/INSTITUTIONAL PROBLEMS**: Are there serious or worsening or institutional problems that could have destabilizing consequences?

    1.4.1. **CONSTITUTIONAL CONFLICT/CRISIS**: Is there a constitutional conflict/crisis?

    1.4.2. **ERODING LEGAL AUTHORITY/ADMINISTRATIVE FUNCTIONS**: Are legal authorities or administrative functions eroding?

an argument template. These might be links to Web pages, queries to databases or search engines, parameterized launches of other analytic tools, or references to cascaded templates. They capture an important aspect of an analyst's knowledge, namely, where and how to go about seeking information relevant to answering questions. Knowledge of this form is one thing that distinguishes an expert from a novice analyst. Discovery tools are captured on primitive questions within a template by storing their URLs along with short citation strings used to reference them. Again, simple text editing is all that is needed to define these.

Finally, the author should establish a *situation descriptor,* for a new template, that describes the type of situations for which the template is intended to be used. Unlike the other information provided by the user in defining a template, much of the information in a situation descriptor is chosen from a situation ontology rather than being free text. The situation ontology serves much the same purpose as a card catalog in a library; it establishes indices and terms that are useful for retrieving objects based upon the type of situation to which they are applied. For national security problems, these include the part of the world being analyzed (e.g., the continent, region, or country under assessment), the principal actor (e.g., the leadership, the government, or its people), the event (e.g., political, economic, financial, or currency), and the time period. These descriptions, with the exception of time, are selected from hierarchies of terms that are established through traditional knowledge engineering techniques. By indexing objects according to this situation ontology, both exact and semantically close matches can be automatically retrieved based upon a description of the situation of interest expressed in the same terms. These situation descriptors are augmented by free text fields where the specific aspects of the situation can be fully expressed; thus, the ontological terminology need not fully capture every distinction.

In practice, we have found that analysts are capable of authoring templates after minimal training, but that authoring high-quality templates is challenging and requires additional experience. To jump-start this process for problems of national defense, we convened a multidisciplinary team of experts to establish high-level templates for assessing the stability of nation states. The idea was to provide analysts with an example that they could then improve upon or adapt to specific situations, because it is easier to modify an example than to generate a template anew. The results have been well received. We imagine that variants of this high-level template will eventually be supported by cascaded templates that are more pointed. While the high-level template is useful in reminding analysts of the full range of indicators that need to be assessed and for generally organizing the analysis, their abstract nature prevents them from delivering much

in the way of expert guidance. Given that templates cascaded under this high-level template will address more specific and limited analytic tasks, we anticipate that they will capture expert knowledge suitable for guiding analysts in doing analytic tasks that fall outside of their areas of expertise. Thus, these templates capture and deliver best practice.

CAPTURING ANALYTIC PRODUCTS

Arguments are formed by answering the questions posed by a template and attaching the evidence that was used in arriving at the selected answers. In essence, an argument organizes the indications and warning signs for the given type of opportunity or threat.

Figure 3: Argument hierarchy showing answers

Answers are chosen from the multiple choices given by the associated template. If the available information does not allow the analyst to reduce the possible answers to a single choice, multiple ones can be selected bounding the answers that remain possible, given the available information. The *rationale* for answering in that way is recorded as a text string with attribution given to the answering analyst and the time that that answer was given.

Upon answering each question, the template's inference method is applied, deriving the answers to derivative questions. Using the traffic light metaphor, arguments can be displayed as a tree of colored nodes. Nodes represent questions, and colors represent answers. Figure 3 shows one such tree. The line of reasoning can be easily comprehended and the analyst is able to quickly determine which answers are driving the conclusion. By examining the high-value answers, the rationale behind the line of reasoning can be understood.

Information used as evidence to support the answers given in an argument is recorded as part of the argument. When information that is potentially relevant to answering a question posed is first found, it is entered as an *exhibit.* An exhibit assigns a unique identifier to the information, and records the URL for accessing it and a *citation* string for referring to it (typically consisting of some combination of title, author, and date). When the *relevance* of the information to the question at hand is determined, the exhibit is promoted to *evidence.* The relevance is recorded in two ways: as a text string explaining the significance and as the answer(s) to the question that would be chosen if the answer were to be based solely upon this evidence. The analyst making this assessment and the time of the assessment are recorded as well. When evidence is present, the rationale typically explains how the collective
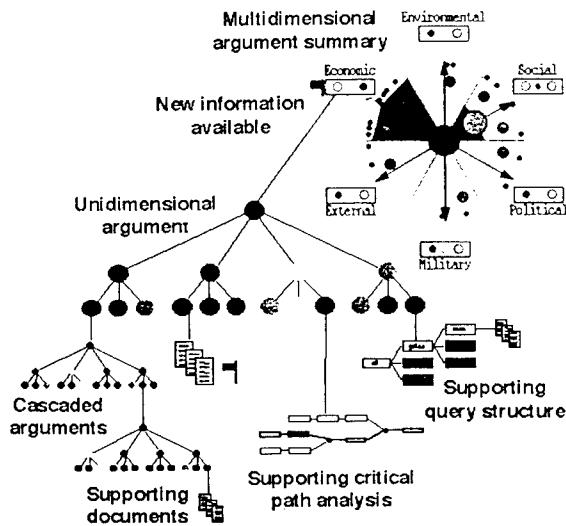
evidence supports the answer(s) chosen, explaining away that evidence that contradicts the answer and weaving together the supporting evidence to arrive at the stated conclusion.

When discovery tools are present, they can be used to aid in the collection of evidence. When these tools are based upon cascaded templates, cascaded arguments result from their use. In this way, the analyst can choose where they want to do a more thorough analysis, delving more deeply in a targeted way. A cascaded argument's conclusion can be automatically used as its relevance in support of the higher-level argument.

The analyst also chooses a *fusion method* for combining all of the evidence gathered supporting a single template question. The fusion method can be manual (i.e., the analyst answers the question based on his or her understanding of the evidence and its relevance) or automated (i.e., the answer is automatically reached by applying a combination method to the relevance of the supporting evidence). When an automated method is in use, changes in supporting arguments can ripple up through the arguments that they support, changing their conclusions.

As seen in Figure 4, complex lines of reasoning can be captured using this methodology. Here a multidimensional argument (i.e., a coordinated set of unidimensional arguments like those discussed) is graphically depicted at the top; it represents a coordinated assessment along multiple perspectives. It is supported by structured arguments as well as documents and analytic products produced by other tools. This structure allows analysts to quickly come to understand the reasoning of others and compare and contrast it with their own.

Figure 4: Cascaded structured arguments

Like argument templates, arguments too have associated situation descriptors. An argument's situation descriptor is like a template's situation descriptor except that it captures information pertaining to the prevailing situation for which the argument was developed. Like the situation descriptors associated with templates, they are used to find arguments that address related situations.

A CORPORATE MEMORY OF ANALYTIC KNOWLEDGE

To support the application of the structured argumentation methodology, SRI is developing SEAS, the Structured Evidential Argumentation System. SEAS has been developed as a Web server that communicates with remote browser-based clients. Through HTML and JavaScript, SEAS supports analysts in locating, understanding, and developing templates and arguments. This analytic knowledge is maintained within a knowledge-base management system, with ephemeral views served up upon demand. Figure 5 shows one such view of a primitive question within an argument.

Figure 5: SEAS argument in browser client

If we are to recognize future opportunities and threats, then we must relate the present to the opportunities and

threats of the past. We must understand how the current situation is like or unlike previous situations; how the indications and warning signs are similar or dissimilar; how previous opportunities or threats were recognized or missed; how previous opportunities or threats evolved and thereby how the present situation might evolve; and how previous situations were leveraged, mitigated, exacerbated, or missed. In short, we need a corporate memory that is more than a historical data repository; we need a corporate memory of analytic products and methods on which to base future analysis.

By recording and retaining analytic thinking in a common knowledge repository, analysts can leverage the thinking from the past and present when addressing new tasks. Based upon the indexing provided by the situation descriptors, potentially relevant templates and arguments can be found.

Beyond the analytic methods (i.e., argument templates), analytic products (i.e., arguments), and their associated situations (i.e., situation descriptors), we have found that analysts need additional means for associating meta-knowledge with these objects. To address this need, SEAS supports *memos*.

Memos are structured annotations that are attached to other objects within the SEAS knowledge base. Each memo includes text strings for its *subject* and *body* and a *type* selected from a pre-established set including critique, to do, summary, instruction, and assumption. Like arguments and templates, they have a designated *audience* that restricts their access by others; only those that are members of the audience will know of their existence. As such, memos provide a means for private, semiprivate, or public communication among analysts. Critiques are a way for contemporary analysts to contribute to each other's work. Assumptions might be added so that analysts in the future will better be able to interpret a historical analysis. Within SEAS, memos can be selectively filtered based upon their type, with graphical depictions indicating to the user where they can be found. This provides a ready means for analysts to find and interpret this form of meta-knowledge.

While analytic knowledge that is developed in SEAS is retained in its corporate memory, as are references to external analytic products used as evidence, there are times when one would like to import arguments produced using other technologies, so that they can be extended or otherwise modified. Our objective is to provide a means for the exchange of information among tools that can be said to produce arguments. If tools can be said to be argumentation tools, then they should be able to exchange arguments. Although argumentation tools share common concepts, they invariably have some unshared concepts, necessarily making importation imperfect.

Toward this objective, we are defining the Argument Markup Language (AML), an XML representation of arguments, and modifying SEAS to support the importation and exportation of these objects. The initial set of argumentation tools that we aim to support comprises those based upon Bayesian nets, particularly drawing from the Bayesian Net Interchange Format (Microsoft 2001), CIM (Veridian 2001), a structured argumentation tool developed at the same time as SEAS but with an emphasis on arguments about processes, and SEAS. While this is the initial set, we are aiming for a general design that will support a far greater number of tools, including those based upon both numeric and symbolic representations of certainty. We began by looking for common semantic concepts within these tools and using terminology from the Law to capture them. Legal terminology was selected since the Law already includes a rich notion of argumentation from evidence and provides a technology-neutral vocabulary, many of whose terms are in common use. An initial version of AML has been defined, and CIM and SEAS are being modified to support it.

RELATIONSHIP TO OTHER WORK

The structured argumentation methodology and SEAS were developed to aid those performing analytic tasks. In particular, we were not looking to automate the analytical reasoning that they perform, but to facilitate it. This methodology

- Encourages careful analysis, by reminding the analyst of the full spectrum of indicators to be considered

- Eases argument comprehension and communication by allowing multiple visualizations of the data at different levels of abstraction, while still allowing the analyst or decision maker to "drill down" along the component lines of reasoning to discover the detailed basis and rationale of others' arguments

- Invites and facilitates argument comparison by framing arguments within common structures

In addition, SEAS provides synchronous and asynchronous access to a corporate memory of analytic methods and results, which allows analysts to work together on common arguments as well as leveraging historical results. Collaboration, then, is recognized as an important part of the process and leads to arguments that are richer than would have been otherwise the case. The Web is an ideal medium for collaboration, driven by the near ubiquity of browser software and the information explosion on the Web.

The goals of structured argumentation differ from those of other knowledge capturing tasks. In most knowledge engineering efforts the objective is to elicit and represent the knowledge of humans in machines so that the machine can later use this knowledge to approximate the reasoning of humans. This largely requires that the knowledge be

captured in not natural language but in ontological structures that can be more readily manipulated by machines. Examples include Cyc [1], DARPA High Performance Knowledge Bases [2], DARPA Rapid Knowledge Formation [3], GKB-Editor [13], EcoCyc [8], and Ontolingua [5].

Today, intelligence analysts usually capture their knowledge in text documents. Typically, these documents have minimal structure, limited to section titles that break up the document. These intelligence reports are intended for human consumption. However, because of their limited structure they are time consuming to read and understand. To compare one report with another requires that both reports be read, and it is up to the reader to find common and uncommon aspects of the underlying reasoning. It is also up to the reader to extract the analytic method if it is to be employed in doing related analyses. Searching a collection of such reports to find ones that might be related to the current problem of interest is also time consuming. Of course, word processing and search engines can help to speed this process, but the level of aid is fundamentally limited.

Structured argumentation fits between these two approaches. It introduces more structure into the analytic environment than is in use today but not as much as typical knowledge engineering efforts. The analytic method is separated from the analytic products, resulting from its application. The analytic method is broken down into a set of smaller analytic tasks, with their interrelationships captured. Methods for acquiring information in support of these analytic tasks are also broken out. In structure, analytic results parallel the analytic methods on which they are based, with links to the information that supports the conclusions retained, and to the interpretations of that information relative to each analytic task. The type of situation for which a method was designed and for which a result was produced is also captured. However, much of the knowledge captured remains in natural language. In fact, when one compares an analytic product produced using SEAS with a contemporary analytic product expressed in a text document, one finds that most of the text found in the document is found within the structured argument. The structure has not replaced the words as much as it has augmented them, making it possible for the machine to aid the analysts in new ways.

This approach bears a resemblance to some recent work in support of knowledge mobility [6]. In this work, the rationale and sources of knowledge, drawn upon in engineering knowledge, are retained within the resulting formal structures. So doing allows others to more readily understand why knowledge was captured as it was, making it easier to reapply, extend, or modify. These resilient hyper knowledge bases use a layered architecture to capture knowledge from the most to the least formal.

This work can be viewed as building up from formal knowledge to less formal knowledge, while our work can be viewed as building down from informal knowledge to more formal knowledge.

The structure introduced into the analytic process by the structured argumentation methodology, although motivated by the desire to help humans, also represents an opportunity for greater accessibility to automated methods. These methods might attempt to provide critical feedback to the analyst or automatically make corrections. Such feedback can be readily communicated using the SEAS memo facility; thus, automated collaborators would interact in the same way as human collaborators. Some of these capabilities could be introduced without the need to perform any natural language understanding; other capabilities might require some limited understanding; still others would benefit from more comprehensive natural language understanding.

Without the introduction of natural language understanding, we intend to develop an automated argument critic that provides several kinds of feedback. For example, such a critic could examine the answer to every question in an argument, to determine if the answer is supported by evidence, if each piece of evidence includes a statement of its relevance, and if the rationale for the overall answer is given. It could also check for overreliance on any single document supporting the answer to multiple questions, since overuse of any source of information leaves one vulnerable to its accuracy and truthfulness.

The corporate memory of arguments presents other opportunities. By comparing the focal argument to successful arguments from the past, other useful sources could be identified that have not been used in the focal argument. Likewise, sources that had previously led to poor results could be flagged. Similarly, more complex patterns of previous use could be exploited.

With the aid of some natural language understanding technology, coupled with inference capabilities based upon formal knowledge representations, we might develop more sophisticated aids. These might look to find logical contradictions in the way that evidence was interpreted or in the rationale accompanying answers given. They might also look to suggest alternative interpretations.

CONCLUSIONS

We believe that our structured argumentation methodology, as implemented in SEAS, has shown that the addition of even minimal structure into the analytic process can aid analysts in developing, communicating, explaining, and comparing analytic results. An important aspect of this methodology is the retention of direct links to the source material and its interpretation relative to the conclusions drawn, allowing analysts to readily comprehend the thinking of others. This, coupled with a

collaborative environment and a corporate memory of analytic thought, retaining the analytic methods and products of an enterprise, allows analysts to leverage the thinking of others both past and present. Finally, even though our methodology was motivated by the desire to help human analysts, it lays the groundwork for the introduction of automated methods to substantially aid or partially supplant human analytic reasoning. We contend that this methodology complements those knowledge capturing methodologies that strive to formally represent human knowledge in rich ontological structures.

ACKNOWLEDGEMENTS

REFERENCES

[1] Cycorp (2001); The Cyc Knowledge Server; http://www.cyc.com/

[2] DARPA (2000); High Performance Knowledge Bases Project, http://reliant.teknowledge.com/HPKB/

[3] DARPA (2001); The Rapid Knowledge Formation Project; http://reliant.teknowledge.com/RKF/

[4] Dung, P. (1995); On the Acceptability of Arguments and Its Fundamental Role in Nonmonotonic Reasoning, Logic Programming and n-Person Games; *Artificial Intelligence* 77 (pp. 321-58)

[5] Fikes, R., Farquhar, A., and Rice, J. (1997); Tools for Assembling Modular Ontologies in Ontolingua; Knowledge Systems Larobatory, Stanford University

[6] Gil, Y. (2001); Knowledge Mobility: Semantics for the Web as a White Knight for Knowledge-Based Systems, forthcoming

[7] Heuer, R. (1999); *Psychology of Intelligence Analysis*; Center for the Study of Intelligence, Central Intelligence Agency

[8] Karp, P., Chaudhri. V., and Paley, S. (1999); A Collaborative Environment for Authoring Large Knowledge Bases; in *Journal of Intelligent Information Systems*, Vol. 13 (pp. 155-194)

[9] Loui, R. (1987); Defeat Among Arguments: A System of Defeasible Inference; in *Computational Intelligence*; Vol 3 (pp. 100-106).

[10] Lowrance, J., Harrison, I., and Rodriguez, A. (2000); Structured Argumentation for Analysis; in *Proc. 12th Int. Conf. on Systems Research, Informatics, and Cybernetics: Focus Symposium on Advances in Computer-Based and Web-Based Collaborative Systems* (pp. 47-57)

[11] Lorenzen, P. and Lorenz, K. (1977); *Dialogische Logik*; Wissenschaftliche Buchgesellschaft Darmstadt

[12] Microsoft (2001); XML Belief Net File Format; http://www.research.microsoft.com/dtas/bnformat/

[13] Paley, S., Lowrance, J., and Karp, P. (1997); A Generic Knowledge Base Browser and Editor; in *Proc Ninth Conf. on Innovative Applications of Artificial Intelligence*

[14] Perelman, C. (1970); *Le Champ de l'argumentation*; Bruxelles: Éditions de l'Université

[15] Perelman, C. and L. Olbrechts-Tyteca. (1958); *Traité de l'argumentation - la nouvelle rhétorique*; Bruxelles: Éditions de l'Université

[16] Pollock, J (1987); Defeasible Reasoning; in *Cognitive Science*, Vol. 11 (pp. 481-518).

[17] Sycara, K. (1990); Persuasive Argumentation in Negotiation, *Theory and Decision* Vol. 28, No. 3 (pp. 203-42)

[18] Stokke, R., Boyce, T., Lowrance, J., and Ralston, W. (1994); Evidential Reasoning and Project Early Warning Systems; *Journal of Research and Technology Management*

[19] Toulmin, S. (1958); *The Uses of Arguments*; Cambridge University Press

[20] Veridian Systems Division (2001); Critical Intent Modeling; unpublished

Knowledge Capture and Utilization in Virtual Communities

Yasmin Merali
Information Systems Research Unit
Warwick Business School
The University of Warwick
Coventry CV4 7AL
United Kingdom
Yasmin.Merali@warwick.ac.uk

John Davies
Advanced Business Applications
BTexact Technologies
Adastral Park
Ipswich IP5 3RE
United Kingdom
john.nj.davies@bt.com

Abstract
The literature on knowledge management highlights issues of fit between IT-based systems for knowledge management and the socially situated leveraging of knowledge assets by organisations [1]. This paper explores the way in which a knowledge-sharing environment (KSE) can facilitate knowledge capture and utilization in virtual communities. The KSE (Jasper II) is a system of information agents for organising, summarizing and sharing knowledge from a number of internal and external sources, including the World Wide Web (WWW). The paper describes the features and functionality of Jasper II, and goes on to show how it can be leveraged to support the capture of both tacit and explicit knowledge in virtual communities. The final discussion focuses on the dynamics of the knowledge capture and utilization process, highlighting the importance of the feedback mechanisms that enable the KSE to meet the specific needs of diverse, evolving communities. It suggests that besides supporting the dynamic knowledge requirements of communities, the KSE can play a key role in the evolution of existing communities.

Keywords
Knowledge management, knowledge sharing environments, virtual communities

INTRODUCTION
Whilst the ubiquity of communication and access to information afforded by the internet, intranets and extranets provides unprecedented opportunity for the exploration of inter- and intra- organizational information and knowledge resources, it has created new challenges for the effective exploitation of these resources.

This paper is concerned with the way in which IT-based systems can enhance the utilization and leveraging of knowledge in organisations. It shows how a knowledge sharing environment (KSE) can be utilized to explore and exploit both tacit and explicit knowledge processes in virtual communities. The next section, outlining key issues in knowledge management today, is followed by a description of the salient features of the KSE (Jasper II). The final section concludes with a discussion on the socially situated utilization of Jasper II to support knowledge workers and meet the specific needs of diverse, evolving communities.

ISSUES IN KNOWLEDGE MANAGEMENT
Knowledge processes are often classified according to whether they entail knowledge creation or knowledge reuse. However in effect, the two are not orthogonal, as new knowledge builds on (or, alternatively, uses as a point of departure) existing knowledge [2]. Knowledge reuse entails three main activities [3]:

- location of documents or records that *may* contain relevant explicit knowledge,

- selection of relevant/significant items from the set retrieved through the search and

- applying the knowledge in a particular context.

The escalation in the volume of available information has exacerbated problems of information location, selection and evaluation (of quality and currency of retrieved information). The deployment of IT to automate the process of locating, retrieving, delivering and disseminating information makes good sense. Most knowledge management systems attempt to deal with these aspects, albeit with varying degrees of success.

The processes of selection, evaluation and application are all context dependent and socially situated (i.e. people determine what these processes look like, and the processes themselves shape, and are shaped by, the standards, values and expectations of the society that gives rise to them). A piece of information perceived to be highly valuable by one person or group here and now may not have the same value for a different person or group at the same time in some

other place, engaged in some different task, or working with a different value system. Equally the value of information may change over time.

Taking knowledge management to be the process by which organisations manage the creation, capture, dissemination and utilization of knowledge, the main challenges for practitioners include:

- scanning multiple internal and external sources effectively,

- meeting the diverse, dynamic, context specific information needs of individual and groups of knowledge workers in real time,

- capturing the knowledge that is generated when people use knowledge to do their jobs

- getting people to disseminate what they have learnt and

- getting people to use knowledge that has been generated by others (overcoming the "not invented here syndrome" and getting people to trust and value the contributions of others).

The knowledge capture issue is often discussed in terms of capturing explicit and tacit knowledge. Explicit knowledge is that which can be expressed in language and can is therefore be codified and recorded. Tacit knowledge is that which cannot be expressed in language [4, 5]. It is generally accepted that tacit knowledge can be transmitted through socialization processes [6] such as a master-apprentice "learning by accompanying, watching, helping and copying" arrangements. Most organisational action is context-specific, and tacit knowledge underpins the choice of appropriate actions for given situations. It is thus a valuable resource, and failure to manage it effectively can lead to loss of expertise when people leave, failure to benefit from the experience of others, needless duplication of a learning process, and so on.

Most knowledge management systems cater for the organisation, storage and dissemination of explicit knowledge. For access to tacit knowledge, they provide a "yellow pages" facility for the location of people who are considered to be particularly knowledgeable about particular subjects and situations. In the final section we will see how KSE's like Jasper II can contribute more effectively to the process of sharing tacit knowledge.

Three organisational trends have added to the complexity of the problem:

- the move towards flexible work practices resulting in increasing numbers of mobile and home workers, so that people who would normally share information contexts are no longer co-located,

- the increasing importance of cross-functional and inter-organisational collaborative work practices and project-based organisation: this has generated the need for people to share information contexts with others from disparate disciplines and backgrounds, and

- the thrust towards responsive organisation: in the increasingly interconnected world, there is greater uncertainty in the competitive context, and the context is more dynamic, demanding fast (and often innovative) organisational responses. This trend underlines the importance of knowledge creation and reuse and highlights its relationship to organisational learning [7].

The activities of knowledge creation and organisational learning take place in the social context [8, 9]. Consequently the focus of knowledge management experts has extended from the design of *systems* to capture and deliver explicit information to the development of virtual *environments* to foster and support knowledge networks and virtual communities of practice.
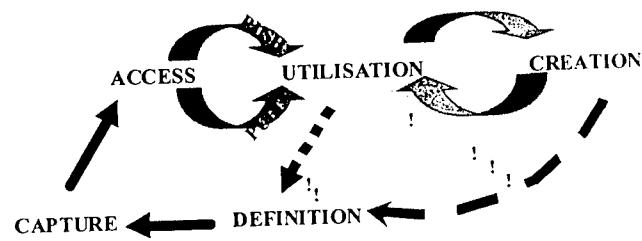
Communities of Practice

The term 'community of practice' [10] describes the informal groups where much knowledge sharing and learning takes place and has been increasingly applied in the knowledge management context. Essentially a community of practice is a group of people who are 'peers in the execution of real work' [11]. They are typically not a formal team but an informal network, each sharing in part a common agenda and shared interests.

Knowledge Networks

The networking aspect is particularly important in dynamic contexts in which knowledge workers may be confronted with the need to locate and harness rapidly the expertise of individuals from disparate disciplines and locations with whom they have no continuity of shared interest or common agenda. Communities of practice and social networks both highlight the importance of the link between social capital and knowledge resources for effective knowledge management.

Figure 1 (modified from [12]) provides a schematic representation of the main issues relating to the process of knowledge management discussed so far (problem areas are denoted by the "!" sign, and dotted lines represent weak or inadequate links).

Figure 1: Issues in the Process of Knowledge Management (modified from [12])

Most knowledge management systems aspire to capture information matching specified user profiles and queries. Increasingly, the more advanced products on the market offer both push (proactive delivery of information that matches individual user profiles or specific tasks) and pull facilities (reacting to user requests).

The bigger challenge for knowledge management lies in the problem of capturing and re-using knowledge that is generated during knowledge work (depicted by the cycle on the right in the diagram). Whilst individuals and groups working on a problem may learn significantly from their experience, the knowledge created by this process tends to remain private. This is due to a number of reasons including

- the time and effort required to analyse and record what has been learnt,

- the lack of a context within which to articulate individual learning,

- the lack of recognition for individual contributions to the organisational knowledge pool, and

- the "knowledge is power" syndrome and the fear of losing their niche in the organisation.

The establishment of communities of practice is thought by many to offer a way of overcoming some of these barriers to knowledge sharing [11], and internet-based knowledge support environments are seen as a way of enabling the establishment of virtual communities of practice. The next section describes the features of one such environment, and the final section discusses the way in which KSEs can be deployed to foster and sustain such communities and networks of communities.

FEATURES OF THE KNOWLEDGE SHARING ENVIRONMENT (KSE)

In this section we outline the main features of Jasper II, a knowledge sharing environment (KSE). Jasper II is comprised of a system of intelligent software agents that retrieve, summarize and inform other agents about information considered to be of some value by a Jasper II user. The information may be from a number of different sources: it can be generated by the user himself, it can be an internet/intranet page, archived information from internal/external repositories or from another application on the user's own computer.

The process by which Jasper II agents search for, select, retrieve, and present information that matches user-specified profiles and queries is outlined below.

Storage and Organisaiton o fInforamtion

Information is not copied from its original location to the local server: the agents store only the relevant meta-information. This meta-information is then used to index on the actual information when a retrieval request is made. In the case of WWW-based information the URL of the WWW page is then added to the Jasper II store. Similarly, when the user wishes to store some information from a source other than WWW, (s)he can enter the information in a text box on their WWW browser and can again supply a relevant annotation. The information thus entered could be from a document in another format or might be a note or snippet of knowledge that the user wishes to enter directly. This information is converted to a WWW HTML page on the user's Jasper II server and stored as before.

Essentially, the Jasper II store is a simple term-document matrix M.

Each user has a personal agent that holds a user profile based on a set of key phrases which models that user's information needs and interests. As we will see below, the modeling process is an adaptive one with the Jasper II agent suggesting modifications aimed at refining the profile to better reflect the user's actual information needs and interests.

A major advantage of the Jasper scheme of using explicit terms (words and phrases) to represent a user's interests via their profile is that the profile is explicitly available to the user at all times. In other schemes (e.g. using neural or Bayesian networks), the user profile is essentially a "black box" which is invisible to the user. Trials of Jasper with around 1000 users in one organisation revealed that users preferred the version that made their profiles visible.

Matching and Selection of Information

Jasper uses the vector space model [13] for assessing the relevance of shared information to individual users. Essentially, the shared information (document) and user profile (query) are placed in an n-dimensional vector space, where n is the number of unique terms (words and phrases) in the data set. A vector matching operation, based on the cosine correlation used to measure the cosine of the angle between vectors can then be used to measure the similarity between a document and a query (or user profile). Terms are weighted according to a variant of the $tf.idf$ weighting scheme [14], which takes into account the frequency of the term in the given document, the document length and the frequency of the term across the entire Jasper document collection, with more weight being given to rarer terms.

The similarity of Jasper users is calculated by calculating the Dice coefficient for their profiles. The Dice coefficient provides a measure of similarity between 2 profiles based on the number of terms (words and phrases) which co-occur in the profiles, normalised for profile length. [15]

Dissemination and Delivery

When a user finds information of sufficient interest to be shared with their community of practice, a 'share' request is sent to Jasper II via a menu option on his or her WWW browser. Jasper II then invites the user to supply an annotation to be stored with the information. Typically,

94

users provide annotations to do one or more of the following:

- give reasons for sharing the information,

- provide a comment on the information content,

- highlight the relevance of the information to current issues and contexts, and

- highlight the relationship of the information to past discussions or postings.

At storage time, the Jasper II agent performs four tasks:

- it creates an abridgement of the information, to be held on the user's local Jasper II server. This summary is created using the ProSum automatic text summarisation tool. Access to this locally held summary enables a user to quickly assess the content of a page from a local store before deciding whether to retrieve (a larger amount of) remote information,

- it analyses the content of the page and matches it against every user's profile in the community of practice. If the profile and document match strongly enough, Jasper II emails the user, informing him or her of the page that has been shared, by whom and any annotation added by the sharer,

- it matches the information against the sharer's own profile. If the profile does not match the information being shared, the agent will suggest phrases that the user may elect to add to their profile. These phrases are those reflecting the information's key themes and concepts and are automatically extracted using the ProSum system. Thus Jasper II agents have the capability to adaptively learn their user's interests by observing the user's behaviour and

- for each document, it makes an entry in the Jasper II store, holding keywords, an abridgement of the document, document title, user annotation, universal resource locator (URL), the sharer's name and date of storage.

In summary, Jasper II allows a user to store information of interest using an enhanced, shared community bookmark concept. However, this facility goes well beyond the bookmarks familiar from WWW browsers such as Netscape Communicator, in that in addition to the reference to the remote WWW document, a summary of the document, an annotation, date of storage and the user who stored the information are recorded in a shared store. Furthermore, Jasper II can be used to store and organise information from many sources and in many formats (rather than only WWW-based information).

Proactive Delivery

As described above, when information is stored by a Jasper II agent, the agent checks the profiles of other agents' users in its particular community (the set of users who contribute to that particular Jasper II community). If the information matches a user's profile sufficiently strongly, an email message is automatically generated by the agent and sent to the user concerned, informing the user of the discovery of the information. Thus in cases where a user's profile indicates that they would have a strong interest in the information stored, they are immediately and proactively informed about the appearance of the information.

Keyword Retrieval – Accessing Information and People

From his or her Jasper II home page, a user can supply a query in the form of a set of key words and phrases in the way familiar from WWW search engines (see Figure 2). The Jasper II agent then retrieves the most closely matching pages held in the Jasper II store, using a vector space matching and scoring algorithm [16].

In addition to these pages from the Jasper II store, the agent can also retrieve a set of pages from an organisation's intranet and from the WWW. The agent then dynamically constructs an HTML page with a ranked list of links to the pages retrieved and their abridgements, along with the scores of each retrieved page. In the case of pages from the Jasper II store, any annotation made by the original user is also shown.

Figure 2 depicts a typical Jasper II home page displaying retrieved information. In addition, a series of buttons are provided so that the user can:

- add their own comment or annotation to information stored by another user,

- indicate interest or disinterest in a particular piece of information – this feedback will be used to modify the user's profile,

- examine a locally held summary of the information before deciding to download all the information, and

- ask their Jasper II agent to identify other users with an interest in the information under consideration. We will have more to say about this capability to identify other *users* as well as *information* later in this paper when we look at the role of Jasper II in managing the tacit dimension of knowledge management.

Figure 2: A typical Jasper II home page

What's new

A user can ask his or her Jasper II agent "What's new?" The agent then interrogates the Jasper II store and retrieves the most recently stored information. It determines which of these pages best match the user's profile. A WWW page is then presented to the user showing a list of links to the most recently shared information, along with annotations where provided, date of storage, the sharer and an indication of how well the information matches the user's profile (the thermometer-style icon in Figure 2).

This What's New information is in fact displayed on the user's Jasper II home page, so that whenever they access the system, they are shown the latest information.

Adaptive Agents

We have already mentioned that Jasper II agents adapt to better understand their user's interests over time. There are two types of event which trigger the profile adaptation process.

As discussed above, when a user is sharing some information, if the sharer's profile does not match the information being stored Jasper II will automatically extract the main themes from the information using ProSum. The user's agent then suggests to the user new phrases that they may wish to add to their profile. The user can accept or decline these suggestions.

Similarly, when information stored by another member of the community is retrieved by a user using one of the methods described earlier, a feedback mechanism is provided whereby the user can indicate interest or disinterest in the information by clicking on a button (indicated by ☺ or ☹ as shown in Figure 2). Again, the

agent will suggest to the user phrases that should be added to or removed from the profile.

SOCIALLY SITUATED DEPLOYMENT OF THE KSE

In the last section we focused on the *technical* aspects of Jasper II and on the sharing and storing of explicit knowledge. Explicit knowledge we take to be that knowledge which has been codified in some way. This codification can take place in many different media (paper, WWW page, audio, video, and so on). This captured, codified form is referred to as a "knowledge artifact" in the discussions that follow. In the context of Jasper II, by explicit knowledge, we mean the information shared in Jasper II, along with the meta-information associated with it such as the sharer, the annotations attached to it, and so forth.

We now turn to the *social* aspects of the system, involving the organisational capture and utilization of socially situated and contextual (sometimes tacit) knowledge. We revisit the issues of knowledge management highlighted at the beginning of this paper and discuss the way in which the features of a KSE like Jasper II can be leveraged to facilitate the more dynamic aspects of the knowledge management process in virtual communities of practice.

Capture and Codification of Explicit Knowledge and the Issue of Context-Specific Knowledge

Before looking at the socially situated processes of knowledge management it is useful to review some of the fundamental characteristics of knowledge reuse and the way in which formal processes of knowledge capture and codification deal with contextual knowledge.

The Formal Process

There are three major roles in the knowledge reuse process [3]:

- the *knowledge producer* (who originally expresses and records explicit knowledge),

- the *intermediary* (who structures knowledge for reuse by indexing, summarising, sanitising and packaging it), and

- the *knowledge consumer* (who retrieves the knowledge content and applies it in some way.

It has been shown that the way in which producers record knowledge differs significantly depending on whether they are recording it for themselves, for similar others or for different others.

Whilst one individual can perform all three roles, it is generally considered inadvisable for the producer to also act as the intermediary if the knowledge is intended for use by somebody else. This is because of the issue of context: for individuals who work in similar contexts, contextual detail associated with the application of a piece of knowledge is helpful in understanding the utility value of

that knowledge. However for contextually distant workers, the inclusion of detail creates confusion and acts as noise, obfuscating the intrinsic value of the knowledge being transmitted [17]. The producer is too close to the original context to be able to sanitise the knowledge effectively.

The Need for a More Expedient Complementary Mechanism

The formal knowledge capture process therefore has the following characteristics:

- it is time consuming,

- it tends to sanitise (and strip away the context from) knowledge descriptions and

- it tends to "freeze" knowledge definitions.

These characteristics contribute to the development of validated, stable knowledge repositories. To reuse this knowledge, the knowledge consumer must recognise or (re)define the context within which to best leverage the retrieved knowledge.

On the other hand if we are

- interested in capturing knowledge from the cycle on the right hand side of Figure 1 (i.e. capturing the context specific by-product of knowledge work), and

- dealing with dynamic contexts in which the pressures to act appropriately in a given time and space are high, so that the *right* context specific information is very valuable, but the shelf-life of context-specific knowledge is low (because the context is dynamic),

we need to find more expedient but robust ways of dealing with the needs of knowledge workers in a complementary fashion alongside the formal process (which remains valuable for archiving validated knowledge claims and for providing access to stable knowledge resources).

The KSE-Enabled Virtual Community of Practice

The virtual community of practice presents itself as a way of organising the less formal, more socially embedded knowledge management activities. The following discussion is based on observations made over a period of time in several different types of Jasper II communities.

As highlighted earlier, members of a community share a degree of contextual proximity, rendering the sanitisation process unnecessary, enabling the exploitation of contextual information. As outlined below, a KSE-enabled community of practice plays a variety of roles in the knowledge management process:

As a medium for the diffusion of knowledge generated as a by-product of knowledge reuse:
When a Jasper II user retrieves a useful knowledge artefact, annotates it and decides to share it with the rest of the community, the circulated artefact effectively incorporates the sender's judgement which is a product of his or her engagement with the artefact and his or her attempt to evaluate its utility.

As a mechanism for facilitating and expediting the knowledge reuse process
When a Jasper II user selects and commends a knowledge artefact to his or her peers, the artefact enters the community context. Subsequent annotations may serve to refine the contextual utility of the artefact. The selection and introduction of the knowledge artefact into the community space and the subsequent additions of annotations effectively act as collaborative filtering and contextualisation mechanisms.

As a substrate for the co-evolution of shared awareness
The sharing and annotation activities reinforce a shared understanding amongst the members of the community. Because Jasper II agents are able to search a variety of internal and external sources, and because between them the different individuals instruct their agents to search a diversity of sources, the community space can be populated with items of current relevance, and the annotation facility enables capture of individual perspectives on the items. Because Jasper II enables this type of dynamic contextualisation of retrieved knowledge artefacts, it can be utilised to raise the collective awareness of contemporaneous issues and views, and can help individual perceptions to evolve in step with the demands of the dynamic external context.

In summary, the mechanisms outlined above highlight the way in which Jasper II can support the dynamic, instantaneous and sometimes transitory utilisation of knowledge generated as a by product of knowledge work. The process of annotating and sharing knowledge artefacts can be considered to

- feed off the shared nature of the community context,

- reinforce the shared nature of community context and

- refresh and update the collective perception of the community context.

This type of utilization of Jasper II is complementary to the more formal processes described earlier. Formally constructed archives are an information source for agents to search. Jasper II logs constitute an organisational "memory" in addition to providing up-to-date data that can be used in formal processes for the evaluation of the popularity (frequency of access) and utility of the knowledge artifacts.

A KSE-Enabled Social Network

One way in which a system such as Jasper II can encourage the sharing of tacit knowledge is by using its knowledge of the users within a community of interest to put people who would benefit from sharing their knowledge in touch with one another automatically.

One important way we gain new insights into problems is through 'weak ties', or informal contacts with other people [18, 19]. Everyone is connected to other people in social networks, made up of stronger or weaker ties. Stronger ties occur between close friends or parts of an organisation where contact is maintained constantly. Weak ties are those contacts typified by a 'friend of a friend' contact, where a relationship is far more casual. Studies have shown that valuable knowledge is gathered through these weak ties, even over an anonymous medium such as electronic mail and that weak ties are crucial to the flow of knowledge through large organisations. People and projects connected to others through weak ties are more likely to succeed than those that are isolated [20, 21].

Though Jasper II does not explicitly support weak ties, initial trials of Jasper II have shown a number of features that support social networking:

- people contributing information are more likely to make informal contact with others using Jasper II,

- Jasper II can identify those people who could be sources of information and

- the store of URLs, with associated annotations and other meta-information, becomes a long-term memory for the community.

User profiles can be used by the Jasper II system to enable people to find other users with similar interests. The user can request Jasper II via their WWW client to show them a list of people with similar interests to themselves. Jasper II then compares their profile with that of every user in the store and returns to the WWW client for viewing by the user a list of names of users whose interests closely match their own. Each name is represented as a hypertext link which when clicked initiates an email message to the named user. Profiles in Jasper II are a set of phrases and the vector space model can be used to measure the similarity between two users. A threshold can then be used to determine which users are of sufficient similarity to be deemed to 'match'.

This notion is extended to allow a user to view a set of users who are interested in a given document. When Jasper II presents a document to the user via their WWW client using the "What's new?" facility (see above), there is also a hyperlink presented which when clicked will initiate a process in the Jasper II system to match users against the document in question, again using the vector cosine model. Jasper II determines which members of the community match the relevant document above a predetermined threshold figure and presents back to the user via their WWW client a list of user names. As before, these names are presented as hypertext links, allowing the user to initiate an email message to any or all of the users who match the document.

In addition, as discussed earlier, a user can carry out a keyword search on other users and thus identify users with an interest in a particular subject.

In this way, Jasper II, while not claiming to actually capture tacit knowledge, provides an environment which actively encourages the sharing of tacit knowledge, perhaps by people who previously would not otherwise have been aware of each other's existence.

Networks of Communities

Because Jasper II allows individuals to be members of multiple virtual communities concurrently, it supports cross-fertilisation of ideas between communities. This has obvious advantages for individuals who are involved in cross-boundary projects, and it can serve to counteract the institution of "silo" mentalities amongst members of close-knit communities.

More significantly for the knowledge management process, this structure of networked communities makes it possible to deploy cross-functional, multi-skilled teams without sacrificing access to the collective and specific expertise of individual communities.

CONCLUSION

The main purpose of this paper was to explore the role of KSEs in the facilitating knowledge capture and utilization in virtual communities of practice. The following list summarizes the key concepts emerging from this discussion

Organisational Learning

The discussion highlighted the importance of capturing and reusing the knowledge that is generated as a by-product of knowledge work. This is the knowledge resulting from individual "learning by doing". In showing how features of the KSE can be utilized to leverage this type of knowledge in virtual communities, we effectively described a process for the transfer of individual learning to organisational learning.

Dynamic Contextualisation

The discussion also highlighted the importance of rapid dynamic contextualisation of retrieved knowledge artifacts and the role of the shared community understanding in expediting this process.

Networking

The other important aspect to emerge from this discussion was the notion of using KSEs to support networking at both the individual and community levels. The importance of social networks in knowledge management is well established, and the concept of inter-community networking represents an important mechanism for sustaining a diversity of community-based expertise within an open structure enabling cross-fertilization of ideas between different virtual communities.

In conclusion it is important to note that KSEs like Jasper II are effective in supporting and sometimes enhancing formal and informal practices of knowledge management, but their

effectiveness is predicated on their sensitive deployment within the social and organisational contexts.

REFERENCES

1. Merali, Y., "Information, Systems and *Dasein*", in *Systems for Sustainability: People, Organisations and Environments*, Stowell, F. , McRobb, I., Landor, R., Ison, R., Holoway, J., (Editors); pp 595-600, Plenum, (1997).

2. Schumpeter, J.A., *The Theory of Economic Development*. Harvard University Press, Cambridge. MA, (1934).

3. Markus, M.L., "Toward a Theory of Knowledge Reuse: Types of Knowlwdge Reuse Situations and Factors in Reuse Success " *Journal of Management Information Systems,* 18 (1), pp 57-93 (2001).

4. Polanyi, K., *Personal Knowledge: Towards a Post-Critical Philosophy*, Routledge and Kegan Paul, London, (1958).

5. Polanyi, K. ,*The Tacit Dimension*. Routledge and Kegan Paul, London, (1967).

6. Nonaka, I. and Takeuchi, H., *The Knowledge-Creating Company*, Oxford University Press, New York, (1995).

7. Argyris, C. and Schon, D. A., *Organisational Learning*, Addison-Wesley, Reading, MA, (1978).

8. Merali, Y., "Leveraging Capabilities: A Cognitive Congruence Framework" in *Knowledge Management and Organizational Competence*, Ed. Sanchez, R., Oxford University Press, New York, (2001).

9. Merali, Y., "Individual and collective congruence in the knowledge Management Process", *Journal of Strategic Information Management, 9 (2-3)*: Special Issue on Knowledge Management and Knowledge Management Systems, (2000), pp 213–234.

10. Wenger, E., *Communities of Practice*, Cambridge University Press, Cambridge, UK, (1998).

11. Brown, J. S. and Duguid, P.. "Organizational learning and communities of practice: Toward a unified view of working, learning and innovation", *Organization Science*, 2, pp 40-57, (1991).

12. Merali, Y. "Information Technology and *Dasein*" *Working Paper*, Warwick Business School, (2000)

13.Harman, D., "Ranking Algorithms", *in Information Retrieval*, Frakes, W. and Baeza-Yates, R., Prentice-Hall, New Jersey, USA, 1992.

14.Salton, G. & C. Buckley, "Term-weighting approaches in automatic text retrieval", *Information Processing & Management,* 24(5), pp 8-36, 1988.

15. McGill, M.et al., 1979. "An evaluation of factors affecting document ranking by IR Systems," *Project Report.* Syracuse, NY, USA: Syracuse University School of Information Studies.

16. Salton, G., *Automatic Text Processing*. Reading, Mass., USA: Addison-Wesley, (1989).

17. Ackerman, M.S. *Definitional and Contextual Issues in Organizational and Group Memories*, University of California, Irvine, (1994), Available at http//www.ics.uci.edu/-ackerman/.

18. Granovetter, M., "The Strength of Weak Ties", *American Journal of Sociology, 78,* 1360-1380, (1974).

19. Granovetter, M., "The Strength of Weak Ties: A Network Theory Revisited", in *Social Structure and Network Analysis*, Marsden, P. and Nan, L. (Editors), Sage Publications, California, (1982)

20. Constant, D., Sproull, L. and Kiesler, S., "The Kindness of Strangers: The Usefulness of Electronic Weak Ties for Technical Advice", *Organization Science, 7* (2), 119-135, (1996).

21. Hansen, M.T., "The Search-Transfer Problem: The Role of Weak Ties in Sharing Knowledge Across Organisation Subunits", *Working Paper*, Harvard Business School, 1997

Capturing knowledge of User Preferences: Ontologies in Recommender Systems

Stuart E. Middleton, David C. De Roure and Nigel R. Shadbolt
Department of Electronics and Computer Science
University of Southampton
Southampton, S017 1BJ, UK
Email : {sem99r,dder,nrs}@ecs.soton.ac.uk

ABSTRACT

Tools for filtering the World Wide Web exist, but they are hampered by the difficulty of capturing user preferences in such a dynamic environment. We explore the acquisition of user profiles by unobtrusive monitoring of browsing behaviour and application of supervised machine-learning techniques coupled with an ontological representation to extract user preferences. A multi-class approach to paper classification is used, allowing the paper topic taxonomy to be utilised during profile construction. The Quickstep recommender system is presented and two empirical studies evaluate it in a real work setting, measuring the effectiveness of using a hierarchical topic ontology compared with an extendable flat list.

Keywords

Ontology, recommender system, user profiling, machine learning

INTRODUCTION

The mass of content available on the World-Wide Web raises important questions over its effective use. With largely unstructured pages authored by a massive range of people on a diverse range of topics, simple browsing has given way to filtering as the practical way to manage web-based information – and for most of us that means search engines.

Search engines are very effective at filtering pages that match explicit queries. Unfortunately, most people find articulating what they want extremely difficult, especially if forced to use a limited vocabulary such as keywords. The result is large lists of search results that contain a handful of useful pages, defeating the purpose of filtering in the first place.

Recommender Systems Can Help

Now people may find articulating what they want hard, but they are very good at recognizing it when they see it. This

insight has led to the utilization of relevance feedback, where people rate web pages as interesting or not interesting and the system tries to find pages that match the interesting examples (positive examples) and do not match interesting examples (negative examples). With the not interesting examples (negative examples), modern machine sufficient positive and negative examples, modern machine learning techniques can classify new pages with impressive accuracy.

Obtaining sufficient examples is difficult however, especially when trying to obtain negative examples. The problem with asking people for examples is that the cost, in terms of time and effort, of providing the examples generally outweighs the reward they will eventually receive. Negative examples are particularly unrewarding, since there could be many irrelevant items to any typical query.

Unobtrusive monitoring provides positive examples of what the user is looking for, without interfering with the users normal activity. Heuristics can also be applied to infer negative examples, although generally with less confidence. This idea has led to content-based recommender systems, which unobtrusively watch users browse the web, and recommend new pages that correlate with a user profile.

Another way to recommend pages is based on the ratings of other people who have seen the page before. Collaborative recommender systems do this by asking people to rate explicitly pages and then recommend new pages that similar users have rated highly. The problem with collaborative filtering is that there is no direct reward for providing examples since they only help other people. This leads to initial difficulties in obtaining a sufficient number of ratings for the system to be useful.

Hybrid systems, attempting to combine the advantages of content-based and collaborative recommender systems, have proved popular to-date. The feedback required for content-based recommendation is shared, allowing collaborative recommendation as well. A hybrid approach is used by our Quickstep recommender system.

This work follows the tradition of over 30 years of knowledge acquisition. Knowledge acquisition above the normal workflow is intrusive and counterproductive. We present a system with a low level of intrusiveness, driven by

people making explicit choices that reflect the real world to capture profiles.

The Problem Domain

As the trend to publish research papers on-line increases, researchers are increasingly using the web as their primary source of papers. Typical researchers need to know about new papers in their general field of interest, and older papers relating to their current work. In addition, researchers time is limited, as browsing competes with other tasks in the work place. It is this problem our Quickstep recommender system addresses.

Since researchers have their usual work to perform, unobtrusive monitoring methods are preferred else they will be reluctant to use the system. Also, very high recommendation accuracy is not critical as long as the system is deemed useful to them.

Evaluation of real world knowledge acquisition systems, as Shadbolt [21] discusses, is both tricky and complex. A lot of evaluations are performed with user log data (simulating real user activity) or with standard benchmark collections. Although these evaluations are useful, especially for technique comparison, they must be backed up by real world studies so we can see how the benchmark tests generalize to the real world setting. Similar problems are seen in the agent domain where, as Nwana [16] argues, it has yet to be conclusively demonstrated if people really benefit from such information systems.

This is why we have chosen a real problem upon which to evaluate our Quickstep recommender system.

User Profiling in Recommender Systems

User modelling is typically either knowledge-based or behaviour-based. Knowledge-based approaches engineer static models of users and dynamically match users to the closest model. Behaviour-based approaches use the users behaviour itself as a model, often using machine-learning techniques to discover useful patterns of behaviour. Kobsa [10] provides a good survey of user modelling techniques.

The typical user profiling approach for recommender systems is behaviour-based, using a binary model representing what users find interesting and uninteresting. Machine-learning techniques are then used to assess potential items of interest in respect to the binary model. There are a lot of effective machine learning algorithms based on two classes. Sebastiani [20] provides a good survey of current machine learning techniques and De Roure [5] a review of recommender systems.

Although more difficult than the binary case, we choose to use a multi-class behavioural model. This allows the classes to represent paper topics, and hence domain knowledge to be used when constructing the user profile. We thus bring together ideas from knowledge-based and behaviour-based modelling to address the problem domain.

Ontology Use and the World Wide Web

Ontologies are used both to structure the web, as in Yahoo's search space categorization, and to provide a common basis for understanding between systems, such as in the knowledge query modelling language (KQML). In-depth ontological representations are also seen in knowledge-based systems, which use relationships between web entities (bookmarks, web pages, page authors etc.) to infer facts about given situations.

We use an ontology to investigate how domain knowledge can help in the acquisition of user preferences.

Overview of the Quickstep System

Quickstep unobtrusively monitors user browsing behaviour via a proxy server, logging each URL browsed during normal work activity. A machine-learning algorithm classifies browsed URLs overnight, and saves each classified paper in a central paper store. Explicit feedback and browsed topics form the basis of the interest profile for each user.

Each day a set of recommendations is computed, based on correlations between user interest profiles and classified paper topics. Any feedback offered on these recommendations is recorded when the user looks at them.

Users can provide new examples of topics and correct paper classifications where wrong. In this way the training set improves over time.
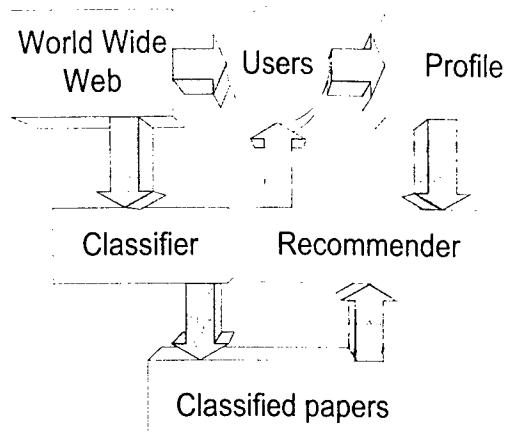
Figure 1 The Quickstep system

Empirical Evaluation

The current literature lacks many clear results as to the extent knowledge-based approaches assist real-world systems, where noisy data and differing user opinions exist. For this reason we decided to compare the use of an ontology against a simple flat list, to provide some empirical evidence as to the effectiveness of this approach.

Two experiments are detailed within this paper. The first has 14 subjects, all using the Quickstep system for a period

of 1.5 months. The second has 24 subjects, again over a period of 1.5 months.

Both experiments divide the subjects into two groups.

The first group uses a flat, extensible list of paper topics. Any new examples, added via explicit feedback, use this flat list to select from. The users are free to add to the list as needed.

The second group uses a fixed size topic ontology (based on the dmoz open directory project hierarchy [6]). Topics are selected from a hierarchical list based on the ontology. Interest profiles of this group take into account the super classes of any browsed topics.

Performance metrics are measured over the duration of the trial, and thus the effectiveness of both groups compared.

APPROACH

The Quickstep System

Quickstep is a hybrid recommendation system, combining both content-based and collaborative filtering techniques. Since both web pages and user interests are dynamic in nature, catalogues, rule-bases and static user profiles would quickly become out of date. A recommender system approach thus appeared well suited to our problem.

Explicit feedback on browsed papers would be too intrusive, so unobtrusive monitoring is used providing positive examples of pages the user typically browses. Many users will be using the system at once, so it is sensible to share user interest feedback and maintain a common pool of labelled example papers (provided by the users as examples of particular paper topics).

Since there are positive examples of the kind of papers users are interested in, we have a labelled training set. This is ideal for supervised learning techniques, which require each training example to have a label (the labels are then used as classification classes). The alternative, unsupervised learning, is inherently less accurate since it must compute likely labels before classification (e.g. clustering techniques). We shall use a term vector representation, common in machine learning, to represent a research paper. A term vector is a list of word weights, derived from the frequency that the word appears within the paper.

We could have used a binary classification approach, with classes for "interesting" and "not interesting". This would have led to profiles consisting of two term vectors, one representing the kind of thing the user is interested in (computed from the positive examples) and the other what the user is not interested in (computed from the negative examples). Recommendations would be those page vectors that are most similar to the interesting class vector. The binary case is the simplest class representation, and consequently produces the best classification results when compared with multi-class methods.

One problem with such a representation is that the explicit knowledge of which topics the user is interested in is lost, making it hard to benefit from any prior knowledge we may know about the domain (such as the paper topics). With Quickstep, we have chosen a multi-class representation, with each class representing a research paper topic. This allows profiles that consist of a human understandable list of topics. The classifier assigns each paper a class based on which class vector it is most similar to. Recommendations are selected from papers classified as belonging to a topic of interest.

The profile itself is computed from the correlation between browsed papers and paper topics. This correlation leads to a topic interest history, and a simple time-decay function allows current topics to be computed.

Details of Specific Techniques Used

Research Paper Representation

Research papers are represented as term vectors, with term frequency / total number of terms used for a terms weight. To reduce the dimensionality of the vectors, frequencies less than 2 are removed, standard Porter stemming [18] applied to remove word suffixes and the SMART [22] stop list used to remove common words such as "the". These measures are commonly used in information systems; van Rijsbergen [24] and Harman [9] provide a good discussion of these issues.

Vectors with 10-15,000 terms were used in the trials along with training set sizes of about 200 vectors. Had we needed more dimensionality reduction, the popular term frequency-inverse document frequency (TF-IDF) weighting could be used (term weights below a threshold being removed) or latent semantic indexing (LSI).

Only Postscript and PDF formats (and compressed formats) are supported, to avoid noisy HTML pages. This makes classification easier, at the expense of HTML only papers.

Research Paper Classification

The classification requirements are for a multi-class learning algorithm learning from a multi-labelled training set. To learn from a training set, inductive learning is required. There are quite a few inductive learning techniques to choose from, including information theoretic ones (e.g. Rocchio classifier), neural networks (e.g. backpropagation), instance-based methods (e.g. nearest neighbour), rule learners (e.g. RIPPER), decision trees (e.g. C4.5) and probabilistic classifiers (e.g. naive Bayes).

Multiple classifier techniques such as boosting [7] exist as well, and have been shown to enhance the performance of individual classifiers.

After reviewing and testing many of the above options, we decided to use a nearest neighbour technique. The nearest neighbour approach is well suited to our problem, since the training set must grow over time and consists of multi-class examples. Nearest neighbour algorithms also degrade well, with the next closest match being reported if the correct one is not found. The IBk algorithm [1] we chose outperformed

naive Bayes and a J48 decision tree in our tests. We also use the boosting technique AdaBoostM1 [7], which works well for multi-class problems if the boosted classifier is strong enough. We found that boosting always improved the base classifiers performance in our tests.

Nearest neighbour algorithms represent instances of documents as term vectors within a term vector space. Proximity of vectors within this term vector space indicates similarity. To classify a new paper, the vector distance from each example instance is calculated, and the closest neighbours returned as the most likely classes. Inverse distance weighting is used to decrease the likelihood of choosing distant neighbours.

AdaBoostM1 extends AdaBoost to handle multi-class cases since AdaBoost itself is a binary classifier. AdaBoostM1 repeatedly runs a weak learning algorithm (in this case the IBk classifier) for a number of iterations over various parts of the training set. The classifiers produced (specialized for particular classes) are combined to form a single composite classifier at the end.

Profiling Algorithm
The profiling algorithm performs correlation between the paper topic classifications and user browsing logs. Whenever a research paper is browsed that has a classified topic, it accumulates an interest score for that topic. Explicit feedback on recommendations also accumulates interest values for topics. The current interest of a topic is computed using the inverse time weighting algorithm below, applied to the user feedback instances.

$$\text{Topic interest} = \sum_{1..\text{no of instances}}^{n} \text{Interest value}(n) / \text{days old}(n)$$

Interest values   Paper browsed = 1
                              Recommendation followed = 2
                              Topic rated interesting = 10
                              Topic rated not interesting = -10

The profile for each user consists of a list of topics and the current interest values computed for them (see below). The interest value weighting was chosen to provide sufficient weight for an explicit feedback instance to dominate for about a week, but after that browsed URL's would again become dominant. In this way, the profile will adapt to changing user interests as the trial progresses.

Profile = (<user>,<topic>,<topic interest value>)*

e.g.   ((someone,hypertext,-2.4)
      (someone,agents,6.5)
      (someone,machine learning,1.33))

If the user is using the ontology based set of topics, all super classes gain a share when a topic receives some interest. The immediate super class receives 50% the main topics value. The next super class receives 25% and so on until the most general topic in the is-a hierarchy is reached. In this way, general topics are included in the profile rather than just the most specific ones, producing a more rounded profile.

Recommendation Algorithm
Recommendations are formulated from a correlation between the users current topics of interest and papers classified as belonging to those topics. A paper is only recommended if it does not appear in the users browsed URL log, ensuring that recommendations have not been seen before. For each user, the top three interesting topics are selected with 10 recommendations made in total (making a 4/3/3 split of recommendations). Papers are ranked in order of the recommendation confidence before being presented to the user.

Recommendation confidence = classification confidence * topic interest value

The classification confidence is computed from the AdaBoostM1 algorithm's class probability value for that paper (somewhere between 0 and 1).

Research Paper Topic Ontology
The research paper topic ontology is based on the dmoz [6] taxonomy of computer science topics. It is an is-a hierarchy of paper topics, up to 4 levels deep (e.g. an "interface agents" paper is-a "agents" paper). Pre-trial interviews formed the basis of which additional topics would be required. An expert review by two domain experts validated the ontology for correctness before use in the trials.

Feedback and the Quickstep Interface
Recommendations are presented to the user via a browser web page. The web page applet loads the current recommendation set and records any feedback the user provides. Research papers can be jumped to, opening a new browser window to display the paper URL. If the user likes/dislikes the paper topic, the interest feedback combo-box allows "interested" or "not interested" to replace the default "no comment". Finally, the topic of each paper can be changed by clicking on the topic and selecting a new one from a popup menu. The ontology group has a hierarchical popup menu; the flat list group has a single level popup menu.
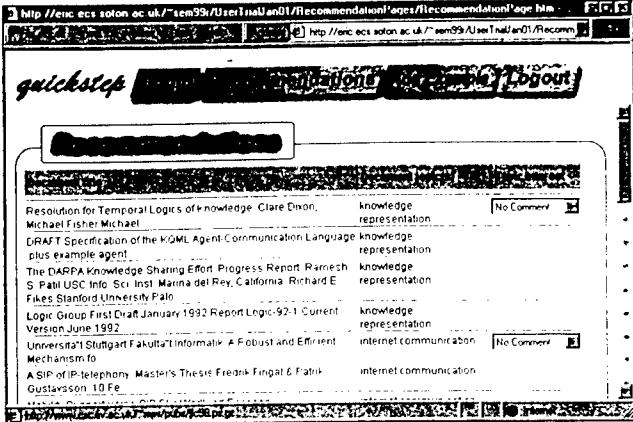
Figure 2 Quickstep's web-based interface

New examples can be added via the interface, with users providing a paper URL and a topic label. These are added to the groups training set, allowing users to teach the system new topics or improve classification of old ones.

All feedback is stored in log files, ready for the profile builders run. The feedback logs are also used as the primary metric for evaluation. Interest feedback, topic corrections and jumps to recommended papers are all recorded.

EVALUATION

Details of the Two Trials

Two trials were conducted to assess empirically both the overall effectiveness of the Quickstep recommender system and to quantify the effect made by use of the ontology.

The first trial used 14 subjects, consisting of researchers from the IAM research laboratory. A mixture of 2nd year postgraduates up to professors was taken, all using the Quickstep system for a duration of 1.5 months.

The second trial used 24 subjects, 14 from the first trial and 10 more 1st year postgraduates, and lasted for 1.5 months. Some minor interface improvements were made to make the feedback options less confusing.

The pre-trial interview obtained details from subjects such as area of interest and expected frequency of browser use.

The purpose of the two trials was to compare a group of users using an ontology labelling strategy with a group of users using a flat list labelling strategy. Subject selection for the two groups balanced the groups as much as possible, evening out topics of interest, browser use and research experience (in that order of importance). Both groups had the same number of subjects in them (7 each for the pilot trial, 12 each for the main trial).

In the first trial, a bootstrap of 103 example papers covering 17 topics was used. The bootstrap examples were obtained from bookmarks requested during the pre-trial interview.

In the second trial, a bootstrap of 135 example papers covering 23 topics was used. The bootstrap training set was updated to include examples from the final training sets of

the first trial. The first trials classified papers were also kept, allowing a bigger initial collection of papers from which to recommend in the second trial.

Both groups had their own separate training set of examples, which diverged in content as the trial progressed. The classifier was run twice for each research paper, classifying once with the flat list groups training set and once with the ontology groups training set. The classifier algorithm was identical for both groups; only the training set changed.

The system interface used by both groups was identical, except for the popup menu for choosing paper topics. The ontology group had a hierarchical menu (using the ontology); the flat list group had a single layer menu.

The system recorded the times the user declared an interest in a topic (by selecting "interesting" or "not interesting"), jumps to recommended papers and corrections to the topics of recommended papers. These feedback events were date stamped and recorded in a log file for later analysis, along with a log of all recommendations made. Feedback recording was performed automatically by the system, whenever the subjects looked at their recommendations.

Experimental Data

Since feedback only occurs when subjects check their recommendations, the data collected occurs at irregular dates over the duration of the trial. Cumulative frequency of feedback events is computed over the period of the trial, allowing trends to be seen as they develop during the trial. Since the total number of jumps and topics differ between the two groups, the figures presented are normalized by dividing by the number of topics (or recommendations) up to that date. This avoids bias towards the group that provided feedback most frequently.

Figure 3 shows the topic interest feedback results. Topic interest feedback is where the user comments on a recommended topic, declaring it "interesting" or "not interesting". If no feedback is offered, the result is "no comment".

Topic interest feedback is an indication of the accuracy of the current profile. When a recommended topic is correct for a period of time, the user will tend to become content with it and stop rating it as "interesting". On the other hand, an uninteresting topic is likely to always attract a "not interesting" rating. Good topics are defined as either "no comment" or "interesting" topics. The cumulative frequency figures are presented as a ratio of the total number of topics recommended. The not interesting ratio (bad topics) can be computed from these figures by subtracting the good topic values from 1.

The ontology groups have a 7 and 15% higher topic acceptance. In addition to this trend, the first trial ratios are about 10% lower than the second trial ratios.

Figure 4 shows the jump feedback results. Jump feedback is where the user jumps to a recommended paper by opening it via the web browser. Jumps are correlated with topic interest feedback, so a good jump is a jump to a paper on a good topic. Jump feedback is an indication of the quality of the recommendations being made as well as the accuracy of the profile. The cumulative frequency figures are presented as a ratio of the total number of recommendations made.

There is a small 1% improvement in good jumps by the ontology group. Both trials show between 8-10% of recommendations leading to good jumps.

Figure 5 shows the topic correction results. Topic corrections are where the user corrects the topic of a recommended paper by providing a new one. A topic correction will add to or modify a groups training set so that the classification for that group will improve. The number of corrections made is an indication of classifier accuracy. The cumulative frequency figures are presented as a ratio of the total number of recommended papers seen.

Although the flat list group has more corrections, the difference is only by about 1%. A clearer trend is for the flat list group corrections to peak around 10-20 days into the trial, and for both groups to improve as time goes on.
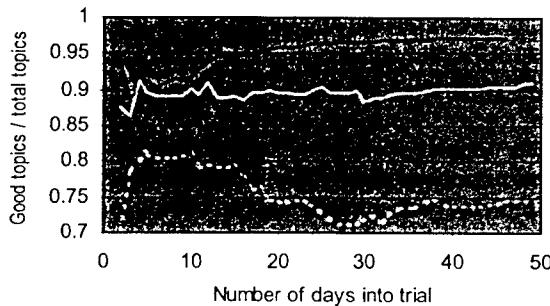
Figure 3 Ratio of good topics / total topics
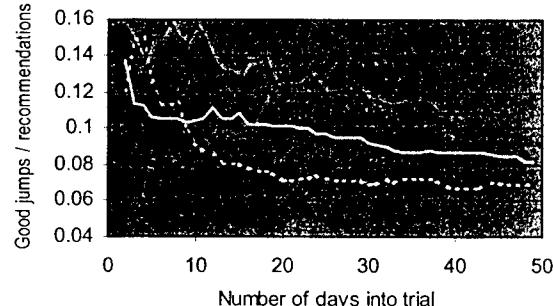
Figure 4 Ratio of good jumps / total recommendations
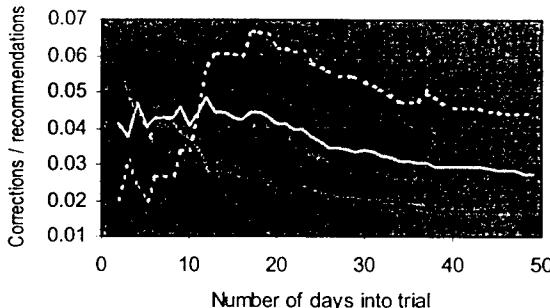
Figure 5 Ratio of topic corrections / total recommendations

A cross-validation test was run on each group's final training set, to assess the precision and recall of the classifier using those training sets. The results are shown in table 1.

Group (trial)	Precision	Recall	Classes
Trial 1, Ontology	0.484	0.903	27
Trial 1, Flat list	0.52	1.0	25
Trial 2, Ontology	0.457	0.888	32
Trial 2, Flat list	0.456	0.972	32

Table 1 Classifier recall and precision upon trial completion

Discussion of Trends Seen in the Experimental Data
From the experimental data of both trials, several suggestive trends are apparent. The initial ratios of good topics were lower than the final ratios, reflecting the time it takes for enough log information to be accumulated to let the profile settle down. The ontology users were 7-15% happier overall with the topics suggested to them.

Our hypothesis for the ontology group's apparently superior performance is that the is-a hierarchy produces a rounder, more complete profile by including general super class topics when a specific topic is browsed by a user. This in turn helps the profiler to discover a broad range of interests, rather than just latching onto one correct topic.

The first trial showed fewer good topics than the second trial (about a 10% difference seen by both groups). We think this is because of interface improvements made for the second trial, where the topic feedback interface was made less confusing. Subjects were sometimes rating interesting topics as not interesting if the paper quality was poor. As there are more poor quality papers than good quality ones, this introduced a bias to not interesting topic feedback resulting in a lower overall ratio.

About 10% of recommendations led to good jumps. Since 10 recommendations were given to the users at a time, on average one good jump was made from each set of recommendations received. As with the topic feedback, the ontology group again was marginally superior but only by a 1% margin. We think this smaller difference is due to people having time to follow only 1 or 2 recommendations. Thus, although the ontology group has more good topics, only the top topic of the three recommended will really be looked at; the result is a smaller difference between the good jumps made and the good topics seen.

The flat list group has a poor correction / recommendation ratio 10-20 days into the trial. We think this is due to new topics being added to the system. Most new topics were added after the users became familiar with the system, and know which topics they feel are missing. The familiarization process appeared to take about 10 days. The classification accuracy of these new topics is poor until enough examples have been entered, typically after another 10 days.

The ontology group has about 1% fewer corrections for both trials. This is small difference may indicate the utility of imposing a uniform conceptual model of paper topics on the subjects (by using the common topic hierarchy). Classifying papers is a subjective process, and will surely be helped if people have similar ideas as to where topics fit in a groups overall classification scheme.

These preliminary results need to be extended so as to enable the application of more rigorous statistical analysis. Nevertheless, we believe the trend in the data to be encouraging as to the utility of ontologies in recommender systems.

When compared with other published systems, the classification accuracy figures are similar, if on the low side (primarily because we use multi-class classification). Nearest neighbour systems such as NewsDude [3] and Personal Webwatcher [14] report 60-90% classification accuracy based on binary classification. The higher figures tend to be seen with benchmark document collections, not real-world data. NewsWeeder [12] reports 40-60% classification accuracy using real user browsing data from two users over a period of time, so this would be the best comparison. If the number of classes we classify is taken into consideration, our system compares well.

Multi-class classification is not normally applied to recommender systems making direct comparison of similar systems difficult. We would have liked to compare the usefulness of our recommender to that of other systems, but the lack of published experimental data of this kind means we can only usefully compare classification accuracy.

CONCLUSIONS

Most recommender systems use a simple binary class approach, using a user profile of what is interesting or not interesting to the user. The Quickstep recommender system uses a multi-class approach, allowing a profile in terms of domain concepts (research paper topics) to be built. The multi-class classification is less accurate than other binary classification systems, but allows class specific feedback and the use of domain knowledge (via an is-a hierarchy) to enhance the profiling process.

Two experiments are performed in a real work setting, using 14 and 24 subjects over a period of 1.5 months. The results suggest how using an ontology in the profiling process results in superior performance over using a flat list of topics. The ontology users tended to have more "rounder" profiles, including more general topics of interest that were not directly suggested. This increased the accuracy of the profiles, and hence usefulness of the recommendations.

The overall performance compares reasonably with other recommender systems.

Related Work

Collaborative recommender systems utilize user ratings to recommend items liked by similar people. Examples of collaborative filtering are PHOAKS [23], which recommends web links mentioned in newsgroups and Group Lens [11], which recommends newsgroup articles.

Content-based recommender systems recommend items with similar content to things the user has liked before. Examples of content-based recommendation are Fab [2], which recommends web pages and ELFI [19], which recommends funding information from a database.

Personal web-based agents such as Letizia [13], Syskill & Webert [17] and Personal Webwatcher [14] track the users browsing and formulate user profiles. Profiles are constructed from positive and negative examples of interest, obtained from explicit feedback or heuristics analysing browsing behaviour. They then suggest which links are worth following from the current web page by recommending page links most similar to the users profile.

News filtering agents such as NewsWeeder [12] and News Dude [3] recommend news stories based on content similarity to previously rated examples.

Systems such as CiteSeer [4] use content-based similarity matching to help search for interesting research papers within a digital library. Ontologies are also used to improve content-based search, as seen in OntoSeek [8].

Mladenic [15] provides a good survey of text-learning and agent systems, including content-based and collaborative approaches.

Future Direction of Work

The next step for this work is to run more trials and perform rigorous statistical analysis on the results. As the subjects increase in number, we can become increasingly confident of the power of the effects we are seeing.

Paper quality ratings will be elicited from users, so once an interesting topic has been discovered, good quality papers can be recommended before poorer quality papers.

The idea of building a profile that is understandable by the users could be extended to actually visualizing the knowledge contained within it. This will allow the recommender to engage the user in a dialogue about what exactly they are interested in. The knowledge elicited from this dialogue should allow further improvements to the recommendations made. Additionally, visualizing the profile knowledge will allow users to build a better conceptual model of the system, helping to engender a feeling of control and eventually trust in the system.

ACKNOWLEDGEMENTS

REFERENCES

1. Aha, D. Kibler, D. Albert, M. Instance-based learning algorithms, Machine Learning, 6:37-66, 1991

2. Balabanovi, M. Shoham, Y. Fab: content-based, collaborative recommendation, Communications of the ACM Volume 40, No. 3 (Mar. 1997)

3. Billsus, D. Pazzani, M. A Personal News Agent that Talks, Learns and Explains, Proceedings of the Third International Conference on Autonomous Agents (Agents '99), Seattle, Washington, 1999

4. Bollacker, K.D. Lawrence, S. Giles, C.L. CiteSeer: An Autonomous Web Agent for Automatic Retrieval and Identification of Interesting Publications, Proceedings of the Second International Conference on Autonomous Agents, Minneapolis MN, USA, 1998

5. De Roure, D. Hall, W. Reich, S. Hill, G. Pikrakis, A. Stairmand, M. MEMOIR – an open framework for enhanced navigation of distributed information, Information Processing and Management, 37, 53-74, 2001

6. dmoz open directory project, Project home page http://dmoz.org/

7. Freund, Y. Schapire, R.E. Experiments with a New Boosting Algorithm, Proceedings of the Thirteenth International Conference on Machine Learning, 1996

8. Guarino, N. Masolo, C. Vetere, G. OntoSeek: Content-Based Access to the Web, IEEE Intelligent Systems, Vol. 14, No. 3, May/June 1999

9. Harman, D. An Experimental Study of Factors Important in Document Ranking. Proceedings of 1986 ACM conference on Research and development in information retrieval, September 1986, Pisa Italy

10. Kobsa, A. User Modeling in Dialog Systems: Potentials and Hazards, AI & Society: The Journal of Human and Machine Intelligence, 4:214-231, 1990

11. Konstan, J.A. Miller, B.N. Maltz, D. Herlocker, J.L. Gordon, L.R. Riedl, J. GroupLens: applying collaborative filtering to Usenet news, Communications of the ACM Volume 40, No. 3 (Mar. 1997)

12. Lang, K. NewsWeeder: Learning to Filter NetNews, ICML'95 Conference Proceedings, Tahoe City, CA, July 1995, pp 331-339

13. Lieberman, H. Letizia: An Agent That Assists Web Browsing, Proceedings of the 1995 International Joint Conference on Artificial Intelligence, Montreal, Canada, August 1995

14. Mladenic, D. Personal WebWatcher: Implementation and Design, Technical Report IJS-DP-7472, Department of Intelligent Systems, J.Stefan Institute, Slovenia, 1996

15. Mladenic, D. Text-Learning and Related Intelligent Agents: A Survey, IEEE Intelligent Systems, Vol. 14, No. 4, July/August 1999

16. Nwana, H.S. Software agents: an overview. The Knowledge Engineering Review, Vol. 11:3, 1996, 205-244.

17. Pazzani, M. Muramatsu J. Billsus, D. Syskill & Webert: Identifying interesting web sites, Proceedings of the National Conference on Artificial Intelligence, Portland, Oregon, 1996

18. Porter, M. An algorithm for suffix stripping, Program 14 (3), July 1980, pp. 130-137

19. Schwab, I. Pohl, W. Koychev, I. Learning to Recommend from Positive Evidence, Proceedings of Intelligent User Interfaces 2000, ACM Press, pp 241-247

20. Sebastiani, F. Machine Learning in Automated Text Categorization. Consiglio Nazionale delle Ricerche, Italy.

21. Shadbolt, N. O'Hara, K. Crow, L. The experimental evaluation of knowledge acquisition techniques and methods: history, problems and new directions, International Journal of Human-Computer Studies (1999) 51, pp 729-755

22. SMART Staff, User's Manual for the SMART Information Retrieval System, Technical Report 71-95, Revised April 1974, Cornell University (1974)

23. Terveen, L. Hill, W. Amento, B. McDonald, D. Creter, J. PHOAKS: a system for sharing recommendations, Communications of the ACM Volume 40, No. 3 (Mar. 1997)

24. van Rijsbergen, C.J. Information retrieval. Department of Computing Science, University of Glasgow.

Human Directability of Agents

Karen L. Myers   David N. Morley

Artificial Intelligence Center
SRI International
333 Ravenswood Ave.
Menlo Park, CA 94025
myers@ai.sri.com   morley@ai.sri.com

Abstract

Many potential applications for agent technology require humans and agents to work together in order to achieve complex tasks effectively. In contrast, much of the work in the agents community to date has focused on technologies for fully autonomous agent systems. This paper presents a framework for the *directability* of agents, in which a human supervisor can define policies to influence agent activities at execution time. The framework focuses on the concepts of *adjustable autonomy* for agents (i.e., varying the degree to which agents make decisions without human intervention) and *strategy preference* (i.e., recommending how agents should accomplish assigned task). The directability framework has been implemented within a PRS environment, and applied to a multiagent intelligence-gathering domain.

Keywords

Advisable Systems, Agents, Mixed-initiative Control

INTRODUCTION

The technical and public press are filled these days with visions of a not-too-distant future in which humans rely on software and hardware agents to assist with problem solving in environments both physical (e.g., smart offices, smart homes) and virtual (e.g., the Internet). The notion of *delegation* plays a central role in these visions, with humans offloading responsibilities to agents that can perform activities in their place.

Successful delegation requires more than the assignment of tasks. A good manager generally provides directions to a subordinate so that tasks are performed to his or her liking. To ensure effectiveness, the manager will monitor the progress of the subordinates, occasionally interrupting to provide advice or to resolve problems.

The agents research community has, for the most part, focused on the mechanics of building autonomous agents and

techniques for communication and coordination among agents. In contrast, little attention has been paid to supporting human interactions with agents of the type required for extended problem-solving sessions. Most agent frameworks fall at the extremes of the interaction spectrum, either assuming full automation by the agents with no means for user involvement, or requiring human intervention at each step along the way. Recently, however, there has been increased interest in agent systems designed specifically to support interaction with humans (e.g., [2, 3, 5, 16]).

We are developing a framework, called Taskable Reactive Agent Communities (TRAC), that supports directability of a team of agents by a human supervisor. Within TRAC, the human assigns tasks to agents along with guidance that imposes boundaries on agent behavior. By adding, deleting, or modifying guidance at execution time, the human can manage agent activity at a level of involvement that suits his or her needs. In essence, our approach can be viewed as form of process management technology that enables human control of agent communities.

A key issue in developing technology to support agent directability is determining the types of guidance to be provided. This paper focuses on guidance for *adjustable agent autonomy* and *strategy preferences*. Guidance for adjustable autonomy enables a supervisor to vary the degree to which agents can make decisions without human intervention. Guidance for strategy preferences constitutes recommendations on how agents should accomplish assigned tasks.

The main contributions of this paper are the characterization of these forms of guidance, presentation of a formal language for representing such guidance, the description of a semantic model for satisfaction of such guidance by an agent, and techniques for enforcing such guidance during agent operation.

Effective delegation and management by a human supervisor also requires visibility into ongoing agent operations. Although not described in this paper, the TRAC framework includes a capability for *customizable reporting* that enables a supervisor to tailor the amount, type, and frequency of information produced by agents to meet his evolving needs.

Details can be found in [12].

The paper begins with a description of our underlying model for agents. From there, we present an informal characterization of guidance for adjustable autonomy and strategy preferences. Next, we describe a multiagent system, called TIGER, which instantiates the TRAC approach to directability for the application of multiagent intelligence gathering in a simulated natural disaster scenario. We use TIGER to provide concrete examples of the directability concepts throughout the paper. Following this description, we present our representation for guidance and describe both our semantic model for guidance satisfaction and techniques for guidance enforcement. The paper concludes with a discussion of related work and directions for further research.

AGENT MODEL

We adopt a typical Belief-Desire-Intention (BDI) model of agency in the style of [14], whereby an agent undertakes actions to address its desires, relative to its current beliefs about the operating environment. BDI agents are so-called due to the three components of their "mental state": *beliefs* that the agent has about the state of the world, *desires* to be achieved, and *intentions* corresponding to actions that the agent has adopted to achieve its desires.

Each agent has a library of *plans* that define the range of activities that an agent can perform to respond to events or to achieve assigned tasks; our plan model is based on [17]. Plans are parameterized templates of activities that may require variable instantiations to apply to a particular situation. The *cue* of a plan specifies a stimulus that activates the plan, either a new goal or a change in the agent's beliefs. *Preconditions* associated with plans define gating constraints that must be satisfied in order for a plan to be applied. A plan is said to be *applicable* to a world change (e.g., new goal or belief change event) when the plan cue matches the stimulus, and the plan preconditions are satisfied in the current world state. The *body* of a plan specifies how to respond to the stimulus, in terms of actions to perform and subgoals to achieve.

An agent's plan library will generally contain a range of plans describing alternative responses to posted goals or events. Sets of these plans may be *operationally equivalent* (i.e., they share the same cue and preconditions) but differ in the approach that they embody. Some form of meta-control policy can be defined to select among such alternatives, should the need arise.

A BDI interpreter runs a continuous *sense-decide-act* loop. In each iteration the agent executes a single step of one of its intentions on the basis of its current beliefs about the state of the world. This entails performing actions, adopting new goals to achieve, updating its set of beliefs, and updating the set of current intentions. Within this paradigm, agents make three classes of decisions:

D1 *whether to respond to new goals and events*
D2 *how to select among multiple applicable plans*
D3 *how to select instantiations for plan variables.*

Our directability framework assumes that agents are capable of fully autonomous operation. More concretely, an agent's plan library covers the range of activities required to perform its assigned tasks. This assumption means that agents do not depend on the human supervisor to provide knowledge for task execution. Within this setting, guidance provides customization of agent behavior to suit the preferences of the human supervisor. In many applications, such guidance will enable superior performance, given that few plan libraries will reflect the full the experience, breadth of knowledge, and reasoning capabilities that a human supervisor can bring to the decision-making process.

TRAC FRAMEWORK FOR AGENT DIRECTABILITY

Our model of agent directability focuses on general and task-specific policies to influence the activities undertaken by agents in their execution of assigned tasks. In particular, we emphasize the areas of (a) adjustable levels of agent autonomy and (b) strategy preferences that describe approaches to be used by an individual agent in executing assigned tasks. Given the need to adjust to dynamic environments, these guidance policies can be defined and modified at any point during agent execution.

Adjustable Autonomy

We define the autonomy of an agent to be the extent to which it is allowed to make decisions (specifically, D1 – D3) on its own. In situations where activities are routine and decisions straightforward, a human may be content to delegate all problem-solving responsibility to an agent. However, in situations where missteps could have severe consequences, the degree of autonomy of an individual agent should necessarily be controllable by a human.

Because we are interested in domains where agents will need to operate with high degrees of autonomy, we assume a *permissive* environment: unless stated otherwise, agents are allowed to operate independent of human interaction. Our approach allows the human to adjust the scope of operations that can be undertaken by an agent on its own terms, focusing on the notions of *permission requirements* for action execution and *consultation requirements* for decision making.

Permission Requirements Permission requirements declare conditions under which an agent must elicit authorization from the human supervisor before executing actions. For example, the directive "Obtain permission before abandoning survey tasks with Priority > 3" imposes the constraint that an agent request approval from its supervisor to abandon a certain class of tasks.

Consultation Requirements Consultation requirements designate a class of agent decisions that should be deferred to the human supervisor. These decisions relate to the selection of

values for variable instantiation, for example, "Consult when selecting locations for staging bases."

Our model of permission and consultation requirements, like earlier work on authority models, provides a mechanism to block performance of certain actions by an agent. However, authority models are generally static (e.g., the *levels of autonomy* in [2]) and often derived from organizational structures. In contrast, our approach provides a rich language for expressing permission and consultation policies, which can vary throughout a problem-solving session.

Strategy Preference

Strategy preferences express recommendations on how an agent should accomplish tasks. These preferences could indicate specific plans to employ or restrictions on plans that should not be employed, as well as constraints on how plan variables can be instantiated.

For example, the directive "Try contacting Nongovernmental Organizations for information before sending vehicles to towns on the west coast" expresses a preference for selecting among operationally equivalent plans. On the other hand, the directive "Only use helicopters for survey tasks in sectors that are expected to be inaccessible by truck for more than 1 week" restricts the choice of resource type for instantiating certain plan variables.

THE TIGER SYSTEM

We have developed a prototype implementation of our TRAC framework for agent guidance on top of the Procedural Reasoning System (PRS) [7]. The TRAC implementation has been used as the basis for a demonstration system called TIGER (TRAC Intelligence Gathering and Emergency Response) that serves as a testbed for exploring our ideas on agent directability. Within TIGER, a human supervisor can delegate tasks to agents while providing guidance to control their runtime behavior.

TIGER Functionality

TIGER provides control over a collection of simulated physical assets (trucks and aircraft), each embodied as a separate agent. These physical assets can be tasked to perform a range of actions related to intelligence gathering, and to provide assistance with eventualities such as medical emergencies, evacuations, and infrastructure repair.

TIGER serves as part of a *disaster response team* whose objective is to provide humanitarian relief in the wake of a natural disaster. Other organizations within the team provide logistics (e.g., supplies distribution), operations (e.g., repair of infrastructure), and medical services. These organizations have their own physical assets (trucks and aircraft) available for their use. As would be expected, these organizations need to share information and resources to perform their functions effectively. A human commander oversees operations, dynamically tasking organizations to implement
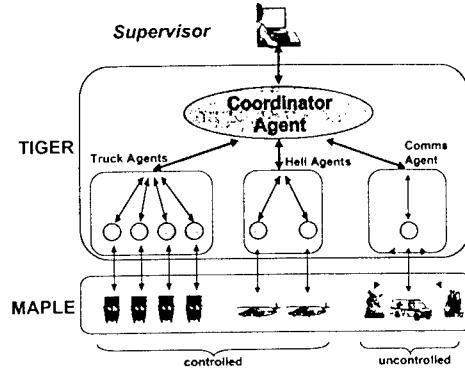
Figure 1. TIGER Architecture

the relief process.[1]

The primary role for TIGER is to gather information in response to requests initiated by other members of the disaster response team or the supervisor. These requests can result in tasks to seek out information such as the current state of infrastructure (roads, bridges) in designated regions, or to collect supply requirements (medical, food, water, shelter) of designated population centers within impacted regions. There can also be requests to be informed of key events (such as medical emergencies) as they become known.

A secondary role is to respond to certain types of unexpected events (e.g., participating in evacuations, assisting with medical emergencies). Thus, TIGER agents must incorporate reactive capabilities that balance responsiveness with ongoing goal attainment.

The scope and complexity of the intelligence-gathering operations within the disaster relief context preclude step-by-step management of agent operations by a human commander. However, effective coordination of the available assets requires human supervision. As such, this domain provides an excellent example of an application that will benefit from technology for agent directability.

Agent Community Organization

Figure 1 displays the organization of agents within TIGER. The system has at its disposal a collection of simulated *physical agents* (trucks and helicopters) that can be used to gather information and respond to emergencies. In addition, there is a set of simulated *communications agents* (other relief organizations, nongovernment organizations, local officials) that can be consulted to obtain information. TIGER contains a

---

[1] The system operates within a testbed that simulates a major hurricane in Central America; the testbed is built on the MAPLE system (http://www.cs.cmu.edu/~maple/).

110

separate controller for each of the physical agents, as well as a communications manager for interacting with the various simulated communications agents. We refer to these controller agents as the *task execution agents* within TIGER, because they instigate and manage the activities required to perform assigned tasks.

The *coordinator agent* provides global management of tasks within the community, acting as a mediator between the human supervisor and the task execution agents. It also manages interactions with members of the disaster response team who request information (i.e., its *information clients*).

Tasking Model

The TIGER coordinator agent places incoming task requests into a pool of waiting tasks. It also maintains a pool of currently unallocated agents. The coordinator agent matches a waiting task with an unallocated agent based on properties of the task, the available agents, and current knowledge about the state of the roads and bridges. Task properties include *location*, *priority* (an integer from 0 to 10), *type* (e.g., survey, rescue), *deadline* (for completing the task), and *status* (e.g., pending, completed, failed). The agent properties include *agent type* (helicopter or truck) and *location*.

Task management constitutes a major component of an execution agent's decision-making process. An execution agent must determine what to do if, while executing one task, the coordinator agent passes it a second task. It must also decide when to drop tasks that are not progressing well in favor of new tasks with higher potential for success.

For simplicity, we limit each task execution agent to at most one active task at any point in time. Agents may also have pending tasks (which they intend to undertake) and preempted tasks (which were begun but put aside for higher-priority tasks). Tasks are assigned to individual agents and do not require coordination with other agents for their completion.

Unexpected events, such as a medical emergency, may require immediate response. Events are characterized by the properties *location*, *time* (of the event), *severity* (an integer 0 to 10), *number of people affected*, and *type* (e.g., evacuation, medical). Rather than creating a new task for the task pool, the coordinator agent selects an appropriate task execution agent to deal directly with each event.

These characteristics of tasking simplify the decision process for what an execution agent should do when it receives a task request. The agent can choose among several combinations of actions, including *ignore* the event, *adopt* a new task to respond to the event, *abandon* the current active task, *transfer* the task to another agent, or *postpone* the current task until the new task is completed. Alternatives in the agent's plan library encode each of these choices.

REPRESENTATION OF GUIDANCE

Our language for representing agent guidance builds on three main concepts: the underlying *agent domain theory*, a *domain metatheory*, and the connectives of first-order logic. Using these elements, we develop the main concepts underlying our model of agent guidance. These consist of an *activity specification* for describing abstract classes of action, a *desire specification* for describing abstract classes of goals, and an *agent context* for describing situations in which guidance applies.

Domain Metatheory

A standard domain theory for an agent consists of four types of basic element: *individuals* corresponding to real or abstract objects in the domain, *relations* that describe characteristics of the world, *goals* that an agent may adopt, and *plans* that describe available means for achieving goals.

The *domain metatheory* provides an abstracted characterization of elements of the domain theory that highlights key semantic differences. As discussed in [11], a metatheory can yield a rich vocabulary for describing activity, thus providing a powerful basis for supporting user communication. The main concepts within our metatheory for agent guidance are *features* and *roles* (similar in spirit to those of [10]) defined for agent plans and goals.

Consider first plans. A *plan feature* designates an attribute of interest for a plan that distinguishes it from other plans that could be applied to the same task. For example, among plans for route determination, there may be one that is OPTIMAL but SLOW with a second that is HEURISTIC but FAST; each of these attributes could be modeled as a feature. Although the two plans are operationally equivalent (i.e., same cue and preconditions), their intrinsic characteristics differ significantly. Features provide the means to distinguish between such operationally equivalent alternatives.

A *plan role* describes a capacity in which a domain object is used within a plan; it maps to an individual variable within a plan. For instance, a route determination plan may contain variables location.1 and location.2, with the former corresponding to the START and the latter the DESTINATION.

In analogous fashion, roles and features can also be defined for goals. For example, a goal of informing another party of task progress may have a COMMUNICATION feature and RECIPIENT role associated with it. These metatheoretic constructs can be used to specify the class of goals that involve communicating with the commander.

Activity Specification

An *activity specification* characterizes an abstract class of plan instances for an agent. Our domain metatheory provides the basis for defining an activity specification, in terms of a set of required and prohibited features on a plan, as well as constraints on the way in which plan roles are filled.

Definition 1 (Activity Specification) *An* activity specification $\alpha = \langle \mathcal{F}^+, \mathcal{F}^-, \mathcal{R}, \phi \rangle$ *consists of*

- *a set of* required features $\mathcal{F}^+$
- *a set of* prohibited features $\mathcal{F}^-$
- *a set of* roles $\mathcal{R} = [R_1, \ldots, R_k]$ *and*
- *a* role-constraint *formula* $\phi[R_1, \ldots, R_k]$

For example, the following activity specification describes the class of plan instances with the feature SURVEY but not HEURISTIC, where the variables that fill the roles START and DESTINATION are instantiated to values in the same sector.

```
<{SURVEY}, {HEURISTIC},
 {START, DESTINATION},
 {(= (SECTOR START) (SECTOR DESTINATION))}>
```

Desire Specification

A *desire specification* constitutes the goal-oriented analogue of an activity specification, consisting of a collection of required features, prohibited features, roles, and role constraints for goals. We use the symbol δ to represent a generic desire specification.

Agent Context

Just as individual plans employ preconditions to limit their applicability, guidance requires a similar mechanism for defining scope. To this end, we introduce the notion of an *agent context*. While plan preconditions are generally limited to beliefs about the world state, our model of agent context focuses on the full operational state of an agent, characterized in terms of its beliefs, desires, and intentions. Beliefs are specified in terms of constraints on the current world state. Desires are specified as desire specifications that describe goals that the agent has adopted. Intentions are specified through activity specifications that describe plans currently in execution by the agent.

Our model of agency assumes a hierarchical collection of plans and goals; furthermore, agents are capable of multi-tasking (i.e., addressing multiple goals in parallel). Within a given phase of the BDI execution cycle, goals for an agent of this type can be scoped in three different ways:

- *Current goal:* the goal for which the BDI interpreter is selecting a plan to execute
- *Local goals:* the current goal, or any of its ancestors
- *Global goals:* any goal of the agent

By distinguishing these different scopes for goals, guidance can be localized to more specific situations. Plans being executed can be scoped in a similar fashion.

Definition 2 (Agent Context) *An* agent context *is defined by a tuple* $\kappa = \langle \Phi, \Delta, A \rangle$, *where*

- Φ *is a set of well-formed formulae*
- $\Delta = \Delta^C \cup \Delta^L \cup \Delta^G$ *is a set of current, local, and global desire specifications, respectively*

- $A = A^L \cup A^G$ *is a set of local and global activity specifications, respectively.*[2]

Permission Requirements

Permission requirements are defined in terms of an *agent context* and a *permission-constrained activity specification*. The agent context defines conditions on the operating state of the agent that limit the scope of the permission requirement. The permission-constrained activity specification designates a class of plan instances for which permission must be obtained.

Definition 3 (Permission Requirement) *A* permission requirement $\langle \kappa, \alpha \rangle$ *consists of an agent context* κ *and an activity specification* α.

The interpretation of a permission requirement is that, when an agent's BDI state matches the specified agent context, permission must be obtained from the supervisor in order to execute a plan instance that matches the permission-constrained activity.

Example 1 (Permission Requirement) The statement "Seek permission to abandon survey tasks with priority > 5" could be translated into a permission requirement of the form

```
Agent Context:
 Local Activity Spec:
  Features+: SURVEY-TASK
Permission-Constrained Activity Spec:
  Features+: ABANDON
  Roles: CURRENT-TASK
  Constraint: (> (TASK-PRIORITY CURRENT-TASK) 5)
```

Consultation Requirements

A consultation requirements consists of an *agent context* and a *consultation role*. The interpretation of a consultation requirement is that when an agent's BDI state matches the agent context, any instantiation decision for a variable corresponding to the consultation role should be passed to the human supervisor.

Definition 4 (Consultation Requirement) *A* consultation requirement $\langle \kappa, R \rangle$ *consists of an agent context* κ *and a role R.*

Example 2 (Consultation Requirement) The guidance "When responding to medical emergencies, consult when selecting a medical evacuation site" would be translated into a permission requirement of the form

```
Agent Context:
 Local Activity Spec:
  Features+: EMERGENCY-RESPONSE, MEDICAL
Consultation Role: MEDEVAC-SITE
```

---

[2]Because the motivation for guidance is to influence the choice of plan for the current goal, we exclude the specification of a current plan from the intentions of an agent context.

Strategy Preference

Strategy preference guidance consists of two components: an *agent context* and a *response activity specification*. The activity specification designates the class of recommended plan instances to be applied (i.e., choice of plan and variable instantiations for designated roles) when the agent enters a state that matches the designated agent context.

Definition 5 (Strategy Preference) *A strategy preference rule is defined by a pair* $\langle \kappa, \alpha \rangle$ *where* κ *is an agent context and* α *is an activity specification.*

Example 3 The statement "Don't take on medical emergencies involving fewer than 5 people when the current task priority exceeds the emergency severity" could be represented by the following strategy preference:

```
Agent Context:
  Current Desire Spec:
    Features+: RESPOND-TO-EMERGENCY
    Roles: EVENT
    Constraint:
     (AND
      (= (EVENT-TYPE EVENT) MEDICAL)
      (< (EVENT-NUMBER-AFFECTED EVENT) 5))
Response Activity Spec:
   Features-: ADOPT
   Roles: EVENT, CURRENT-TASK
   Constraint:
      (> (TASK-PRIORITY CURRENT-TASK)
         (EVENT-SEVERITY EVENT))
```

A goal with the feature RESPOND-TO-EMERGENCY and role EVENT triggers consideration of the guidance, provided EVENT is a an emergency of type MEDICAL, and fewer than 5 people are affected. The response activity specification indicates not to adopt responsibility for the emergency in the event that the priority of CURRENT-TASK is greater than the severity of EVENT.

SEMANTICS AND ENFORCEMENT OF GUIDANCE

Space limitations preclude full descriptions of the formal semantics for satisfaction of guidance by agent execution and algorithms for guidance enforcement. We present a brief overview here; details can be found in [13].

Semantically, guidance acts as a filter on the plan instances that an agent can execute. When a standard BDI agent attempts to find an instance of a plan from its library to apply to a goal, it determines a set of applicable plan instances based on the plan cues and preconditions. The guidance limits this set further in accord with the following conventions.

A guidance rule is deemed *relevant* at the time that the applicable plans are being filtered if the agent context matches the current operational state of the agent. Each relevant strategy preference rule filters out plan instances that do not match the response activity specification. Each relevant permission requirement rule filters out plan instances that both match the permission-constrained activity specification and are refused permission by the supervisor. Each relevant consultation requirement rule filters out plan instances that have the consultation role but do not bind the corresponding role variable to a value desired by the supervisor.

Enforcement of guidance is attained through a simple modification to the standard BDI interpreter loop at the point where where a plan instance is selected in response to a posted goal. First, the current BDI operational state for an agent is matched to the agent context components of all currently defined guidance to determine the relevant guidance for the current execution cycle. The relevant strategy preference rules are then used to eliminate plan instances that do not match their response activity specification. The remaining plan instances are then ordered in accord with any meta-control policies for plan ordering that may have been defined. This list is then traversed in order to find the first for which either the plan instance is not affected by relevant permission or consultation requirement rules, or queries to the human supervisor elicit any required execution permissions and instantiations for role variables. The agent then applies the selected plan instance to the current goal.
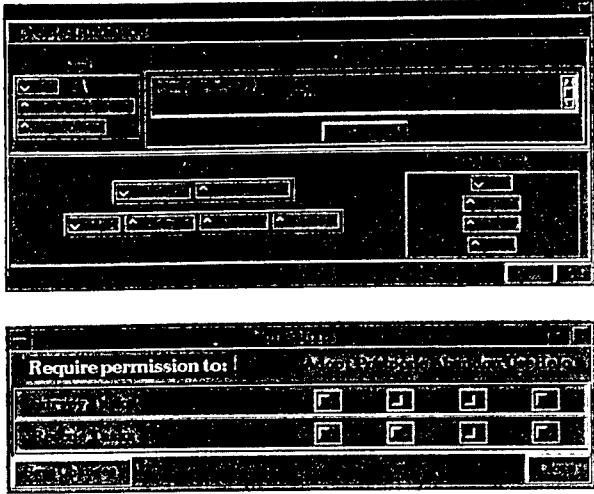
CONFLICTING GUIDANCE

User guidance provides a powerful mechanism for runtime customization of agent behavior. However, it also introduces the potential for problems in the event that the guidance recommends inconsistent responses. Robustness of operations necessarily requires mechanisms that can detect problematic user guidance and respond in a manner that does not jeopardize the stability of an agent.

Conflicts can arise in different forms. Here, we distinguish between *direct* and *indirect* conflicts.

Direct conflicts arise when guidance yields recommendations that conflict with each other within a given cycle of the BDI interpreter. For example, *Execute plan P* and *Don't execute plan P*. Direct conflicts are easily detected. They can be resolved by associating *weights* with strategy preferences rules that indicate degree of preference. A policy for combining and comparing the weights associated with the strategy preference rules that made the conflicting recommendations can then be used to select a preferred response. TIGER incorporates an approach of this type to deal with direct conflicts.

Indirect conflicts arise when guidance recommends multiple plans for execution such that, while their execution can be initiated, it is impossible for all of them to complete successfully. For example, the simultaneous execution of two plans could lead to deadlock or livelock situations, or downstream resource contention. Powerful detection mechanisms are required to deal with this class of conflict; TIGER does not yet include capabilities of this type.

Figure 2. Selected Guidance Specification Tools

GUIDANCE INTERFACE TOOLS

The motivation for our work on agent directability is to enable users to more readily direct and manage agents in dynamic, unpredictable environments. The language presented in earlier sections provides a highly expressive formalism in which to define agent guidance; however, the complexity of the language could overwhelm a typical user. For this reason, we have been developing tools to help users define and manipulate agent guidance. Figure 2 presents two such tools from the TIGER system.

The first tool is a *guidance authoring interface* that walks the user through the process of constructing complex pieces of guidance. To enable a simple specification process, the tool does not support the full expressivity of the formal guidance language; however, it supports a broad range of expressions (including the examples described in this paper).

The second tool is a *permissions window*. It enables users to activate/deactivate permission requirements for certain classes of action performed on certain types of task. In particular, selections made through this interface are compiled into corresponding permission requirement structures. While this interface limits the scope of permission requirements that can be expressed, it provides a simple, accessible specification mechanism.

In addition to the two tools described above, we have also developed a *guidance library* that stores predefined pieces of guidance. Users can then select from predefined guidance, as appropriate, to address their needs in a particular situation.

RELATED WORK

Recognition of the need for technologies to support human-agent interactions has grown substantially in the past few years. To date, however, few concrete technical approaches have been proposed to address the problem of agent directability.

Scerri et al. [15] apply Markov decision processes (MDPs) to provide a form of adjustable autonomy. Their approach involves predefining an MDP for each agent to describe possible courses of action. The agent uses expected utility estimates from this model to determine when to consult the supervisor, and adjusts the model parameters based on experience. To avoid learning inappropriate behavior, users can define predefined constraints on what can be learned. In contrast to our approach of having a human explicitly define a policy for autonomy, an agent within this framework determines an appropriate level on its own.

Schreckenghost et al. [16] apply the concept of adjustable autonomy to the management of space-based life support systems. In their system, a human can take over both the selection of tasks to perform and the execution of those tasks. In contrast to our use of an explicit policy language, the level of autonomy is specified by directly altering a "level of autonomy" setting (*manual* versus *autonomous*) either for all tasks, for a subsystem, or for an individual task.

Our strategy preference guidance selects among previously defined alternative plans; it does not expand the behavioral capabilities of the agent. In contrast, the work on *policy-based control* for distributed systems managements supports the runtime definition of new behaviors (e.g., [9]). Policy languages in this community focus on the concepts of *authority* and *obligation* to perform actions.

CONCLUSION

This paper presents a framework for human directability of agents that enables a user to define polices for adjustable agent autonomy and strategy preference. Through these mechanisms, a human supervisor can customize the operation of agents to suit his individual preferences and the dynamics of unexpected situations. In this way, system reliability and user confidence can be increased substantially over fully autonomous agent systems. The power of these ideas has been demonstrated within the TIGER system, which supports a human intelligence officer in managing a community of agents engaged in tasks for information gathering and emergency response.

Many outstanding issues in this area remain to be addressed; we briefly describe three topics for future work.

Detecting and Resolving Guidance Conflicts As discussed above, TIGER recognizes only a limited class of guidance-related conflicts (namely, direct conflicts among guidance). Indirect conflicts among guidance, and conflicts between guidance and ongoing activities require more powerful detection methods that reason about the downstream effects and requirements of plans. We are also interested in expanding our simple prioritization approach to resolving direct con-

flicts among guidance to incorporate more advanced conflict resolution policies (e.g., [4, 8]).

Community Guidance The forms of agent directability described in this paper focus on influencing the behavior of an individual agent. Human supervisors will also want to express control at the *community* level, to encourage or discourage various forms of collective or emergent behaviors. The guidance Keep 2 trucks within 15 miles of headquarters. provides an example. Enforcement of this type of guidance will require mechanisms that support information exchange and coordinated action selection among groups of agents.

Collaborative Control Our model of agent directability provides a form of *supervised autonomy* [1] in which control over autonomy rests solely with the human supervisor. Some situations may benefit from a more collaborative approach [6], where both sides share control over initiative. For example, an agent may choose to initiate a dialogue with the human in situations where adherence to guidance would interfere with the pursuit of current goals, rather than blindly following the user's recommendations.

REFERENCES

1. K. S. Barber and C. E. Martin. Agent autonomy: Specification, measurement, and dynamic adjustment. In *Proceedings of the Autonomy Control Software Workshop at Autonomous Agents*, 1999.

2. P. Bonasso. Issues in providing adjustable autonomy in the 3T architecture. In *Proceedings of the AAAI Spring Symposium on Agents with Adjustable Autonomy*, 1999.

3. H. Chalupsky, Y. Gil, C. A. Knoblock, K. Lerman, J. Oh, D. Pynadath, T. A. Russ, and M. Tambe. Electric Elves: Applying agent technology to support human organizations. In *Proceedings of the Thirteenth Conference on Innovative Applications of Artificial Intelligence*, 2001.

4. F. Dignum, D. Morley, E. A. Sonenberg, and L. Cavedon. Towards socially sophisticated BDI agents. In *Proceedings of the Fourth International Conference on MultiAgent Systems (ICMAS'2000)*, 2000.

5. G. Ferguson and J. Allen. TRIPS: Towards a mixed-initiative planning assistant. In *Proceedings of the AIPS Workshop on Interactive and Collaborative Planning*, 1998.

6. T. Fong, C. Thorpe, and C. Baur. Collaborative control: A robot-centric model for vehicle transportation. In *Proceedings of the AAAI Spring Symposium on Agents with Adjustable Autonomy*, 1999.

7. M. P. Georgeff and F. F. Ingrand. Decision-making in an embedded reasoning system. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, 1989.

8. E. Lupu and M. Sloman. Conflicts in policy-based distributed systems. *IEEE Transactions on Software Engineering, Special Issue on Inconsistency Management*, 25(6), 1999.

9. J. D. Moffett and M. S. Sloman. Policy hierarchies for distributed systems management. *IEEE Journal on Selected Areas in Communications*, 11(9), 1993.

10. K. L. Myers. Strategic advice for hierarchical planners. In L. C. Aiello, J. Doyle, and S. C. Shapiro, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Fifth International Conference (KR '96)*. Morgan Kaufmann Publishers, 1996.

11. K. L. Myers. Domain metatheories: Enabling user-centric planning. In *Proceedings of the AAAI-2000 Workshop on Representational Issues for Real-World Planning Systems*, 2000.

12. K. L. Myers and D. N. Morley. Directing agent communities: An initial framework. In *Proceedings of the IJCAI Workshop on Autonomy, Delegation, and Control: Interacting with Autonomous Agents*, 2001.

13. K. L. Myers and D. N. Morley. The TRAC framework for agent directability. Technical report, Artificial Intelligence Center, SRI International, 2001.

14. A. S. Rao and M. P. Georgeff. BDI agents: From theory to practice. In *Proceedings of the International Conference on Multi-Agent Systems (ICMAS-95)*, San Francisco, USA, 1995.

15. P. Scerri, D. Pynadath, and M. Tambe. Adjustable autonomy in real-world multi-agent environments. In *Proceedings of the International Conference on Autonomous Agents*, 2001.

16. D. Schreckenghost, J. Malin, C. Thronesbery, G. Watts, and L. Fleming. Adjustable control autonomy for anomaly response in space-based life support systems. In *Proceedings of the IJCAI Workshop on Autonomy, Delegation, and Control: Interacting with Autonomous Agents*, 2001.

17. D. E. Wilkins and K. L. Myers. A common knowledge representation for plan generation and reactive execution. *Journal of Logic and Computation*, 5(6), 1995.

Applying Natural Language Processing (NLP) Based Metadata Extraction to Automatically Acquire User Preferences

Woojin Paik, Sibel Yilmazel, Eric Brown, Maryjane Poulin, Stephane Dubon, Christophe Amice

solutions-united, inc.
Syracuse, NY USA
{woojin, sibel, eric, mjp, stephane, chris }@solutions-united.com

Abstract

This paper describes a metadata extraction technique based on natural language processing (NLP) which extracts personalized information from email communications between financial analysts and their clients. Personalized means connecting users with content in a personally meaningful way to create, grow, and retain online relationships. Personalization often results in the creation of user profiles that store individuals' preferences regarding goods or services offered by various e-commerce merchants. With the introduction of e-commerce, it has become more difficult to develop and maintain personalized information due to larger transaction volumes. <!metaMarker> is an NLP and Machine Learning (ML)-based automatic metadata extraction system designed to process textual data such as emails, discussion group postings, or chat group transcriptions. <!metaMarker> extracts both explicit and implicit metadata elements including proper names, numeric concepts, and topic/subject information. In addition, Speech Act Theory inspired metadata elements, which represent the message creators' intention, mood, and urgency are also extracted. In a typical dialogue between financial analysts and their clients, clients often discuss the items that they liked or have an interest. By extracting this information, <!metaMarker> constructs user profiles automatically. This system has been designed, implemented, and tested with real-world data. The overall accuracy and coverage of extracting explicit and implicit metadata is about 90%. In summary, the paper shows that an NLP-based metadata extraction system enables automatic user profiling with high effectiveness.

Keywords

Natural Language Processing, user preference elicitation, metadata extraction.

INTRODUCTION

For a number of years both manual and automatic approaches to the construction of knowledge bases have been studied and implemented. Manual construction of knowledge bases has been too expensive to be practical and automatic approaches have not yet produced domain-independent and usable knowledge bases [9].

Lack of practically usable knowledge bases led to two key problems in preventing wide-scale deployment of knowledge-based systems; that is the knowledge base and inference engine. These problems are commonly referred to as brittleness and the knowledge acquisition bottleneck [8&9]. A brittle system can respond appropriately only to a narrow range of questions. More precisely, such a system cannot answer questions that were not originally anticipated by the programmer. The other problem with knowledge-based systems is that crafting the statements that are entered into the knowledge base requires an enormous amount of training, time, and effort. Knowledge engineers tend to be highly skilled people but few of them can enter more than a small number of statements into a knowledge base in an average day. Brittleness and the knowledge-acquisition bottleneck are severe limitations.

In recent years there has been increased interest in textual information extraction research using natural language processing techniques. The most common medium of storing knowledge is text; textual information extraction is an approach to acquire knowledge from text.

The study reported in this paper describes an adaptation of a Natural Language Processing (NLP) based information extraction system which was originally developed to automatically populate knowledge bases, as a user preference elicitation tool. The focus of this paper will be the application of the information extraction technology to enable the data-controlled personalization in the context of e-business [14]. Personalization modifies an underlying system to better address the preferences of end users, be they corporate professionals or consumers [13]. It often results in the creation of user profiles that store individuals' preferences regarding goods or services offered by various e-commerce merchants. Based on Gartner Group [14] it was predicted that major enterprises with an Internet presence will analyze

their employees' and clients' behavior with a view toward automatically tailoring online interaction by 2003.

The email communication between the financial analysts and their clients was selected as the source for extracting information to populate the client profiles. The personalization information extraction system was able to achieve a high level of accuracy.

PERSONALIZATION

In its most general form, personalization modifies an underlying system to better address the preferences of end users, be they corporate professionals or consumers [13]. The Profile, which is the collection of data describing the criteria for customizing presentation or content, is the key to personalization. Linguistically speaking, personalization can be considered as a way to satisfy the Maxim of Relation [3]. According to Grice, in a talk exchange the participants are expected to be conscious of the so-called Cooperative Principle, which states: "Make your conversational contribution such as is required, at the stage at which it occurs, by the accepted purpose or direction of the talk exchange in which you are engaged" [3]. Conversing in accordance with the Cooperative Principle will yield maxims of Quantity (i.e. Don't say more or less than is required), Quality (i.e. Tell what you believe is true, be sincere), Relation (i.e. Be relevant), and Manner (i.e. Avoid ambiguity and obscurity) [1].

On the other hand, personalization has different meanings to different people. Today, the three most common forms of personalization are: Enterprise-Controlled, End-user Controlled, and Data-Controlled [14]. The Enterprise-controlled form of personalization is making decisions based upon the preferences or predefined criteria set by the owner of the content. Criteria may be based on the factors of target platform, user role, level of service, or information extracted from an enterprise or a third-party repository. The systems of this type control access to content or functionality based on what the user is likely to purchase or has licensed. End-user controlled content delivery is based on criteria set by the customer. User controlled content applications in portals and in the enterprise context are examples of end-user controlled form of personalization [13&14]. Data-controlled personalization is generated by affinity-data; for instance, the purchasing patterns and preferences of like consumer groups. Affinity-data are derived by applying data-mining algorithms to market basket analysis. Affinities can be used to fine-tune customer interaction. For example, data-mining questionnaires can reveal the dislikes of different customer groups which can be further used to refine marketing campaigns. Furthermore, methods like collaborative filtering explore the choices of similar peer groups and recommend what other customers did at a certain point. Another form of data-controlled personalization is to leverage similarity of product descriptions in electronic product catalogs to cross-market similar products, given consumers' interest in a particular product [14].

APPLYING BOTH DOMAIN-INDEPENDENT & DOMAIN DEPENDENT INFORMATION EXTRACTION TO ACQUIRE USER PREFERENCES

One of the underlying text analysis models behind the information extraction system, which is described in this paper, is a recently emerged broad & shallow information extraction framework. This domain-independent information extraction framework was used to develop an automated system to update knowledge bases [10]. In comparison to the traditional deep & narrow domain-dependent information extraction systems such as the ones reported in the Message Understanding Conferences [5,6,7,&8], which require extensive manual development effort by the subject matter experts, the broad & shallow information extraction systems are considered to adapt more easily to new subject domains [10].

Like many other systems, the domain-independent information extraction algorithm is based on sub-language analysis of text by taking advantage of the common practices of writers on a similar subject [11]. For example, there are regularities in the way that weather reports are composed. It is fairly straightforward to develop rules to extract key information about the weather reports by anticipating what type of information will be described in what manner. Similarly, previous work has shown that it is possible to develop a sub-language grammar to extract highly accurate information from news type stories. In conjunction with the use of case grammar type simple semantic relations such as 'agent', 'location', and 'cause', the use of sub-language grammar has been shown to enable extraction of practical, usable information from news type text. This approach to extracting domain-independent information has been tested and shown successful in the Defense Advanced Research Project Agency (DARPA)'s High Performance Knowledge Base (HPKB) program [10]. The system developed for HPKB exhibited both high precision and high recall for information extraction tasks.

In this paper, we describe <!metaMarker>, an eXtensible Markup Language (XML)-based automatic metadata generation tool. <!metaMarker> is a novel hybrid information extraction system, which utilizes both domain-independent and domain-dependent information extraction algorithms. <!metaMarker> does not extract case grammar type semantic relations like other information extraction systems. However, <!metaMarker> extracts and classifies information objects from numerous types of business communications. The foundation of <!metaMarker> is built upon the richness and accuracy of Natural Language Processing (NLP) techniques and the adaptability and customization potential of Machine Learning (ML). It utilizes an expanded metadata framework developed for enterprise communications consisting of:

- Traditional descriptive, citation-like features: author, subject, time/date/place of creation

- Descriptive features unique to business communications: company/organization information, a specific order, named product features

- Additional situational or use aspects which provide critical contextual information: author's intention or goal, degree of urgency, mood or attitude

<!metaMarker> also facilitates addition of custom categories by derivation from previously extracted information. For example, extracted metadata elements such as 'subject', 'intention', and 'mood' might be used as the basis for defining another tag 'priority' that could be automatically assigned to a specific email based on the extracted values for the three original metadata elements. One possible instantiation is 'high' value assigned to 'priority' element if 'return of purchased product' was the value for 'subject' metadata element, 'complain' was the value for 'intention' element, and 'angry' was the value for 'mood' element.

In applying <!metaMarker> to email communication, derivation of relevant metadata elements was accomplished through both inductive means by analyzing a large number of emails, and deductive means by considering general theories of human communications and research results in the area of computer mediated communication. There were some explicit metadata elements and their values which were directly extractable from the body of email messages. For example, typical biographical information such as 'name of sender', 'title', 'affiliation', 'physical address', 'phone number', 'home page', or 'motto', were extracted by applying an email sublanguage grammar. The email sublanguage grammar was developed based on an analysis of output from various natural language processing components such as the 'concept categorization module'.

There were also implicit metadata elements and their values, identifiable through an email discourse model analysis. These elements were, 'subject/topic'. 'intention', 'mood', and 'urgency'. Subject/topic refers to the classification of the message contents into categories such as are used in a general purpose thesaurus such as Roget's. Some examples of the values for this element are: law & politics, religion, science & technology, business & economics, and recreation & sports. The 'intention' metadata element comes from Searles's [13] speech act theory, which focuses on what people 'do' with language i.e. the various speech acts that are possible within a given language. <!metaMarker> utilizes discourse analysis of the email messages to classify authors' intentions into values such as 'claims', 'promises', 'requests', 'blessing', 'thanking', or 'permitting'. The 'mood' element refers to the email authors' emotional state. The values for this element are: 'strongly negative', 'negative', 'neutral', and 'positive'. Finally, 'urgency' is related to time, i.e. when something needs to be done (or was supposed to be done). The messages are classified and the following values are assigned to each message: 'very urgent', 'urgent', and 'neutral'.

In the research reported in this paper, <!metaMarker> is used as an implementation platform to automatically extract metadata for user preferences by incorporating user preferences specific extraction and tagging algorithms. To adapt <!metaMarker> to extract user preference specific metadata elements, the situational or use aspect related metadata are expanded to included new metadata elements such as 'like', 'dislike', 'interested', or 'not-interested.' Specifically these are the elements explaining the author's intention or goal. They are implicit in the text and thus derived through a text discourse model analysis of email type communicative text.

The following is a sample email communication between a financial analyst and his/her client.

Question from a client:

I think the key to the future is the use of personalization software. Do you think BroadVision will rebound to its high in the next six months?

Response from a financial analyst:

BroadVision is more heavily concentrated in the B2B market, which, long term, we believe, is attractive. Though we like BroadVision, we think Ariba; I2 Technologies; and Commerce One will be the dominant players.

In addition to the typical metadata which are proper named concepts or numeric concepts, the user preference specific <!metaMarker> extracts the concepts that client liked, disliked, and also was interested in from this example. When the same type of information extraction is applied to the financial analyst's response, <!metaMarker> also extracts the concepts that the financial analyst's liked. In the following, a step-by-step analysis of the client question will be shown. This depiction shows the underlying NLP and ML processing of <!metaMarker>.

Step #1 (NLP) – sentence boundary identification

<s#1> I think the key to the future is the use of personalization software. </s#1> <s#2> Do you think BroadVision will rebound to its high in the next six months? </s#2>

<s> denotes the beginning of a sentence and </s> denotes the end of a sentence.

Step #2 (NLP) – part-of-speech tagging

<s#1> I|PRP think|VBP the|DT key|NN to|TO the|DT future|NN is|VBZ the|DT use|NN of|IN personalization|NN software|NN .|. </s#1> <s#2> Do|MD you|PRP think|VBP BroadVision|NP will|MD rebound|VB to|TO its|PRP$ high|JJ in|IN the|DT next|JJ six|CD months|NNS ?|. </s#2>

This step assigns part-of-speech information after each word in the sentence. '|' is used to delimit the word and the corresponding part-of-speech tag. The tag set is based on University of Pennsylvania's Penn Treebank Project [12]. For example, PRP means 'personal pronoun', VBP means 'present tense verb', and DT means 'determiner'.

Step #3 (NLP) – morphological analysis

<s#1> I|PRP think|VBP the|DT key|NN to|TO the|DT fu-ture|NN is|VBZ|be the|DT use|NN of|IN personalization|NN software|NN .|. </s#1> <s#2> Do|MD you|PRP think|VBP BroadVision|NP will|MD rebound|VB to|TO its|PRP$ high|JJ in|IN the|DT next|JJ six|CD months|NNS|month ?|. </s#2>

This step determines the root form of each word and adds it to each word. In this example, there are two cases. 'is' is assigned with 'be' and 'months' is assigned with 'month'.

Step #4 (NLP) – multi-word concept identification

<s#1> I|PRP think|VBP the|DT key|NN to|TO the|DT fu-ture|NN is|VBZ|be the|DT use|NN of|IN <cn> personaliza-tion|NN software|NN </cn> .|. </s#1> <s#2> Do|MD you|PRP think|VBP <pn> BroadVision|NP </pn> will|MD rebound|VB to|TO its|PRP$ high|JJ in|IN the|DT <nc> next|JJ six|CD months|NNS|month </nc> ?|. </s#2>

This step identifies the boundary of the concepts. For example, proper names are identified by <pn> tags. Numeric concepts are delimited by <nc> tags. All other multi-word concepts are bracketed by <cn> tags.

Step #5 (NLP) – concept categorization

<s#1> I|PRP think|VBP the|DT key|NN to|TO the|DT fu-ture|NN is|VBZ|be the|DT use|NN of|IN <cn> personaliza-tion|NN software|NN </cn> .|. </s#1> <s#2> Do|MD you|PRP think|VBP <pn cat=company> BroadVision|NP </pn> will|MD rebound|VB to|TO its|PRP$ high|JJ in|IN the|DT <nc cat=time> next|JJ six|CD months|NNS|month </nc> ?|. </s#2>

Each proper name and numeric concept is assigned with its semantic type information according to the predetermined schema. Currently, there are about 60 semantic types, which are automatically determined by <!metamarker>.

Step #6 (ML) – implicit metadata – mood, urgency, inten-tion, and topic – generation

<s#1> I|PRP think|VBP the|DT key|NN to|TO the|DT fu-ture|NN is|VBZ|be the|DT use|NN of|IN <cn> personaliza-tion|NN software|NN </cn> .|.

<modalityInfo>

    <mood> neutral </mood>

    <urgency> neutral </urgency>

    <intention> belief & judgment </intention>

</modalityInfo>

<topic> computer science & technology </topic>

</s#1>

<s#2>

Do|MD you|PRP think|VBP <pn cat=company> BroadVi-sion|NP </pn> will|MD rebound|VB to|TO its|PRP$ high|JJ in|IN the|DT <nc cat=time> next|JJ six|CD months|NNS|month </nc> ?|.

<modalityInfo>

    <mood> neutral </mood>

    <urgency> neutral </urgency>

    <intention> belief & judgment </intention>

</modalityInfo>

<topic> trade & commerce </topic>

</s#2>

This step assigns implicit metadata to each sentence by categorizing each sentence according to the predetermined schema of modality and topic/subject. The sentence-by-sentence categorization is carried out by the text classifiers such as Bayesian probabilistic classifier or k-Nearest Neighbor classifier by utilizing a training data set, which consists of a pre-coded set of example sentences. Each sen-tence is represented as a feature vector, which consists of NLP extracted explicit metadata from the steps #1 to #5. At the end of this stage, <!metaMarker>, which is not adapted to extract user preferences, is designed to generate a table to be incorporated as a part of a relational database.

Step #7 (ML) – user preference extraction

<s#1> I|PRP think|VBP the|DT key|NN to|TO the|DT fu-ture|NN is|VBZ|be the|DT use|NN of|IN <cn> personaliza-tion|NN software|NN </cn> .|.

<modalityInfo>

    <mood> neutral </mood>

    <urgency> neutral </urgency>

    <intention> belief & judgment

        <like> personalization software </like>

    </intention>

</modalityInfo>

<topic> computer science & technology </topic>

</s#1>

<s#2>

Do|MD you|PRP think|VBP <pn cat=company> BroadVi-sion|NP </pn> will|MD rebound|VB to|TO its|PRP$ high|JJ in|IN the|DT <nc cat=time> next|JJ six|CD months|NNS|month </nc> ?|.

<modalityInfo>

    <mood> neutral </mood>

    <urgency> neutral </urgency>

    <intention> belief & judgment

        <interested> BroadVision/company

        </interested>

    </intention>

</modalityInfo>

<topic> trade & commerce </topic>

</s#2>

Currently, the scope of the adaptation of <!metaMarker> to extract user preferences is limited to four types of metadata. They are 'like', 'dislike', 'interested', and 'not interested'. The user preference extraction is a combination of explicit

and implicit metadata generation methods. First each sentence is categorized according to the positive and negative facets of 'like' and 'interested' user preferences. Then, certain explicit metadata extraction results such as proper names and multi-word concepts other than numeric concepts for each sentence is correlated with the user preference information. The above output of the step #7 shows that the client likes 'personalization software' and is interested in the company, BroadVision. This information will be entered into the user preference database so that the next interaction between the financial analyst and his/her client can be better focused on the clients' likes and interests. In addition, it is also expected that the financial analyst can push out certain relevant information to the client according to his/her preferences.

APPLYING TEXT CLASSIFICATION TO EXTRACT IMPLICIT METADATA AND USER PREFERENCE

To assign implicit metadata to each sentence, each sentence is categorized according to the predetermined schema of modality and topic/subject.

The first text classification task involves manually classifying a set of training documents in preparation for feeding the automatic system. Each training document is classified as "in" or "out" of the individual classes as outlined by the class definitions.

The next step is to take these manually classified documents and process them through the trainable text classification system. During the process it builds a vector of terms, phrases, and entities extracted from the text. Multi-level Natural Language Processing outputs are the basis for these textual data feature representations.

This collection of automatically generated features is then used to determine membership of new text within a particular class. The system determines the "certainty of membership" for each of the documents compared to each of the classes. If we consider a range of 1 to 0 where 1 means a document is definitely a member of a certain class, and 0 means a document is definitely a non-member of a certain class, we can say that values of 0 and 1 both have a "certainty of membership" value of 1. For either of these cases, we can confidently conclude that the document either 'does' or 'does not' belong within a given class. If we look at values close to .5 on the above scale, we have a "certainty of membership" value close to 0. This means for these cases, we cannot automatically determine whether or not a given document should be assigned to a given class. These documents are considered valuable in refining the classification system. By manually classifying these documents, and then feeding them back into the automatic system, we train it to recognize the subtle differences that distinguish how these documents should be classified.

EXPERIMENTS & RESULTS

The system that was used for the evaluation is a research version of the commercially available <!metaMarker> system. The research system includes new and different algorithms and functionalities, which are not fully tested and incorporated in the commercial system. Thus, the experiment results reported in this paper should not be used to gauge the effectiveness of the commercial version.

Two methods of measuring effectiveness that are widely used in the information extraction research community have been selected to evaluate the metadata extraction including the user preference extraction performance [2]. The methods are:

- **Precision**: the percentage of actual answers given that are correct.

- **Recall**: the percentage of possible answers that are correctly extracted.

Automatically extracted metadata was evaluated with the following criteria:

- If the automatically extracted metadata and the answer key, which is generated manually, are deemed to be equivalent, then the automatic extraction output is considered as "correct."

- If the automatically extracted information and the answer key do not match then it is considered as "incorrect."

Recall and precision are represented by the following equation (*possible* is defined as a sum of correctly extracted and missing metadata, and *actual* is defined as a sum of correctly extracted and incorrectly extracted metadata:

$$Recall = correct/possible$$

$$Precision = correct/actual$$

Explicit metadata extraction rules were developed inductively by analyzing randomly selected training data from a collection of actual emails which were sent by the customers of a commercial e-commerce merchant to the merchant. There were about 5,000 email messages in the training data set. The text classifier used to generate the implicit metadata was trained by the same email messages after the appropriate implicit metadata including the user preferences was manually coded.

The following steps were followed to measure the effectiveness of automatically extracting metadata from emails:

- Test data was randomly selected and consisted of a pre-determined number of email messages that were not used for training.

- A manual evaluation was conducted by presenting the automatically extracted metadata and the source text to three judges and asking them to categorize extracted

metadata as correct or incorrect, and to identify missing information.

- Precision and recall were computed for the automatically extracted metadata by applying the majority principle (i.e. assume the correctness of a judgment if two or more judges make the same judgment.)

- A failure analysis was conducted of all incorrectly extracted missing information.

The metadata extraction experiment was conducted against 100 randomly selected customer inquiry email messages. The evaluation result for the user preference specific metadata using this previously unseen data is shown in the Table 1.

Table 1. User Preference Extraction Evaluation Results

	Precision	Recall
Like	89%	85%
Dislike	91%	93%
Interested	88%	86%
Not Interested	82%	79%

It was expected that the 'Not Interested' category would result in the worst score since the development of the training data for this category was the most difficult one for the human coders. The humans had the most number of discrepancies for this category. On the contrary, 'Dislike' category scored best. This was also consistent with the human coders' experience with developing the training data set. They had the least discrepancies in finding email messages, which belong to the 'Dislike' category.

Table 2 shows the Mood metadata element extraction evaluation result using the same 100 email messages.

Table 2. Mood Extraction Evaluation Results

	Precision	Recall
Positive	71%	81%
Neutral	90%	95%
Negative	93%	90%
Strongly Negative	86%	44%

The working definition of each category is developed inductively by analyzing the data. The 'Positive' category should be assigned when the customer is pleased with the transaction and openly expresses satisfaction and/or happiness. The 'Neutral' category means that the customer states fact or asks a question; does not express emotion either positively or negatively. The customer has found no fault with the service, web site, or product. The 'Negative' category should be assigned when the customer is dissatisfied with the transaction, and sometimes is openly negative,

finding fault with the service, web site, or product and perhaps asking for clarification, explanation, or fix. The communication may include mild sarcasm. Finally, the 'Strongly Negative' means that the customer is extremely dissatisfied with the transaction - disgusted, irate, and many times is going to cancel the order. This is communicated directly in the e-mail. Many times the e-mail shows caustic sarcasm.

We expected that if there is a small number of the training data for a certain category then the categorization effectiveness of that category is usually lower than the other categories with more training data. 'Positive' and 'Strongly Negative' categories had the lesser number of the training data in comparison to 'Negative' and 'Neutral' categories. The evaluation result confirms our hypothesis.

It was also expected that there were high correlation between the occurrences of 'Positive' mood category with 'Like' and 'Interested' user preference categories. It turned out to be the case. In addition, 'Negative' and 'Strongly Negative' categories had high correlation with 'Dislike' category. However, the correlation between the negative mood categories and 'Not Interested' category had comparatively lower correlation. It seems that there are factors other than mood or emotions, which contribute to a customer not having interests in certain objects.

Table 3 shows the Urgency metadata element extraction evaluation result using the same 100 email messages.

Table 3. Urgency Extraction Evaluation Results

	Precision	Recall
Neutral	69%	90%
Urgent	82%	85%
Very Urgent	86%	59%
Urgent + Very Urgent	95%	86%

The working definition of urgency is described in the following. The 'Neutral' category is assigned to the messages when they convey no sense of urgency. The 'Urgent' category means that the message conveys a need for action or response within a *reasonable* timeframe. However, no specific time needs to be mentioned. The 'Very Urgent' category means that the message conveys a need for an *immediate* action or response. Often times the action or response was desired or needed by the customer prior to writing the message. Finally, 'Urgent + Very Urgent' is used to categorize the messages at two dimensions namely that an urgency is conveyed in the message or not.

We expected to see the better effectiveness when there is less number of categories for the system to learn. The evaluation results confirmed our expectation. The decision to have more number of categories versus less number of categories for a certain metadata element is dependent on

the application. The evaluation results shows one of the trade-offs of making such decision.

In summary, the experiment result is consistent with the previous evaluation of <!metaMarker>. The accuracy of assigning the most appropriate 'Intention' type metadata to each sentence in emails also revealed the similar results.

There are five intention type metadata elements. They are: background, beliefs & judgments, niceties, promise, and request. The average precision of correctly assigning these intention metadata elements was 89.40% and the recall was 89.20%. The maximum precision value was 95% and the minimum precision value was 85%. The maximum recall value was 97% and the minimum recall value was 77%. These figures are based on the same experiment procedure described for measuring user preference type metadata element assignment effectiveness.

CONCLUSION

A combined NLP and ML approach to automate user preference extraction is introduced and its performance on a number of email messages is described. The extended system, which is based on a general-purpose metadata generation system, accurately extracts user preferences in addition to the traditional descriptive, citation-like features, descriptive features unique to business communications, and situational or use aspects which provide critical contextual information. This system is designed to be a part of a larger Customer Relation Management (CRM) system that prioritizes & routes incoming customer inquiries and also populates user profiles.

The same underlying metadata extraction framework that is implemented as <!metaMarker> is currently adapted for other applications such as e-learning and monitoring consumer perception of medical goods or services. The goal of the e-learning application is to automatically assign relevant metadata tags to educational resources. For example, *Audience, Duration, Cataloging, Essential Resources, Education Level, Pedagogy, Quality, and Standards* are the education specific metadata elements that will be a part of the newly adapted <!metaMarker>. The goal of the other application is to monitor public perception of over-the-counter and prescription drugs. There are hundreds of chat rooms devoted to various medical conditions as well as discussion groups that discuss a particular medicine and its side effects. The proposed system will automatically categorize harvested information according to the newly developed metadata elements such as *Condition, Side Effects, Severity of Side Effects, Off-label Use, Cures offered to Mitigate the Side Effects, Alternative Medicine, Source, Usage, and Attitude.*

The major potential contribution of the research reported in this paper is the demonstration of successfully using NLP and ML techniques as part of a large-scale work flow system (e.g., CRM system) to solve real-world problems. This success became possible due to the advancement of hybrid domain-independent and domain-dependent NLP techniques, which depart from the common practice of developing a specific one-off NLP application for each problem area.

REFERENCES

[1] Brown, P. & Stephen C.L. Politeness: Some universals in language usage. Cambridge: Cambridge UP, 1987.

[2] Chincor, N. MUC-4 Evaluation Metrics. Proceedings of the Fourth Message Understanding Conference (MUC-4), McLean, VA, 1992.

[3] Grice, H.P. Logic & Conversation. Syntax & Semantics 3: 41-58, 1975.

[4] MUC-3. Proceedings of the Third Message Understanding Conference (MUC-3), San Diego, CA, Morgan Kaufmann, 1991.

[5] MUC-4. Proceedings of the Fourth Message Understanding Conference (MUC-4), McLean, VA, Morgan Kaufmann, 1992.

[6] MUC-5. Proceedings of the Fifth Message Understanding Conference (MUC-5), Baltimore, MD, CA, Morgan Kaufmann, 1993

[7] MUC-6. Proceedings of the Sixth Message Understanding Conference (MUC-6), Columbia, MD, Morgan Kaufmann, 1995.

[8] Musen, M.A. Widening the Knowledge-Acquisition Bottleneck: Automated Tools for Building and Extending Clinical Methods, in Hammond, W.E., Ed., AAAMSI Congress, San Francisco, CA, 1989.

[9] Paik, W. CHronological information Extraction SyStem (CHESS), Ph.D. dissertation, Syracuse University, Syracuse, NY, 2000.

[10] Sager, N., Friedman, C., & Lyman, M.S. Medical Language Processing: Computer Management of Narrative Data, Reading, MA: Addision-Wesley, 1987.

[11] Santorini, B. Part-of-speech Tagging Guidelines for the Penn Treebank Project. Technical report, Department of Computer & Information Science, U. of Penn, 1990.

[12] Searl, J.R. Speech Acts: an Essay in the Philosophy of Language. Cambridge University Press. NY, 1969.

[13] Smith, D. There Are Myriad Ways to Get Personal, Internet Week Online, 2000.

[14] Votsch V. & Linden, A. Do you know what personalization means? Gartner Grp T-10-9346, 2000

Ontology-Guided Knowledge Discovery in Databases

Joseph Phillips
Intelligent Systems Program
University of Pittsburgh
Pittsburgh, PA 15260, USA
josephp@cs.pitt.edu

Bruce G. Buchanan
Intelligent Systems Program
University of Pittsburgh
Pittsburgh, PA 15260, USA
buchanan@cs.pitt.edu

ABSTRACT

We present work in progress on a new methodology for leveraging the semantic content of ontologies to guide knowledge discovery in databases. Our system scans new databases to obtain type and constraint information, which users verify. Our system then uses this information in the context of a shared ontology to intelligently guide the potentially combinatorial process of feature construction. Further, our system learns each time it is applied, easing the user's verification task on subsequent runs.

KEYWORDS

Constructive induction, knowledge discovery in databases, ontology.

INTRODUCTION

Knowledge discovery in databases (KDD) is the knowledge-intensive process of discovering knowledge that is implicit in large and diverse databases [7]. It is an inherently iterative process of selecting data, preprocessing it, transforming it into a workable form, data mining over it, and interpreting the results. The process is knowledge intensive because all steps require domain-specific knowledge to decide which operations from a potentially large set might prove most useful. The choice of vocabulary is critical for discovery programs [17].

The growing importance of large scale, shared ontologies is demonstrated by several projects to construct them in a variety of domains. Two examples are Ontolingua [8] and the Unified Medical Language System [13]. Ontologies allow reasoning in hitherto intractable domains by codifying specific knowledge.

Our goal is to exploit the information contained in ontologies to the help KDD process. Specifically, we hope to:

1. automatically suggest and generate new attributes based upon semantic and domain information,

2. capture useful knowledge for reuse, and

3. reduce the user's workload to interpret new tables.

A tool that gradually accumulates knowledge of the databases of a domain is appropriate for and applicable to KDD because KDD is an iterative process. Researchers often rework their data. Integrating this knowledge with an ontology extends the ontologies usefulness.

The feature construction task shows the value of ontologies clearly. The composition of the most useful constructed attributes depends upon the semantic relationships among the attributes. For example, we expect to create a useful attribute by multiplying one attribute called "length of X" by another called "width of X". Expectations of usefulness diminish when we multiply "length of X" by "width of Y" for different X and Y.

Databases do not have the required semantic knowledge to intelligently guide feature construction because they were not designed to store it. Databases may specify constraints among attribute values (*e.g.*, for all X: length(X) > width(X)). This, however, is not the same as describing the relationships among the attributes themselves. Ontologies are designed to hold this meta-knowledge.

It is already well-accepted that knowledge-based learning and discovery can be enhanced with automatic feature construction [5][15][18], and by learning the historically best operators [16]. Mostly, however, the prior knowledge is specified separately for each new problem. Here we extend this line of research to develop general programs that can capture prior knowledge found to be useful for one problem area and reuse it in another domain. The shared knowledge is stored in a simple ontology, which, presumably, will be part of a much larger ontology like CYC [11].

We do not presume that an ontology is complete at the time a new data mining application is begun; to the con-

trary, we believe that new domains will bring new types of variables and knowledge about them. However, we also believe that data mining is not simply the one-time application of a program to a new database. In our own work, data mining frequently starts with small pilot studies and manual bias space search, including feature construction. With preliminary confirmation that the programs can find some interesting relationships, more data and greater expectations are introduced.

This paper is organized as follows. The next section mentions tools for related, but different, problems. The third section shows our process by stepping through an example database. The following section details and discusses experiments. The last section concludes.

RELATED WORK

Constructive induction is the process of creating new attributes from old ones for the purpose of knowledge discovery. Our work complements IDP and other approaches that use information theory metrics by specifying semantic constraints that can be applied in parallel. Donoho & Rendell [5] describe several types of domain knowledge that assist feature construction for data mining or discovery. In this paper, we extend that work to show how the ontology encoding that knowledge can be extended by the user of a discovery system in the context of a new problem.

Several tools exist to assist with refining and debugging knowledge bases for knowledge-based systems (e.g., [1][3][4][12][14][19][20]). (See Boose [2] and Gaines & Shaw [9][10] for systematic discussions of the types of tools that have been proposed.) Our goal, however, is use an ontology to help discover knowledge from databases. Unlike Boose [1], and others working on interactive knowledge elicitation tools, we try to reduce the amount of time required of a domain expert by starting with data in a database and inferring facts and relations about the variables using an underlying ontology. Unlike Erikson, et al. [6], and others working on construction of ontology editors, we rely on human efforts already invested in problem definition. That is, we assume that a person constructing a

database has provided a considerable amount of structure in the domain, and that automated tools can capture that structure and reuse it.

PROCESS

Overview

Our system has three components that work as follows (please see figure 1). The table interpreter reads a flat file of attribute names and their values. It infers attribute domains from the values, and verifies the domains by asking the user. Next, it re-reads the flat file to create a Prolog program that incorporates this data. The Prolog program also contains attribute annotations that tell how that attribute may be used.

Second, Prolog is invoked. Prolog uses an existing ontology and the newly created program to invent attributes according to defined rules. These rules are guided by the attribute annotations created by the table interpreter. The program generates values of both the existing attributes given in the table and the newly created attributes created by the rules. The values are written to a file.
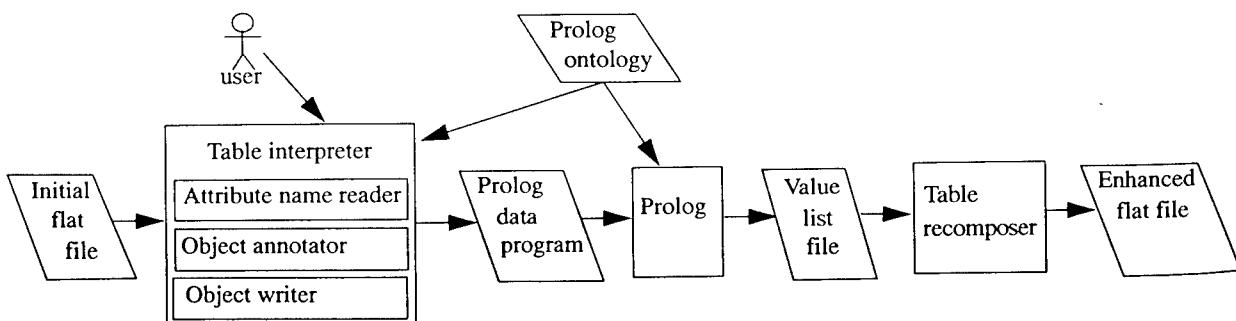
The last component recomposes the Prolog program's output into a new flat file table. This flat file has both the attributes of the original table, and the new attributes created by the Prolog rules. This file can be used as input for a variety of machine learning and data mining programs.

A Running Example

Table 1 is referred to as a running example throughout the paper. It lists a subset of data about patients who received orthopedic treatments in a rehabilitation ward at the University of Pittsburgh.

Table 1 lists the following attributes. **Person** gives a unique label to each patient that masks their identity. The attributes **age** and **sex** both have their conventional English meaning. **Start_time** tells when the patient entered the ward after some arbitrary reference time and **duration** tells the patient's time in the ward. Both are in days.

FIGURE 1. Overview of process

Admit_health and **discharge_health** values are based on the patient's Functional Independence Measurement (FIM^{SM}) at admit and discharge times. The FIM score is a gross indicator of how independently a patient can live without assistance from caregivers. It has been renamed "health" both for reader convenience and to signify that it is no longer the FIM score but the result of a transform on the FIM score. Numbers were changed to showcase the system's strengths (and weaknesses), and to further obscure patient identity.

TABLE 1. Orthopedic patient ward table (abbrev.)

person	age	sex	start_time	dur-ation	admit_health	dis-charge_health
p1	63	female	10	6	9.5	11.5
p2	69	male	20	8	8.4	10.3
p3	79	female	30	9	7.6	10.3

Reading tables

The first step of the first program is to read the original flat file. This provides it with the names of the attributes and a guess of their domains.

Attribute names are assumed to be on the first (or designated) line of the file in columns separated by tabs or commas. We also assume that all names come from the same natural language. We are building a list of likely synonyms of common attributes to enhance the program's knowledge of attribute names. For example: **sex, gender** and **M/F** are commonly used to refer to the same demographic information. Additional synonyms encountered in the future can be added.

Attribute domains are inferred by reading the values on subsequent lines. (They also are assumed to be in tab or comma separated columns.) Domains may be either symbolic, floating point, fixed point, integer, or integer coded.

Attributes that have text values underneath them are considered symbolic, and are assumed to have a fixed set of values associated with them. These values may have a natural order (*e.g.*, {before, during, after}) or may be unordered (*e.g.*, {female, male}). Attributes **person** and **sex** of table 1 would be considered symbolic attributes.

Attributes with numeric values that contain decimal points, are written in scientific notation, or that are too big or too small to represent as integers are considered floating point. Floating point domains have a maximum value, a minimum value, a mantissa size and an exponent size. **Admit_health** and **discharge_health** of table 1 would be considered floating point.

Floating point attributes may later be recast to fixed point attributes. Fixed point numbers cover the same range as floating point numbers, but all values between their minimum and maximum must be able to be generated by integer multiples of a fixed floating point increment. For example, a database of earthquakes may give quake magnitudes as fixed point numbers from 4.0 to 9.0 with fixed increment 0.1.

Most attributes with numeric values that are not floating point (or fixed point) are considered integer. (Read the next paragraph for the exception.) Integer domains also have minimum and maximum values associated with them. Attributes **start_time** and **age** of table 1 would be considered integer.

Attributes with integer values that are all between 0 and some small limit (currently 10) are assumed to have integer-coded domains. The numbers are assumed to be a code for text. Integer coded domains are really ordered symbolic domains (*e.g.* {1=before, 2=during, 3=after}) or unordered symbolic domains (*e.g.* {0=female, 1= male}). Attribute **duration** of table 1 would (incorrectly) be considered integer-coded because all values are between 0 and 10.

The user can, of course, override any default domain identification, and change domain properties such as the minimum or maximum.

Flat files are both the input and output of this process. Although flat files are not as informative as other standard database formats, they have the advantage of being easily generated and read by most database programs. Also, many machine learning and data mining tools expect or may utilize flat files as input. We are considering extending our tool to read common database formats. This would help identify domains.

Automatic and manual annotation of attributes

Although many possible semantic relationships may be invented, we seek a small set that is general and may be extended as needed. Therefore, knowledge of relations among attributes is organized in terms of processes and states (in addition to domain objects and classes). All database tuples are treated as associating values with an explicitly or implicitly stated process. For example, in a database of earthquakes, each quake is a process. In the hospital admittance and discharge data in table 1, each treatment tuple is a process. Both earthquakes and patient treatments are subclasses of the more general frame **process_class**. Both have unique properties specific to their domains, and both inherit properties just by being processes. Chief among these inherited properties are start and stop states, and duration.

States tell what is true at a particular time for a subset of the Universe. All transitory information is associated with

some state. States have at least one key attribute that help to uniquely identify them. The key attribute that all states share is time. Like processes, states are objects that may have and inherit values.

After reading the initial database, the table interpreter has gained as much information as it can from the source document. Its knowledge, however, is very approximate and poorly describes the relations among attributes. Complete attribute annotations specify the attribute name, domain and the object or process that the attribute's values describe. It now turns to two auxiliary sources of knowledge: records of attributes from prior tables stored in a shared ontology, and the user.

The program looks for the previously seen attribute description that best matches each attribute read from the new table. The program uses both pieces of information that it knows about the read table attribute: its name and its domain. For now, the name match is a simple string prefix comparison that would match **age** with **age at admission** but not with **present age**. We envision a more sophisticated matching routine that matches more freely by consulting dictionaries of synonyms, *etc.* In general, such routines would be specific to a given natural language's morphology. That is beyond the scope this research.

The domain match follows the name match, but it too must yield success before the overall comparison is considered to match. Attributes match adequately if they have similar data types (*e.g.*, float-type and fixed type), match better if the data types are identical, and match the best if they specify similar sets of values (*e.g.*, minima and maxima, or similar lists of symbols).

The system verifies as much as it can without bothering the user by integrating both read and matched domain information. If a match was found then the user is asked which description is more accurate. After the single most accurate attribute description is selected, either explicitly by the user or implicitly by lack of matching alternative, the user may accept this attribute's domain and "describes attribute" (see below) and go on to consider the next attribute.

Users who do not accept the system's attribute description must revise both the attribute's domain and its "describes attribute". The "describes attribute" tells the other tuple value that is described by a given attribute. By default all attributes describe the process, but this is not always the case. For example **age**, **sex**, **admit_health** and **discharge_health** are all generally considered properties of **person**, not of the rehabilitation process itself.

Users are asked to revise domains in the following manner. The program shows the properties of the user's explicitly or the system's implicitly closest attribute. The user is then allowed to change the properties. In our running example, attribute **start_time** should be recast to an ordinary integer domain, with minimum value 1 and maximum value 365. Figure 2 shows the interactive dialog for this revision.

FIGURE 2. Revision of attribute *start_time* of Table 1. (Text in boldface is supplied by the user.)

```
Attribute start_time has an int domain from 10 to 30:
  Attribute describes the process.
    Please choose one of the following:
    (0) Accept as-is
    (1) Accept after changing parameters
    (2) Cast to int-coded-type
    (3) Cast to fixed-type
    (4) Cast to float-type

    Your choice? 1
        The current lo value is 10, new value? 0
        The current hi value is 30, new value? 365
    Does this attr describe another attr instead of a
process (Y/N)? N
```

States are annotated after the attributes. Time helps to uniquely identify states because it is their primary (key) attribute. The program asks which attributes signify the starting time, stopping time and duration of the process. Starting and stopping times are later associated with the process's starting and stopping states. The duration is associated with the process itself. The program asks if there are any other attribute pairs that form <before,after> pairs. In this example there is one: **admit_health** and **discharge_health**. The program then asks for a name that represents the quantity in both states. In this case, it is just **health**.

Creating the Prolog program

The last step of the table interpreter is to re-read the file and format it into a Prolog program segment that contains this information. Three distinct ground clauses exist to specify attribute identities and domains, to tell which symbolic and integer-coded domains are ordered, and to specify the "describes attribute" of any attribute that does not describe the process.

States and "timeless" facts have distinct ground clauses. Values of attributes associated with a state are written on that state. Such values may describe the state itself (*e.g.*, time) or another entity (*e.g.*, the health of the person at a particular time). Values of attributes that are not associated with states are considered "timeless facts" that are true in all states.

Intelligent attribute construction

The next step is to run the constructed program with a Prolog interpreter. There are two components to the con-

structed program: the newly-created attribute and data describing file (see above) and the pre-existing ontology.

The ontology organizes knowledge hierarchically by stating more specific facts further from the root. All knowledge is organized into <object, attribute, value> tuples. The ontology uses directly stated <object, attribute, value> tuples, inherited <attribute,value> pairs, decision trees and simple equations to answer queries.

The value of ontologies derives from our abilities to query them and re-use them in other domains. Associated with each query is a current state and a list of things that currently may be assumed to hold true. The current state can be blank to specify "query results must hold in all states". The "may assume" list is a means of both augmenting the knowledge in states (e.g., "Given that X is true in state S, what is the value of V?") and of obtaining the information recorded in states by holding key values that must be matched (e.g., the knowledge "Assume time = T_1" allows the knowledge at any state with time T_1 to be used).

Legal queries may ask:

1. for the value, given the object and attribute,

2. for the attribute and value, given the object, and

3. for the object, given the attribute and value.

Subsequent queries for a value given the same object and attribute may return other values, but the first value should always be trusted more than subsequent ones.

Attributes can be constructed from the following generic rules:

TABLE 2. Domain non-specific attribute rules

Rule	Applicability	Description
pred(attr)	ordered symbol domain	predecessor of ordered symbol value
succ(attr)	ordered symbol domain	successor of ordered symbol value
delta(attr)	state-dependent numeric domain	change of attribute between start and stop states
rate(attr)	same as delta(attr) AND process duration information	average rate of change of attribute
gen(attr)	symbolic domain with hierarchy info in ontology	generalizes attribute values one level up in hierarchy

If attribute_set[0] represents the original set of attributes and attribute_set[1] represents the new set after applying the rules to attribute_set[0], one can imagine re-applying the rules on attribute_set[N-1] to generate attribute_set[N] for any desired, positive N (e.g., gen(gen(attr)) is in attribute_set[2]).

EXPERIMENTS and DISCUSSION

Recall that the goals of this research are threefold: (1) to automatically suggest and generate new attributes based upon available semantic and domain information, (2) to capture useful knowledge for reuse, and (3) to reduce the user's workload to interpret new tables.

Our experiments directly test these assertions. To test the second and third goals we re-ran the program using table 3 as input to see if the table annotation task is easier. This is measured by the number responses that the user must give. To test the first goal we comment on the usefulness of the output from tables 1, 3 and 5.

After running the program on the orthopedic patient data in table 1, we re-ran it on a database of amputation patients in table 3. Tables 1 and 3 describe distinct sets of people who are from the same rehabilitation service and whose data are recorded in the same database with the same attributes. They were deliberately selected for their similarity, yet there are differences. For example, the average orthopedic patient is 14 years older than the average amputation patient in the larger databases from which tables 1 and 3 were drawn. The data of table 3 has been transformed in the same manner as the data in table 1.

TABLE 3. Amputation patient ward data (abbrev.)

person	age	sex	start _time	dur- ation	admit_ health	dis- charge_ health
p4	38	male	40	12	10.2	10.9
p5	47	male	50	16	10.3	11.7
p6	63	female	60	26	8.7	10.2

Our system was able to transfer knowledge between the databases despite the domain differences. The original run required the user to answer 34 attribute verification questions while the second run only required 19. The system's matching mechanism made attribute annotation easier. Figure 3 shows that only two questions had to be answered for **start_time** instead of the four in figure 2.

FIGURE 3. Revision of attribute *start_time* of Table 3. (Text in boldface is supplied by the user.)

```
Current attribute start_time:
  Attribute start_time has an int domain from 40 to 60
  Attribute describes the process.

Closest stored attribute start_time:
```

```
Attribute start_time has an int domain from 0 to 365
Attribute describes the process.

Is the stored attribute a better description (Y/N)? Y


Attribute start_time has an int domain from 0 to 365
Attribute describes the process.
  Please choose one of the following:
  (0) Accept as-is
  (1) Accept after changing parameters
  (2) Cast to int-coded-type
  (3) Cast to fixed-type
  (4) Cast to float-type

  Your choice? 0
```

The results from tables 1 and 3 are combined in table 4. Our system constructed two new features.

TABLE 4. System results on tables 1 and 3:

person	dura-tion	delta_health	rate_health	age	sex
p1	6	2	0.333333	63	female
p2	8	1.9	0.2375	69	male
p3	9	2.7	0.3	79	female
p4	12	0.7	0.0583333	38	male
p5	16	1.4	0.0875	47	male
p6	26	1.5	0.0576923	63	female

Delta_health is the difference between **admit_health** and **discharge_health**, and was constructed by applying the delta rule of table 2. **Rate_health** is that difference divided by **duration**. It was constructed from **delta_health** and the rate rule. Attributes **delta_health** and **rate_health** have been used by a rule inducing program. The resulting rules were verified by a practicing rehabilitation doctor as having accurately encapsulated some the background rehabilitation knowledge (*e.g.*, younger people heal faster).

As another test database from a very different domain (deliberately chosen for its differences), we entered the seismological data of table 5. These data were obtained

TABLE 5. Table of earthquakes (abbrev.)

latitude	longitude	start_time	duration	scalar moment
30.66	137.06	0.4817314	3.6	1.34e+24
-7.90	109.00	0.733273	6.0	7.72e+24
-10.17	118.99	1.4135231	12.0	3.07e+24

from Harvard University's Centroid Moment Tensor (CMT) catalog. **Latitude** and **longitude** are in degrees. Time is in fractions of a day since the midnight 1976 December 31. **Duration** is twice the recorded "half duration" of the quake, which is itself an estimated value. It is given above in seconds for understandability. It was, how-

ever, entered into the initial database in fractions of a day. Finally, **scalar_moment** is a measure of the energy released by the earthquake in dyne-cm.

The system's output for the data of table 5 is shown in table 6.

TABLE 6. System results of data from table 5

latitude	longitude	dura-tion	delta_scalar_moment	rate_scalar_moment
30.66	137.06	3.6	1.34e+24	3.21605e+28
-7.90	109.00	6.0	7.72e+24	1.11169e+29
-10.17	118.99	12.0	3.07e+24	2.21041e+28

The seismological experiment demonstrates our three goals clearly. Although it was a very different domain, starting time domain information was transferred between the rehabilitation and the seismological domains. A slight reduction in cognitive load was achieved by having the system correctly guess and suggesting some of the domain properties as shown in figure 4. Finally, the system created useful attributes. The quantity that it calls **rate_scalar_moment** is inversely correlated with the complexity of the focal mechanism of the quake. Quake focal mechanism complexity is attributed to the several faults breaking in series, perhaps in slightly different directions. This in turn can be due to networks of interlacing and parallel faults. Hence, our tool has found a potentially very useful attribute for KDD.

FIGURE 4. Knowledge transfer between domains

```
Current attribute start_time:
Attribute start_time has a floating pt. domain from
0.481731 to 1.41352 (mantis=24, exp=5)
  Attribute describes the process.

Closest stored attribute start_time:
Attribute start_time has an int domain from 0 to 365
Attribute describes the process.

Is the stored attribute a better description (Y/N)? Y

Attribute start_time has an int domain from 0 to 365
Attribute describes the process.
  Please choose one of the following:
  (0) Accept as-is
  (1) Accept after changing parameters
  (2) Cast to int-coded-type
  (3) Cast to fixed-type
  (4) Cast to float-type
  Your choice? 4
     Current lo value 0, new value? 0
     Current hi value 365, new value? 365
     Please enter mantissa size: 24
     Please enter exponent size: 7
Does this attr describe another instead of a process
(Y/N)? N
```

128

As we have shown above, we are on track to achieve our goals. The annotation task has been eased. Domain knowledge has been transferred across table interpreting tasks. Also, we have constructed useful attributes.

Transfer between such dissimilar domains was achieved in part by using databases that were on similar time scales (in this case, on the scale of days in a year). We plan to make use of units and dimensionality information to make it more robust to different scales.

Data transforming and feature construction are only two subtasks in the larger KDD effort. Many, perhaps most, of the constructed features may not prove useful. This, however, is an issue for the data mining or machine learning tool to address. Our mission here is to create a smaller set of features than a purely naive attribute constructor would, where the fraction of potential useful attributes has been increased over the naive approach's baseline.

The complexity of the expressions generated by the ontology's rules limit this approach. The table interpreter is not a bottleneck: it makes two passes through the data and keeps no records other than gross statistical measures. However, the Prolog program currently grows linearly with the size of the initial database. We plan to address this by using a Prolog interpreter outfitted with a database for efficient retrieval of ground clauses.

Our ontology consisted of 147 inference rules and associated ground clauses. This small ontology served our purposes as a proof-of-concept, since we believe that the same kinds of information are readily available in most fully developed ontologies.

CONCLUSION

KDD is an inherently iterative process, and our tool accelerates our turn-around time between iterations. Table annotation is facilitated by a tool that intelligently guesses attribute domains based upon the given values, examples from other tables, and selected user querying. Feature construction is supported by using the semantic and domain constraints obtained from the first step to guide the creation of selected attributes.

This system is meant to be a subcomponent in the overall KDD process. Its usage of knowledge obtained from prior examples makes it applicable when several related databases are examined. Still, there is a (lessened) need for an information gain or other usefulness-based metric to prune created attributes to offset their eventual combinatorial explosion. Our approach assumes the same term has equivalent meanings in different settings and that someone

will identify and organize the useful terms. Most ontology efforts share these assumptions.

This work can be extended in a variety of ways. Domain identification may be given a more clever attribute name matching routine, may use unit and dimension information for attributes, *etc.* Feature construction may be made more specific by being able to turn specific rules on and off by specifying particular domains. Prolog may be given the ability to read ground clauses from a database. We believe that this approach convincingly addresses a pressing KDD need.

REFERENCES

[1] Boose, J. "Uses of repertory grid centered knowledge acquisition tools for knowledge based systems." *International Journal of Man Machine Studies*, 29: 287 (1988).

[2] Boose, J. "A survey of knowledge acquisition techniques and tools." *Knowledge Acquisition*, 1(1); 3-37 (1989).

[3] Carbonara, L., Sleeman, D. "Effective and efficient knowledge base refinement." *Machine Learning*, 37: 143-181 (1999).

[4] Craw, S., Boswell, R. "Debugging knowledge-based applications with a generic toolkit." *Proceedings of the 12th International Conference on Tools with Artificial Intelligence (ICTAI-2000)*, IEEE Press, Vancouver, Canada (2000).

[5] Donoho, S. and Rendell, L. "Constructive induction using fragmentary knowledge." In L. Saitta, editor, *Proceedings of International Conference on Machine Learning* (ICML-96), 113-121. Morgan Kaufmann Publishers (1996).

[6] Erikson, H., Fergerson, R. W., Shuval, Y., Musen, M. "Automatic Generation of Ontology Editors." *Technical Report SMI-1999-0809*, Stanford University Section of Medical Informatics (1999).

[7] Fayyad, U., Piatetsky-Shapiro, G., Smyth, P. "From data mining to knowledge discovery: an overview." In Fayyad, U., Piatetsky-Shapiro, G., Smyth, P., Uthurusamy, R. (eds.) *Advances in Knowledge Discovery and Data Mining.* Menlo Park, CA: AAAI Press (1996).

[8] Fikes, R., Farquhar, A. "Large-Scale Repositories of Highly Expressive Reusable Knowledge." Knowledge Systems Laboratory, KSL-97-02, Stanford University (1997).

[9] Gaines, B.R. and Shaw, M.L.G. "Integrated knowledge acquisition architectures." *Journal for Intelligent Information Systems* 1(1) 9-34 (1992).

[10] Gaines, B., Shaw, M. "Knowledge acquisition tools based on personal construct psychology." *The Knowledge Engineering Review*, 8(1), (1993).

[11] Lenat, D. B. "Cyc: A Large-Scale Investment in Knowledge Infrastructure." *Communications of the ACM* 38, no. 11 (1995).

[12] Murphy, P. M., Pazzani, M. J. "Revision of production system rule-bases." *Proc. ICML 11th International Conference on Machine Learning*, 199-207. San Mateo, CA: Morgan Kaufmann (1994).

[13] *National Library of Medicine Unified Medical Language Systems Knowledge Sources, 9th Ed.* U.S. Dept. of Health and Human Services, National Institutes of Health, National Library of Medicine (1998).

[14] Ourston, D., Mooney, R. "Theory refinement combining analytical and empirical methods." *Artificial Intelligence*, 66: 273-309 (1994).

[15] Parson, R., Khan, K., Muggleton, S. "Theory recovery" *Inductive Logic Programming Lecture Notes in Artificial Intelligence*, 257-267. Berlin: Springer-Verlag (1998).

[16] Phillips, J. *Representation Reducing Heuristics for Semi-Automated Scientific Discovery*, Ph.D. Thesis, University of Michigan (2000).

[17] Provost, F. J. Buchanan, B. G. "Inductive Policy, The Pragmatics of Bias Selection." *Machine Learning*, 20: 1/2, 35-61 (1995).

[18] Srinivasan, A. and King, R. D. "Feature construction with Inductive Logic Programming: a study of quantitative predictions of biological activity aided by structural attributes." *Data Mining and Knowledge Discovery*, 3(1):37-57, (1999).

[19] Wilkins, D. "Knowledge base refinement as improving an incorrect and incomplete domain theory." In *Machine Learning: An Artificial Intelligence Approach, Vol. III*, 493-513. San Mateo, CA: Morgan Kaufmann (1994).

[20] Woodward, B. "Knowledge engineering at the front-end: defining the domain." *Knowledge Acquisition* 2(1) 73-94 (1990).

A Methodology for Ontology Integration

Helena Sofia Pinto & João P. Martins

Grupo de Inteligência Artificial
Departamento de Eng. Informática
Instituto Superior Técnico
Av. Rovisco Pais, 1049-001 Lisboa, Portugal
sofia,jpm@gia.ist.utl.pt

Abstract

Although ontology reuse is an important research issue only one of its subprocesses (merge) is fairly well understood. The time has come to change the current state of affairs with the other reuse subprocess: integration. In this paper we describe the activities that compose this process and describe a methodology to perform the ontology integration process.

INTRODUCTION AND MOTIVATION

Ontologies aim at capturing static domain knowledge in a generic way and provide a commonly agreed upon understanding of that domain, which may be reused and shared across applications and groups [4]. Therefore, one can define an ontology as a shared specification of a conceptualization. Ontology reuse is now one of the important research issues in the ontology field. There are two different reuse processes [18]: *merge* and *integration*. Merge is the process of building an ontology in one subject reusing two or more different ontologies on that subject [18]. In a merge process source ontologies are unified into a single one, so it usually is difficult to identify regions in the resulting ontology that were taken from the merged ontologies and that were left more or less unchanged.[1] It should be stressed that in a merge process source ontologies are truly different ontologies and not simple revisions, improvements or variations of the same ontology. Integration is the process of building an ontology in one subject reusing one or more ontologies in different subjects[2] [18]. In an integration process source ontologies are aggregated, combined, assembled together, to form the resulting ontology, possibly after reused ontologies have suffered some changes, such as, extension, specializa-

---

[1] In some cases, knowledge from merged ontologies is homogenized and altered through the influence of one source ontology on another (is spite of the fact that source ontologies do influence knowledge represented in the resulting ontology). In other cases, knowledge from one particular source ontology is scattered and mingled with knowledge that comes from the other sources.

[2] The subjects of the different ontologies may be related.

tion or adaptation. In an integration process one can identify in the resulting ontology regions that were taken from the integrated ontologies. Knowledge in those regions was left more or less unchanged. It should be noted that both reuse processes are included in the overall process of ontology building.

A lot of research work has been conducted under the merge area. There is a clear definition of the process [21], operations to perform merge have been proposed [16, 25], a methodology is available [8] and several ontologies have been built by merge [22, 8]. The first tools to help in the merge process are now available [16, 14]. In the integration area a similar effort is now beginning. The most representative ontology building methodologies [24, 13, 7] recognize integration as part of the ontology development process, but none really addresses integration. Integration is only recognized as a difficult problem to be solved. They don't even agree on what integration is: for some it is an activity, for others a step. We have been involved in two integration experiences where publicly available ontologies were reused: we built the Reference ontology [1, 19, 17] and we were involved in building some of the subontologies needed to build an Environmental Pollutants ontology (EPO), namely the Monoatomic Ions ontology [19, 17, 10].

We have found that integration is far more complex than previously hinted. It is a process of its own [17, 19]. Other important conclusions are that integration takes place along the entire life cycle and should begin as early as possible in the ontology building life cycle so that the overall ontology building process is simplified [17, 19]. In both our experiences, integration began as early as the conceptualization phase.

In this article we begin by describing our assumptions and some terminology. Then we discuss and analyze the integration process in relation to the overall ontology building process. Finally, we present our methodology, namely we describe each ontology integration activity and the methods, guidelines and procedures developed to perform them. As far as we know, this is the first integration methodology proposed in the area.

ASSUMPTIONS AND TERMINOLOGY

Ontology building is a process composed of several activities. Some are performed at particular stages: specification, conceptualization, formalization, implementation and maintenance. Others take place along the entire life cycle: knowledge acquisition, documentation and evaluation. The development of an ontology follows an evolving prototyping life cycle [6]. Since integration is a process that takes place along the entire life cycle, integration activities can take place for one ontology in any stage of the ontology building process.

The aim of the conceptualization phase is to describe in a conceptual model the ontology that should be built. We assume that, in this phase of *any* ontology building process questions like the following are answered: (1) what should be represented in the ontology? (2) how should it be represented (as a class, relation, etc.)? (3) which relation should be used to structure knowledge in the ontology? (4) which structure is the ontology going to have (graph, tree, etc.)? (5) which ontological commitments and assumptions should the ontology comply to? (6) which knowledge representation ontology should be used? (7) should the ontology be divided in modules? (8) in which modules should the ontology be divided?

At conceptualization, one uses knowledge level [15] representations of ontologies. Usually, only implemented ontologies are publicly available at ontology libraries. If the knowledge level representation of an ontology is not available, then an ontological reengineering process [10] can be applied. This process returns one possible[3] conceptual model of an implemented ontology. When one begins integration as early as conceptualization, one needs the ontologies that are going to be considered for integration represented in an adequate form. Any conceptual model representation is adequate. In our case, we had access to knowledge level representations of most reused ontologies as proposed by METHONTOLOGY [6]: $(KA)^2$ [2] to build the Reference ontology and Chemicals [7] to build the Monoatomic Ions subontology of EPO. In the case of $(KA)^2$ and Chemicals we had access to the actual conceptual models that produced their Ontolingua versions, but in the case of EPO, a reengineering process was applied [10] to produce one conceptual model of Standard Units [12] (which is reused by Chemicals). However, any knowledge level representation would be appropriate. Moreover, due to the particular framework that was used, ODE [7], all of our work was done at the knowledge level. This simplified the overall process of integration a lot. Since in conceptualization much of the design of the ontology is specified, it is considerably more difficult to try to integrate an ontology at the implementation phase because, unless one has prior knowledge of the ontologies available for reuse, available ontologies will rarely match the needs and the conceptual model found for the resulting ontology. One of the consequences of this conclusion is that more integration effort should be made at the earliest stages, specially in conceptualization and formalization, than at final ones, implementation or maintenance [19]. We would like to point out that in both our experiences there was no need to translate ontologies between different knowledge representation languages. Translation of ontologies is a very important and difficult problem to be solved in order to allow more generalized reuse of ontologies.

For us, an ontology consists of: classes, instances, relations, functions and axioms. Each one of the components of an ontology is generically referred to as a *knowledge piece*. Each knowledge piece is associated with a name, a documentation and a definition.

A METHODOLOGY

As any process, integration is composed of several activities. We have identified the activities that should take place along the ontology building life cycle to perform integration. All integration activities assume that ontology building activities are also performed, that is, the integration process does not substitute the ontology building process, it rather is a part of it. We now describe each activity and the methods, guidelines and procedures developed to perform them. Examples from case studies are partially described in [17, 19]

Identify integration possibility
The framework being used to build the ontology should allow some kind of knowledge reuse. For instance, the Ontolingua Server maintains an ontology library and allows integration operations, such as inclusion or restriction. More general systems, such as KACTUS, do not allow such kind of operations, but allow pre-existent ontologies to be imported and edited. In other cases, integration (or any kind of reuse) may involve rebuilding an ontology in a framework different from the one where the ontology is available. In some cases, this may be cost-effective, in others it may be more cost-effective to build a new ontology from scratch that perfectly meets present needs and purposes than to try to rebuild and adapt a pre-existent ontology.

Identify modules
The modules (building blocks) needed to build the future ontology are identified, that is, the subontologies in which the future ontology should be divided (in integration, the modules are obviously related to ontologies). In integration upper-level modules and domain modules are identified. Representation ontologies are chosen in any ontology building process, therefore they are not specifically addressed in integration.

Identify assumptions and ontological commitments
One needs to identify the assumptions and ontological commitments [11] that *each* module should comply to. They are

---

[3] This process may not produce the actual conceptual model that originated the final ontology. Moreover, if the conceptual model found for the ontology after the reverse engineering step shows some deficiencies, it may be improved through a restructuring step.

described in the conceptual model and in the specification requirements document of the future ontology. This is one of the activities where documentation of an ontology can be crucial to allow better, faster and easier reuse. The assumptions and ontological commitments of the building blocks should be compatible among themselves and should be compatible with the assumptions and ontological commitments found for the resulting ontology.

Identify knowledge to be represented in each module
One needs to identify what knowledge should be represented in each building block. At this stage, one is only trying to have an idea of what the modules that will compose the future ontology should "look like" in order to recognize whether available ontologies are adequate to be reused. At this stage one only identifies a list of essential concepts. The conceptual model of the ontology and abstraction capabilities are used to produce this list.[4]

Identify candidate ontologies
This is subdivided into: (1) *finding available ontologies*, and (2) *choosing from the available ontologies which ones are possible candidates to be integrated*. To find possible ontologies one uses ontology sources. Since available ontologies are mainly implemented ones one should look for them in ontology libraries, as for instance, in the Ontolingua Server (http://WWW-KSL-SVC.stanford.edu:5915) for ontologies written in Ontolingua, in Ontosaurus (http://www.isi.edu/isd/ontosaurus.html) for ontologies implemented in Loom or in the Cyc Server (http://www.cyc.com) for Cyc's upper-level ontology. Conceptualized or formalized ontologies are more difficult to find. Sometimes they are available in the literature or can be obtained by contacting ontology builders. However, not every ontology in a given subject will be appropriate to be reused (some may lack some important concepts, etc.).

To choose candidate ontologies one analyzes all available ontologies according to a series of features. At this stage of the ontology integration process one does not want to leave out any possible candidate. Therefore, only a very general analysis is made. Some of the features are *strict requirements*: (1) domain, (2) is the ontology available? (3) formalism paradigms in which the ontology is available, (4) main assumptions and ontological commitments, (5) main concepts represented. If the ontology does not have adequate values for these properties they cannot be considered for integration. Therefore, these properties are used to eliminate ontologies. Some of these features can only be analyzed at a qualitative level (main concepts represented, main assumptions and ontological commitments). Other features are *desirable requirements* or desirable information: (1) where is the ontology available? (2) at what level is the ontology

---
[4] At later stages one will need to know to what level of detail should that knowledge be represented, which relations should organize (structure) the ontology, and it would be helpful to know how it should be represented (concept, relation, etc.).

available? (3) what kind of documentation is available (technical reports, articles, etc.)? (4) where is that documentation available? If some of the properties have certain values, the ontology is a better candidate: if the knowledge level representation of an ontology is available, then this ontology is a better candidate since the reengineering process would not have to be performed, if the internal and external documentation is available, then the most relevant information about the construction and choices made during the construction of the ontology is available, but if only articles are available about the ontology, then it is likely that some of the choices are not explained. If all of the values of these properties are unknown, that is, if one cannot find where the ontology and the documentation is available, then one cannot reuse it, therefore, the ontology is not a candidate. However, if there is enough documentation available, then it may be possible to reconstruct the ontology, and if the ontology is available, then it may be possible to understand it, provided that the domain is common enough and the ontology is simple and not very large (and possibly after some knowledge acquisition). One can use a very simple *metric* to combine these features. If strict requirements do not have adequate values, the ontology is eliminated. If desirable requirements have appropriate values, then the ontology is a better candidate. If none of the desirable requirements have appropriate values, then the ontology is not a candidate. One does not want to eliminate any possible candidate at this stage of the process, only those that are of no use at all. If, in a particular integration process, other features should be considered while choosing candidate ontologies, the metric can be easily updated. One only has to decide whether the features are strict or desirable requirements. For instance, one can impose the condition that only already evaluated ontologies should be considered as candidates. In that case, one should add this feature as a strict requirement. If one only wishes to prefer already evaluated ontologies, then this feature should be added as a desirable requirement. The advantage of the flexibility of this metric is the fact that it can be adapted to integration processes that should take into account particular features during the choice of one ontology. In particular, this kind of changes can narrow down the possible ontologies to choose from, if one introduces more strict requirements.

Get candidate ontologies
Getting candidate ontologies in an appropriate form includes, not only, their *representations*, but also, all available *documentation*. As already discussed, one should prefer to work with the knowledge level representation of an ontology. In some cases, this representation can be found in the literature (technical reports, books, thesis, etc.), or at least parts of it. Another possibility is contact ontology developers. However, in most cases, only the implementation level representation of an ontology is available. Therefore, the *reengineering process* may be applied using the particular framework that was adopted to design the resulting ontology. If the ontology is not available (either at the implementation or knowledge

level'), one can still try to reconstruct it, or, at least, parts of it, using available documentation. While getting the implementation level representation of an ontology, if the ontology is not written in the adequate language (the language chosen to represent the resulting ontology) a *knowledge translation process* must take place. There are only a few translation attempts. Translation is far from being a fully automatic process. In general, there are not many translators available, their technology is still immature and improving existing translators is a rather difficult task. If translators are available they should be used to produce initial versions. Then, these initial versions should be improved by hand. Translators between different knowledge level representation languages are currently not available. The translation process is, in general, complex. It is important that, if the ontology includes other ontologies, one should also get the included ontologies. When reusing/using one ontology one must understand it fully, which includes every definition of every knowledge piece represented in the ontology. Included ontologies are a part of the ontology. Knowledge pieces from included ontologies can be used in the definitions of the ontology, therefore, in order to understand the ontology and know what one knowledge piece defined in an included ontology means one must have access to its definition or its technical documentation.

Study and analysis of candidate ontologies

To study and analyze candidate ontologies we must perform two activities: (1) *technical evaluation* of candidate ontologies by ontologists through specialized criteria oriented to integration and (2) *user assessment* of candidate ontologies by domain experts through specialized criteria oriented to integration. Both domain experts and ontologists should evaluate and assess as whole and all candidate ontologies. To technically evaluate candidate ontologies domain experts should pay special attention to [17, 19]: (1) what knowledge is missing (concepts, relations, etc), (2) what knowledge should be removed, (3) which knowledge should be relocated, (4) which knowledge sources changes should be performed, (5) which documentation changes should be performed, (6) which terminology changes should be performed, (7) which definition changes should be made, (8) which practices changes should be made. Since domain experts usually find the languages used to implement ontologies difficult to understand [7], they should preferably be given a knowledge level representation of the ontology. To user assess candidate ontologies gists should pay special attention to [17, 19]: (1) the overall structure of the ontology to assess whether the ontology has an adequate (and preferably well-balanced) structure, adequate and enough modules, adequate and enough specialization of concepts, adequate and enough diversity, similar concepts are represented closer whereas less similar concepts are represented further apart, knowledge is correctly "placed" in the structure so that inheritance mechanisms can infer appropriate knowledge from the ontology, etc.; (2) the distinctions (classification criteria made of the concepts described in the

ontology) upon which the ontology is built to assess whether they are relevant and exactly the (quantity and quality) required; (3) the relation used to structure knowledge[5] in the ontology to assess whether it is the required one; (4) the naming convention rules used to assess whether they ease and promote reuse; (5) the quality of the definitions (do they follow unified patterns, are simple, clear, concise, consistent, complete, correct — lexically and syntactically — precise and accurate); (6) the quality of the documentation of the ontology; (7) the knowledge pieces represented (or included) are the ones that should be represented and all appropriate knowledge pieces are represented.

Choosing source ontologies

At this stage, and given the study and analysis of candidate ontologies performed by domain experts and ontologists, the final choices must be made. Among the candidate ontologies the ones that passed strict requirements and among those that scored best in integration-oriented technical evaluation and user assessment one has to choose the source ontology (or set of source ontologies) that best suit our needs and purpose. Once again, the ontology(ies) chosen to be reused may lack knowledge, may require that some knowledge is removed, etc., that is, it may not exactly be what is needed. The best candidate ontology is the one that can better (more closely) or more easily (using less operations) be adapted to become the needed ontology. This choice also depends to some extent on the other ontologies that are going to be reused since in an integration process one can reuse more than one ontology. It is important that reused ontologies are compatible among themselves, namely in what concerns overall coherence. Sometimes, one can choose more than one ontology in a given domain if each one focuses different points of view of that domain. This is a rather complex multi-criteria choice where a lot of different aspects are involved. Since the choice of source ontologies is much more complex than choosing candidate ontologies, we propose that it should be divided into two stages.

In the **first stage** one tries to find which candidate ontologies are best suited to be integrated. Domain expert and ontologist analyses are crucial in this process. We propose that candidate ontologies should be analyzed according to a taxonomy of features, Figure 1.

General features give general information about the ontology. *Development status* gives information about the degree of readiness of an ontology to be reused (intended, on-going, toy example, implemented, mature). A toy example only contains representative knowledge pieces. An implemented ontology can be a good candidate if it has been carefully built or it has been evaluated. A mature ontology used in applications is a good candidate. The ontology should be a more or less stable ontology (provided that the domain does not evolve very rapidly).

---

5 An ontology can be thought of as structured or organized according to one privileged relation, for example, ISA, part-of, etc.

- general
  - type (general, domain)
  - formality
  - development status
- development
  - knowledge acquisition
    - quality of knowledge sources
    - adequacy of knowledge acquisition practices
  - maintenance
    - is it maintained?
    - who does maintenance?
    - how is maintenance done?
  - documentation
    - quality of the documentation available
    - is the available documentation complete?
  - implementation
    - language issues
      - language(s) in which it is available
      - translators: are there translators? for which languages? quality of those translators
      - properties needed of the KR system in which it is built
- content
  - level of detail
  - modularity
  - adequacy from the domain expert point of view
  - adequacy from the ontologist point of view

Figure 1: Choosing source ontologies, first stage

Development features are related to how the ontology was built. The *quality of knowledge sources* and *adequacy of knowledge acquisition practices* are analyzed during the domain expert integration-driven technical evaluation. The ontology should be *maintained*. One interesting finding about ontologies is the fact that they evolve, are "living", since their domains also evolve. Therefore, if they are maintained, it is most likely that they are updated. Maintenance policies differ in *who* changes the ontology (can anybody change the ontology, or only authorized personnel?) and *how* those changes are performed (is the ontology changed regardless of people that built it, use it or reuse it? are the suggestions of change previously discussed among those groups? is there any attempt to reach a consensus between groups? is there a special board that decides upon suggestions for changes?). The *documentation* should have enough *quality* (it is clear, it describes the domain, the ontology, the alternative representations and the preferred alternatives) and is *complete* (the ontology is completely described). If the ontology is available in the required *language* the task is greatly simplified (translation is avoided). Otherwise, it is important to know whether *translators* from those languages are available, *for which languages* and their *quality*. One needs to know which *reasoning capabilities* are required by the ontology from the knowledge representation system where it is implemented, in order to know whether it can be represented under a different knowledge representation system. Full translation between different knowledge representation systems may not be possible. For instance, while translating an ontology represented in first order logic into a pure frame system, if axioms are represented, they are lost. Therefore, one needs to know, among other issues: (1) *formalism paradigm* (frames, semantic networks, description logics, etc.), (2) needed *inference mechanisms* (general purpose, automated concept clas-

sifier, inheritance,[6] monotonic vs modal vs nonmonotonic), 3) are *contexts* required?

Content features give information about what is represented in the ontology and how that knowledge is represented. One needs to know whether the ontology has an adequate *level of detail* (enough intermediate concepts are represented between two arbitrary concepts) and *which concepts are represented in which modules*. Under the feature *adequacy from the domain expert point of view* several analyses are made: does the content of the ontology include most of the relevant knowledge pieces of the domain? is the terminology adequate? are the definitions adopted correct and widely accepted? is the ontology complete in relation to present needs (at least, one needs to know what important knowledge pieces are missing)? is there superfluous knowledge that should be removed from the ontology while integrating it? Under the feature *adequacy from the ontologist point of view* several analyses are made: are the basic distinctions represented in the ontology appropriate? does the ontology have an adequate structure? is the ontology structured according to appropriate relations? are needed knowledge pieces represented (including the appropriate relations, and certain key concepts)? are those knowledge pieces adequately represented (this covers issues like fidelity, minimal encoding bias, correction, coherence, granularity, conciseness, efficiency in terms of time and space[7])? do they follow adequate naming convention rules? can missing knowledge pieces be added to the ontology without sacrificing coherence and clarity (extendible)? is the ontology clear?

The preponderant parts in this choice are played by the adequacy analyses that domain experts and ontologists have made of candidate ontologies. Since this choice is rather complex, simple metrics as the ones proposed to choose candidate ontologies are rather limited. The development of more accurate metrics is an open research area in the OE field. After the first stage, one has chosen one possible set of ontologies to be integrated. It may be possible to have more than one ontology about one particular domain in that set. Those different ontologies represent knowledge about the domain from different perspectives. Those different perspectives should have been found important to be present in the resulting ontology (there should not be duplicated knowledge represented in the resulting ontology). However, chosen ontologies may not be compatible among themselves.

In the **second stage** one tackles compatibility and completeness of possibly chosen ontologies in relation to the desired resulting ontology, Figure 2. If the ontologies which are possibly going to be chosen are not coherent in what concerns the terminology and the definitions of the concepts that are common to more than one ontology, then they are not *compatible* and, therefore, cannot be assembled. Sometimes the

---

[6] Which kind?: defeasible, strict, mixed; credulous vs skeptical; on-path vs off-path; bottom-up vs top-down.

[7] One needs to know if we are reusing an ontology that is not going to meet our needs and the means that we currently have at our disposal.

- compatibility
  - terminology of common concepts
  - definitions of common concepts
- completeness

Figure 2: Choosing source ontologies, second stage

same concept is named differently in different ontologies. In the resulting ontology one concept only has one denomination, therefore one must be adopted. If one concept has the same definition in all chosen ontologies but different denominations, then a change in terminology can solve the problem. All definitions involving the renamed concept have to be checked and revised accordingly. Sometimes different ontologies adopt different definitions for the same concept. One cannot have this kind of inconsistencies in the resulting ontology. One definition should be chosen and adopted all over. It is more difficult to ensure that the same definition can be adopted by all integrated ontologies. A thorough analysis of all ontologies where one particular concept has a different definition from the adopted one has to be made. It is obvious that only a coherent set of ontologies should be considered for integration purposes. If chosen ontologies are not compatible among themselves, then this may imply choosing another possible set of ontologies by combining candidate ontologies into a different set, or it may imply building ontologies from scratch (if none of the candidate ontologies adopts the adequate terminology and definitions, or profound changes have to be made to them in order to integrate them). If chosen ontologies are not *complete*, that is, they do not comprehend all the ontology that has to be built, then this must be known so that missing knowledge pieces are built from scratch and added or another compatible ontology that contains those knowledge pieces is integrated. Since one of the issues involved in the domain expert analysis is missing knowledge, one can check whether it is not represented in another ontology about the same domain that is also (or can also be) integrated.

The problem of choosing the appropriate set of source ontologies is also rather complex. From the set of candidate ontologies, a coherent and adequate subset must be found that is as close as possible to the resulting ontology. Once again, the ontologies in that set may not be perfect candidates. As long as the changes to be made are not very extensive it is more cost effective to reuse the ontologies. This analysis has to be performed on a case by case basis. If it is more cost effective to build the ontology from scratch, then existing ontology building methodologies can be used to build an ontology that perfectly suits our needs. If not, ontologies should be reused and integration operations applied so that adequate changes transform the ontologies into perfect candidates. The result of this activity is a set of ontologies that can and should be assembled together, a description of lacking knowledge that is going to be built from scratch and included in the resulting ontology (since none of the chosen ontologies has it and that
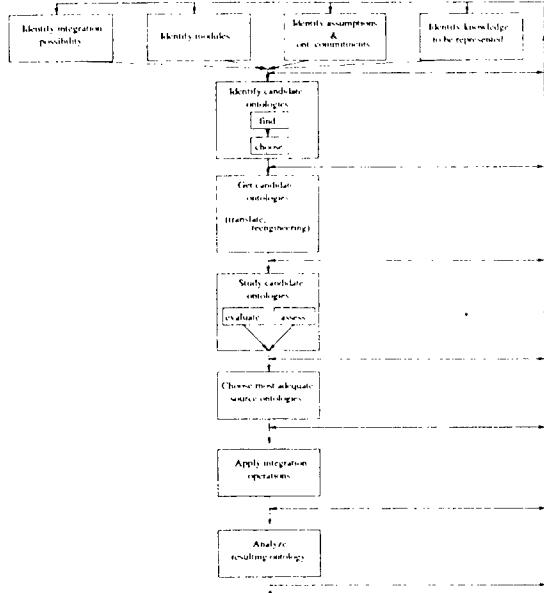
knowledge has been identified as essential knowledge that must exist in the resulting ontology) and a description of the changes that should be performed to the integrated ontologies so that they can be perfect candidates and successfully reused (which is the starting point for the application of the integration operations).

Apply integration operations

All activities described so far precede integration of knowledge from source ontologies into the resulting ontology. They help the ontologist to analyze, compare, and choose the ontologies that are going to be reused. When this part of the process ends, that is the appropriate ontologies to be reused in one particular integration process are found, we must integrate the knowledge of those ontologies. For that, one needs *integration operations* and *integration oriented design criteria*. Integration operations specify how knowledge from an integrated ontology is going to be included and combined with knowledge in the resulting ontology, or modified before its inclusion. These can be viewed as composing, combining, modifying or assembling operations. Knowledge from integrated ontologies can be, among other things, (1) used as it is, (2) adapted (or modified), (3) specialized (leading to a more specific ontology on the same domain) or (4) augmented (either by more general knowledge or by knowledge at the same level). Sometimes the adaptation of ontologies may require restructuring activities similar to those that are performed in reengineering processes. Moreover, it may require introduction/removal of knowledge pieces, correction and improvement of the definitions, terminology and documentation of the knowledge pieces represented in the ontology, etc. These adaptations transform the chosen ontology into the needed ontology. In [5, 3, 19, 17] initial sets of integration operations are proposed. *Integration operations* can be divided into two groups: basic and non-basic. While the former can be algebraically specified the latter can be defined from the former but are custom-tailored operations to be defined in a case by case basis. We have developed an algebraic specification of 39 basic integration operations and specified how 12 non-basic operations can be defined from the previous ones. Design criteria guide the application of integration operations so that the resulting ontology has an adequate design and is of quality. We identified a set of *criteria* to guide integration of knowledge [1]: modularize, specialize, diversify each hierarchy, minimize the semantic distance between sibling concepts, maximize relationships between taxonomies and standardize names of relations.

Analyze resulting ontology

After integration of knowledge one should evaluate and analyze the resulting ontology. Besides having an adequate design [11] and compliance with evaluation criteria [9] the ontology should have a *regular level of detail all over*. By regular level of detail we mean that there are no "islands" of exaggerated level of detail and other parts with an adequate one. None of the parts should have less level of detail than

Figure 3: The integration process

Figure 4: Integration effort along the ontology building process

although sometimes not explicitly or with different levels of importance and effort. All activities, in particular those that precede application of integration operations, should be performed preferably in conceptualization or in formalization stages, that is, before implementation. However, if integration begins later in the ontology development life cycle, they still have to be performed. In both our integration experiences the framework that we used, ODE, automatically generated the implemented versions of the resulting ontologies. Therefore, we performed all integration activities during conceptualization and formalization stages. Using other frameworks may extend the process a bit. If the framework being used does not generate the implementation of the resulting ontology from the conceptual representations, after performing all activities at the knowledge level, the implemented versions of the chosen ontologies must be obtained and then one must apply the already determined sequence of integration operations in order to build the implemented version of the resulting ontology. In this case, only two activities (get ontologies and apply integration operations) had to be performed at the implementation level. This particular process falls into a typical evolving prototyping life cycle. One important aspect of integration is the fact that this process is included in the overall ontology building process. The relation between the integration process and the overall ontology building process is shown in Figure 4. The integration effort is not null during maintenance since integrated ontologies may themselves change due to maintenance activities making it necessary (or desirable) to reapply the integration process.

the required one or else the ontology would be useless, since it would not have sufficient knowledge represented. It should be noted that the features involved in evaluation and design criteria are analyzed in relation to the resulting ontology, for instance, the resulting ontology should be consistent and coherent all over (although composed by knowledge from different ontologies).

DISCUSSION

In Figure 3 we present the activities that compose the ontology integration process. Although ontology building and consequently ontology integration follows an evolving prototyping life cycle, some order must be followed. In general, the activities that compose the integration process tend to be performed following the order by which they were presented. However, some of the activities (and subactivities) to be performed before applying integration operations are interchangeable and some may be even performed in parallel. For instance, integration-oriented technical evaluation and user assessment of candidate ontologies. Moreover, the auxiliary subprocesses, reengineering and translation, may not occur in a particular integration process. If we find an ontology that matches the whole ontology that one needs to build, then one does not need to apply integration operations or analyze the resulting ontology. However, finding candidate ontologies, getting them, their evaluation and assessment for integration purposes, and the choice of the most adequate one remain essential activities to be performed. Finally, one can go back from any stage in the process to any other stage as entailed by the kind of life cycle. The important issue is that these activities are present in any integration process,

CONCLUSIONS

In this article we describe the activities that compose the ontology integration process and present a methodology that provides support and guidance to perform those activities. The advantages of the proposed integration methodology are a direct consequence of its generality. One of the advantages of our integration methodology is the fact that it *can be used with different methodologies to build ontologies from scratch*. The only assumption made by this methodology is that knowledge should be represented at the knowledge level.

Special emphasis is given to the *quality* of the ontologies involved in a particular integration process. Our methodology proposes that all reused ontologies should be evaluated by domain experts from a technical point of view and assessed by ontologists from a user point of view. This assures that reused ontologies have enough technical quality to be used in the process. The analysis of the resulting ontology assures that it has enough quality to be made available and (re)used.

REFERENCES

1. J. Arpirez-Vega, A. Gómez-Pérez, A. Lozano-Tello, H. Sofia Pinto. Reference Ontology and (ONTO)^2 Agent: the Ontology Yellow Pages. *Knowledge and Information Systems*, 2(4):387-412, 2000.

2. V.R. Benjamins, D. Fensel. The Ontological Engineering Initiative (KA)^2. In N. Guarino (ed.), *Formal Ontology in Information Systems*, pages 287-301. IOS Press, 1998.

3. P. Borst, H. Akkermans, J. Top. Engineering Ontologies. *International Journal of Human Computer Studies*, 46(2/3):365-406, 1997.

4. B. Chandrasekaran, J. Josepheson, V.R. Benjamins. Ontologies: What are they? Why do we need them? *IEEE Intelligent Systems*, 14(1):20-26, 1999.

5. A. Farquhar, R. Fikes, J. Rice. Tools for Assembling Modular Ontologies in Ontolingua. In *Proc. AAAI97*, pages 436-441. AAAI Press, 1997.

6. M. Fernández, A. Gómez-Pérez, N. Juristo. METHON-TOLOGY: From Ontological Art Towards Ontological Engineering. In *Proc. of AAAI97 Spring Symposium Series, Workshop on Ontological Engineering*, pages 33-40. AAAI Press, 1997.

7. M. Fernández, A. Gómez-Pérez, A. Sierra, J. Sierra. Building a Chemical Ontology Using METHONTOLOGY and the Ontology Design Environment. *IEEE Intelligent Systems*, 14(1):37-46, 1999.

8. A. Gangemi, D. Pisanelli, G. Steve. Ontology Integration: Experiences with Medical Terminologies. In N. Guarino (ed.), *Formal Ontology in Information Systems*, pages 163-178. IOS Press, 1998.

9. A. Gómez-Pérez, N. Juristo, J. Pazos. Evaluation and Assessment of the Knowledge Sharing Technology. In N. Mars (ed.), *Towards Very Large Knowledge Bases*, pages 289-296. IOS Press, 1995.

10. A. Gómez-Pérez, D. Rojas-Amaya. Ontological Reengineering for Reuse. In D. Fensel, R. Studer (eds.), *Proc. of EKAW99*, pages 139-156. Springer Verlag, 1999.

11. T. Gruber. Towards Principles for the Design of Ontologies for Knowledge Sharing. *International Journal of Human Computer Studies*, 43(5/6):907-928, 1995.

12. T. Gruber, G. Olsen. An Ontology for Engineering Mathematics. In J. Doyle, E. Sandewall, P. Torasso (eds.), *Proc. KR94*, pages 258-269. Morgan Kaufmann, 1994.

13. M. Gruninger. Designing and Evaluating Generic Ontologies. In *Proc. of ECAI96's Workshop on Ontological Engineering*, pages 53-64, 1996.

14. D. McGuiness, R. Fikes, J. Rice, S. Wilder. An Environment for Merging and Testing Large Ontologies. In A. Cohn, F. Giunchiglia, B. Selman (eds.), *Proc. KR2000*, pages 483-493. Morgan Kaufmann, 2000.

15. A. Newell. The Knowledge Level. *Artificial Intelligence*, 18(1):87-127, 1982.

16. N. Noy, M. Musen. PROMPT: Algorithm and Tool for Automated Ontology Merging and Alignment. In *Proc. AAAI2000*, pages 450-455. AAAI Press, 2000.

17. H. Sofia Pinto. Towards Ontology Reuse. In *Proc. of AAAI99's Workshop on Ontology Management*, pages 67-73. AAAI Press, 1999.

18. H. Sofia Pinto, A. Gómez-Pérez, J. P. Martins. Some Issues on Ontology Integration. In *Proc. of IJCAI99's Workshop on Ontologies and Problem Solving Methods: Lessons Learned and Future Trends*, 1999.

19. H. Sofia Pinto, J.P. Martins. Reusing Ontologies. In *Proc. of AAAI2000 Spring Symposium Series, Workshop on Bringing Knowledge to Business Processes*, pages 77-84. AAAI Press, 2000.

20. T. Russ, A. Valente, R. MacGregor, W. Swartout. Practical Experiences in Trading Off Ontology Usability and Reusability. In *Proc. of KAW99*, 1999.

21. J. Sowa. *Knowledge Representation: logical, philosophical and computational foundations*. Brooks/Cole, 2000.

22. B. Swartout, R. Patil, K. Knight, T. Russ. Toward Distributed Use of Large-Scale Ontologies. In *Proc. of AAAI97 Spring Symposium Series, Workshop on Ontological Engineering*, pages 138-148. AAAI Press, 1997.

23. M. Uschold, M. Healy, K. Williamson, P. Clark, S. Woods. Ontology Reuse and Application. In N. Guarino (ed.), *Formal Ontology in Information Systems*, pages 179-192. IOS Press, 1998.

24. M. Uschold, M. King. Towards a Methodology for Building Ontologies. In *Proc. of IJCAI95's Workshop on Basic Ontological Issues in Knowledge Sharing*, 1995.

25. G. Wiederhold. Interoperation, Mediation and Ontologies. In *Proc. of the Intern. Symposium on the 5th Generation Computer Systems, Workshop on Heterogeneous Cooperative Knowledge-Bases*, 1994.

Untangling Taxonomies and Relationships: Personal and Practical Problems in Loosely Coupled Development of Large Ontologies

Alan L. Rector, Chris Wroe, Jeremy Rogers, Angus Roberts
Medical Informatics Group, Department of Computer Science, University of Manchester
Manchester, UK
email {rector|wroec|jrogers|aroberts}@cs.man.ac.uk

Abstract
The GALEN programme has been developing medical ontologies collaboratively for nearly a decade. The ontologies are large and formulated in a specialised description logic, GRAIL. The programme is a broad collaboration of over a dozen groups, most with no prior experience of developing formal ontologies. The programme has developed a methodology for loosely coupled development using layers of intermediate representations, guidelines and tools which minimises training requirements for domain experts and effort by central knowledge engineers.

Issues arise both from problems in formal representations and from the idiosyncrasies of the medical domain. Issues dealt with include 'tangled' taxonomies, part-whole and locative relationships, defaults and exceptions, semantic normalisation, and the difference between medical convention and strict logical criteria for correctness.

Keywords:
Cooperative development; ontology development; ontology design; very large ontologies, medical

INTRODUCTION
The GALEN programme has been developing medical ontologies collaboratively for nearly a decade. (The current versions are available through *Open*GALEN at www.opengalen.org.) The ontologies are large – over 20,000 surgical procedures, nearly 10,000 anatomical concepts and over 10,000 drugs and related notions; the schemas are detailed – between fifty and one hundred families of link types covering different flavours of partonomy and location, function, and causation; and the definitions are complex – a dozen or more conjuncts embedded to four or five levels are not uncommon. They are formulated in a specialised description logic, GRAIL[15-17].

The programme is a loose collaboration which has varied over time from seven to more than a dozen groups. Most of those contributing to the development have little prior experience of building formal ontologies, although many have long experience of developing and or testing medical terminologies. Some were actively sceptical or even hostile to the ideas of formality and standardisation. In the testing phase of the programme, most groups were part of other larger efforts with a their own primary goals, so that the effort available for this project was limited. Training time for most workers had to be confined to no more than six days divided into two workshops.

The same methods have been applied to the development of a large ontology of drugs, their uses, actions, side effects, etc. as part of the UK PRODIGY and Drug Ontology projects [20, 30].

This paper describes the interplay of the different methodological and technical elements which have been brought to bear on this problem and the overall approach and rationale for ontology development which has emerged from it. We view the problems of ontology development as an intimate mix of organisational and technical issues in which different interests and priorities must be reconciled to achieve a successful outcome.

BASIC ELEMENTS OF ONTOLOGY CONSTRUCTION IN GALEN

Goals And Criteria for Correctness
*Open*GALEN aims to produce clinical ontologies which are:

- *Logically correct* and therefore suitable for use in retrieval, rule based systems, etc. For example, all and only "heart diseases" should be classified under "heart disease", all and only procedures on the liver under "liver procedures" etc. Any given concept should be classified in as many ways as appropriate.

- *Reusable* and therefore suitable support system integration, communication etc. The resulting classifications must therefore to contain as fine a grained detail and support as many alternative views as are required by the union of the applications that might reasonably be expected to use them.

139

By contrast most existing medical terminologies, with the exception of SNOMED-RT [22], have been designed for single applications – e.g. bibliographic retrieval, remuneration, or epidemiological reporting – and organised to facilitate access intuitive access by clinicians rather than logical correctness or accuracy of retrieval.

Basis of the approach

*Open*GALEN's requirements are for distributed loosely coupled development of complex ontologies with only modest need for central coordination and limited possibility of central control. The vast majority of the participants are interested in the outcomes rather than the underlying process.

There are four groups who need to participate in the development of an ontology:

- *Content* contributors
- *Domain experts* who capture and quality assure that bulk of the content formally, usually but not always based on some external source of content
- *Knowledge engineers* who design and maintain the formal ontology itself
- *Logicians* who develop and maintain the underlying logic engines and representations

A comprehensive methodology must coordinate the activities of all four groups and provide clean interfaces between them. However, *Open*GALEN concentrates on reconciling viewpoints of different domain experts – often distributed amongst many centres with many different priorities – with those of the knowledge engineers. This distributed approach has led us to a different emphasis from that of authors such as Uschold [26].

The goal has been to allow domain expert's to work as independently as possible with guidelines and agreements which are intuitive at a domain level while, at the same time, allowing the knowledge engineers maximum freedom to develop the underlying description logic ontology.

Domain experts therefore work in tailored 'intermediate representations' which are transformed algorithmically into the underlying description logic based ontology. The use of intermediate representations has a long history in knowledge based systems generally [3], and the use of schemas to create specialist environments for domain experts has many analogies with the approaches of PROTÉGÉ [11, 24, 25]and KADS[27]. However, it has been less widely used in ontology development, although Staab [23] describes the use of a somewhat lower level intermediate representation to separate developers from the details of implementation. Staab's intermediate representation is closer to the level of GRAIL or the rapidly developing interchange language, OIL[1]. It addresses the issues of translating these relatively high level representations into low level expressive description logics such as FaCT [7, 8]. However, our domain experts find even languages at the level of GRAIL or OIL difficult to manage. We therefore envisage the continuing need both

for a knowledge engineering intermediate language at roughly the level of GRAIL, OIL or Staab's representation and for a still higher level user-oriented intermediate representation.

A key aspect of *Open*GALEN's intermediate representations is that they are 'soft' and can be adapted to the requirements of individual sites. An intermediate representation consists of a) a set of user oriented 'descriptors' or terms b) mappings of those terms to concepts in the underlying ontology c) a set of constrained templates providing the links between the descriptors d) a set of transformations between the intermediate representation and expressions in the underlying ontology. Within broad limits, sites can author their ontologies using descriptors and templates tailored to their needs and tastes. All intermediate representations can then be transformed into a common underlying representation [21].

However, the transformation process is not infinitely flexible; some consistency is required from the domain authors. Therefore, in addition to the intermediate representation, guidelines and examples are required for semantic normalisation as described below.

Furthermore, although the intermediate representations are relatively comprehensible, in many applications simple generated pseudo-natural language noun phrases are more compact and familiar and can be adapted to the user's own language. *Open*GALEN has found natural language generation essential to user acceptance. Most language generation is general, but additions to lexicons and grammars are usually required for each new application.

Finally, any real application requires a set of quality assurance criteria and the tools to test them. These should be developed and agreed at the same time as the intermediate representations although, in practice, they often evolve in the course of development.

Development phases

*Open*GALEN therefore divides development of large ontologies into two phases: design and population. In practice, these phases are iterative, but it is easier to describe the processes as if they were sequential.

The Design Phase

In the design phase, knowledge engineers extend the basic ontological schema and prepare user-authors' views or Intermediate Representations. The outputs from the design phase configure two sets of tools, one for the knowledge engineers and one for the domain experts. In the population phase, domain experts populate the ontology using intermediate representations which are transformed into the underlying description logic representation.

The goal of the design phase is to produce five related outputs and incorporate them into a set of tools for the domain experts to use in the population phase as shown in Figure 1:

- *Intermediate Representations* adapted to each major group of domain expert authors' requirements

- *Guidelines* for domain expert authors
- *Schemas* for the underlying ontology, along with transformation rules from the Intermediate Representation to the underlying ontology.
- *Lexicons and Grammars* for natural language generation for display of results to users
- *Quality assurance*(QA) criteria based on the combination of the above three. Ideally quality assurance criteria are set at the time of the original design and modified iteratively, although this ideal is not always achieved in practice.
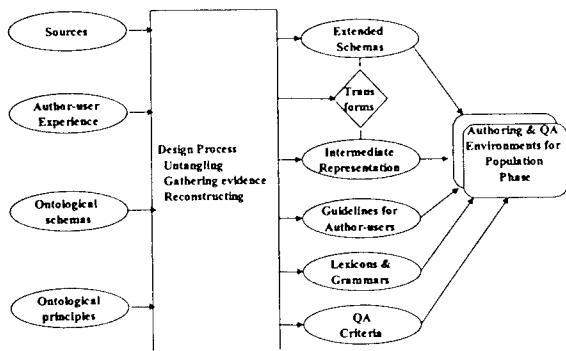
Figure 1: The design phase of OpenGALEN development

The population phase

The population phase is best described by two views: a layered view as in Figure 2 showing the different components and how they interact, and an iterative view as in figure 3 showing the flow of information and interaction between domain experts and the central knowledge engineering team.

In the population phase, domain experts usually work from sources such as existing terminologies or classifications. The first step is to paraphrase the phrases or 'rubrics' from those sources into unambiguous statements to be represented in the intermediate representation. Separating the paraphrase step from the representation step allows quality assurance and discussion of the domain experts' interpretation of the sources to be separated from their representation of those sources in an intermediate representation.

To transform the paraphrase into the intermediate representation and organise the results, the domain experts interact with documentation and tools incorporating the quartet of resources developed in the design phase: guidelines, tools, intermediate representations, and quality assurance criteria. This quartet of resources is linked to the formal ontology through the transforms between the intermediate representation and the underlying formal ontology. (The addition of a classifier at the level of FaCT is likely to produce a further layer of Implementation Logic as shown in grey.)

This methodology provides clear regions of interaction between the various groups involved in the process. The domain expert authors and the knowledge engineers interact over the intermediate representation; the knowledge engineers and logicians over the transformation to the implementation logic and the formal ontology language; the domain experts and the content originators over the paraphrase of the original sources.
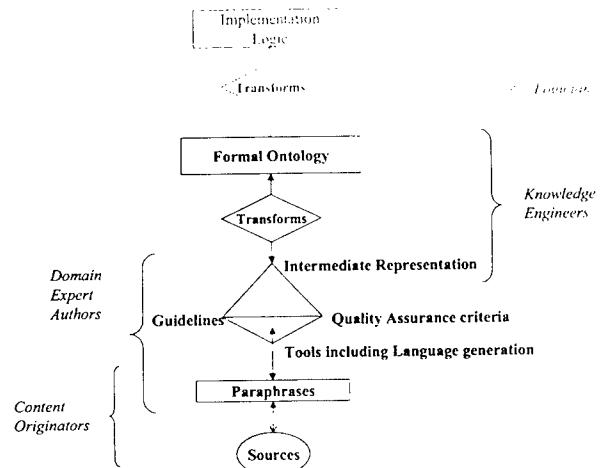
Figure 2: Layering of the population phase of *Open*GALEN development with extrapolation to an additional layer of implementation logic
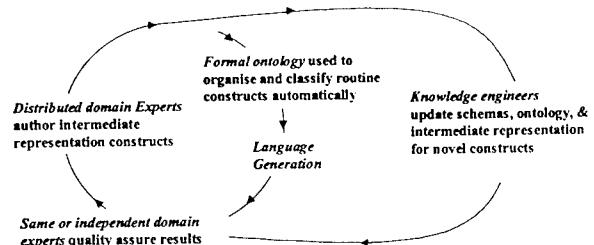
Figure 3: *Open*GALEN Population Cycle

In practice the primary interaction is between the domain experts, who often work in independent units, and the knowledge engineers, who are usually a central resource. In general, the domain experts can author content in the intermediate representation and have it structured, classified, and ready for quality assurance without any intervention from the central knowledge engineers. New concepts in existing categories can be authored locally and reported to the centre automatically. However, novel concepts and constructs require the intervention of the central knowledge engineering team as does the overall reconciliation and integration of the work of the centres. This combination of local autonomy with central support and integration produces the double cycle is shown in Figure 3 which is characteristic of development using GALEN. This pattern is effective at maximising local autonomy and minimising the requirement for central

coordination. Overall, in an established area of development, central services require about 10% of the total effort.

DESIGN ISSUES

The layered architecture has allowed *Open*GALEN to evolved a principled and systematic approach to designing ontologies for clinical applications which addresses a wide range of issues:

- Issues in design of the ontology itself and the transforms between the intermediate representation and ontology. The goal is that domain experts be largely unaware of these choices because they are handled by the transformation between the intermediate representations and the ontology. In most cases, technical changes to the ontology should not require any recourse to the domain experts.

- Issues in dealing with the idiosyncrasies of the domain which can only be implemented as guidelines to the domain experts or constraints within the tools.

Key issues from both sets are discussed below along with how their interrelation with the underlying ontology and the intermediate representation.

Issues in the design of the ontology and transforms

Untangling taxonomies

GALEN's source material typically consists of seriously tangled hierarchies, typically derived from 'broader than'/ 'narrower than' constructs in traditional library science and thesauri rather than the formal inferential meaning of subsumption in description logics. The hierarchies typically mix the notions of kinds, parts, function, use etc. The patterns are familiar to users, make for easy access to terms, but make formal inference all but impossible. For example, heart diseases are found in thirteen of the eighteen chapters of the International Classification of Diseases.

GALEN's approach is to separate out each 'axis' into a separate taxonomy of elementary concepts, and then recombine as expressions in the description logic. Where two axes are highly correlated, this can involve introducing much seemingly redundant information – *e.g.* separating the 'action' and 'use' of drugs may lead to recording separately an action of 'bronchodilation' and a indicated for 'bronchodilation'. However, in other cases the use and action may be quite different – *e.g.* an action of 'vasodilation' and a use of 'management of hypertension'.

Operationally, *Open*GALEN maintains the principle modularity by specifying that *elementary* concepts should break down into disjoint taxonomies, *i.e.* each elementary concept should have only one elementary parent and be disjoint from all its 'sibling' elementary concepts. The taxonomies of elementary first class concepts are open – *i.e.* at each level of the hierarchy siblings are disjoint but do not exhaust the parent concept. This reflects the reality that lists of diseases, abnormalities, and even anatomy can almost never be fully exhaustive, especially when the

possibility of congenital abnormalities are taken into account. By contrast taxonomies of modifying concepts such as 'severity' may exhaustive and therefore closed. All multiple classification and overlapping of concepts are the result of definitions and descriptions. This may involve creating artefactual concepts known as 'roles', *e.g.* "doctor" is defined as a "person who plays a 'doctor role'" and a hormone as a "substance which plays a 'hormonal role'". (This use of word "role" is not to be confused with the use of "Role" for semantic relation in description logic parlance.) This allows clean disjoint taxonomies for the notions of 'organism', 'person', etc. and for 'social role', 'clinical role', 'doctor role', etc.

The structures which result from untangling taxonomies and recombining them through logical definitions are consistent but contain detail which is irrelevant to users. Some of this detail can be hidden by definitions in the ontology itself, but an important function of the intermediate representation is to hide the rest.

Locations, parts, wholes & related spatial notions

Much of the power of *Open*GALEN's ontology stems from its distinction between different sorts of part-whole and other spatial relations. Although adapted from Winston's structure[13, 28], it differs from it and distinguishes:

Location – Lesions and abnormalities are 'located' in things rather than part of them or contained in them. (If physical containment is implied, as in foreign bodies it is specified additionally and separately.)

Parts – in four main flavours

> *Division* – Roughly self similar parts having the same layers, e.g. hand and arm

> *Layers* – horizontal layers such as the skin which extend across divisions

> *Structural Components* – discrete parts which normally reside in only one division

> *Functional Components* – parts of a functional unit which may or may not be contained in or contiguous with the whole, e.g. the various glands which make up the endocrine system.

Containment – physical containment of one structure by another where there is quite different function and origin, e.g. bone marrow in bones

Connections – which may or may not be considered part of the things connected.

There are also distinctions drawn between two-dimensional and three-dimensional parts analogous to those in the Digital Anatomist project [18].

A second issue is that the part-whole structure requires the use of propagation ('specialised_by') axioms similar to Cyc's TRANSFERS-THRO [10] to cope with the paradigm that "diseases/procedures of a parts are diseases/procedures of wholes". The general schema required is equivalent to

$$R_1 \circ R_2 \rightarrow R_1$$

for at least a restricted set of roles R_1 and R_2[15]. Classification algorithms for description logics supporting

such schemas remain an outstanding problem [2] [Franconi, personal communication]. GRAIL implements a partial solution for restricted cases. Shulz and Hahn have suggested an alternative construct that covers most cases [5].

Establishing the schemas for anatomical structure and the propagation axioms requires careful consideration by the knowledge engineers. Once the anatomical structure is established, the use of the different relations in descriptions of surgical procedures, diseases, etc. can be determined automatically based on the concept types by the transforms between the intermediate representations and the underlying ontology.

Defaults and 'Extrinsics'

A major function of Frame systems is to deal with default knowledge – *i.e.* information which is true in general but subject to exceptions. Formal description logics do not support default reasoning. However, they can provide a framework for separate default reasoners.

In a static system it is always possible to 'compile out' default knowledge out by re-representing each item at all highest levels below which there are no exceptions. However, this strategy is inappropriate for knowledge acquisition and for use in many dynamic systems because it may not provide default values for new information when added, which is a key function in many such applications. For example, drug-drug interactions are best specified at the level of drug classes with the exceptions enumerated explicitly, so that when a new drug is added, it acquires the default 'safe' set of interactions unless they are explicitly overridden.

GRAIL provides a special mechanism for attaching 'extrinsic' statements to concepts which do not affect their classification but which can be manipulated by a special set of operations based around the notion of retrieving the set of 'most specific' extrinsic statments of a given type. This mechanism is also used for handling complex mappings to external classifications and terminologies and for links to natural language applications.

The distinction between extrinsic (default) and intrinsic (definitional and descriptive) information is not at all intuitive to domain experts. The decision as to which constructs should be 'extrinsic' is made by the knowledge engineers and implemented in the transforms between the intermediate representation and description logic so as to be transparent to domain experts.

Reification of relations and 'wrapping'

As stated in Section 2.1, the criteria for correctness in *OpenGALEN* is consistent classification and re-use rather than any notion of 'naturalness'. To achieve consistent re-usable representation within the underlying ontology requires a number of complex constructs which are concealed from users by the intermediate representation

For many purposes, all diseases are 'wrapped' and represented as collections of one or more disease

concepts in order to cope with common constructs as used in medical records and existing coding systems such as 'A with B', 'A without B', etc.

- To cope with the fact that GRAIL does not handle negation explicitly and to make the distinction between absence of information and negative information unambiguous, disease and procedures are usually expressed with a second layer of wrapping as 'presence/absence of condition, 'Performance/ Nonperformance of procedure', etc.

- All modifying relations are reified as 'features' which may be chained in order to allow consistent re-usable patterns. For example "elevated temperature" is represented in the ontology itself analogously to *patient-hasFeature-temperature-hasFeauture-elevation-hasState-elevated* rather than *patient-hasTemperature-elevated.*

All of these transformations are hidden from the domain expert, so that a simple notion which appears to the user as 'Diabetes hasState severe' in the intermediate representation is transformed into an internal representation in the ontology analogous to *ClinicalSituation-involves-(Presence-isExistentialStateOf-(Diabetes-hasFeature-(Severity-hasValue-severe))).*

Dealing with the idiosyncrasies of the domain knowledge

Semantic normalisation

It is easy to agree that all surgical procedure are constituted by an 'act' on some 'thing' which either is, or is located in, an anatomical structure. It is less easy to agree on what constitutes an 'act' when there is a hierarchy of motivations: for example, "inserting a pins to fixate a fractured bone" or "destruction of a polyp by cautery" and "removal of a polyp (by excision)". Furthermore, important classifications hang notions of motivation such as "palliative surgery" and "corrective surgery". In addition, some systems wish to be able to record operations just as 'correction of X' without describing the exact 'act' while others wish to record 'insertion of pins in fractured bone' without recording that the purpose is fixation.

To address this problem, one of the project members proposed a classification into four levels: L4 Clinical Goal (palliation, Cure); L3 Physiologic goal: (correction, destruction, ...); L2 primary surgical method (excision, insertion, lysis,...); and L1: low level surgical act (cutting, cautery, ...) [19]. It was tempting to believe that a list of concepts in each category could be agreed, so that resolution could be done automatically. However, intuitions and requirements clashed sufficiently to make this difficult. For example, 'Cautery' can sometimes be a low level act or sometimes a primary method. These ambiguities are dealt with in the formal ontology by having concepts for "simple cautery" and "removal by cauterisation".

Concealing such distinctions from the domain experts completely sometimes adds more confusion than it avoids. Therefore, semantic normalisation is dealt with by a combination of guidelines for how things should be done, transforms which recognise anomalies, and quality assurance procedures to catch remaining inconsistencies.

Dealing with implied and normative knowledge

A key problem in dealing with pre-existing terminologies is that much of the information is implied rather stated. Hence the requirement that each term or 'rubric' be paraphrased before being represented. Many of the guidelines concern paraphrasing of different sorts of rubrics.

A key part of this process is expanding expressions such as "insertion of pins in the Femur" to "Fixation of Femur by means of insertion of pins". That the intended meaning includes fixation can only be inferred from context and general medical knowledge – if the insertion were for any other purpose it would be stated in the rubric since fixation constitutes the overwhelming majority of reasons for inserting pins into femurs - so much so that it is not stated in the rubric. It is part of the meaning in context but not of the literal meaning. Similarly many disease classifications depend on normative anatomy which is not invariably true. For example, the thyroid gland is almost always located in the neck but may be ectopically located in the chest.

Idiomatic meaning vs logical definition

A closely related problem occurs when describing important abstractions such as 'Heart Valve' or 'Endocrine Surgery'. These concepts might naturally be defined as "Valve in the heart" or "Surgery on an endocrine organ" respectively. Both produce results which surprise clinicians. 'Heart Valve' conventionally means one of the four main valves at the entrance and exit of the ventricles rather than any of the other valvular structures, many of which are normally active only prior to birth. Similarly, 'Endocrine surgery' typically refers to a particular set of operations on endocrine organs excluding the reproductive tract, even though all would agree that the gonads have endocrine function.

Achieving a familiar structure: Tagging vs Mapping to original sources

The untangling process by itself provides descriptions of leaf concepts, but may not provide the higher level abstractions users expect. One important requirement is often to provide additional tagging to reproduce the familiar hierarchies, so that users can find concepts where they expect them. This is usually done by adding the mapping as part of the description of leaf concepts, and then creating concepts such as 'drugs in chapter three' as abstractions.

Note that this tagging is used only to mark high level constructs in the source classifications. Detailed mapping of leaf nodes in source classifications almost always requires taking into account special rules of usage unique to that classification and so requires further inference mechanisms beyond the scope of this paper.

Pathology and abnormality

Being 'normal' or 'abnormal', 'pathological' or 'physiological' are key notions in medicine. However, what is meant by such terms is a thorny issue in general medical usage let alone formal ontologies. *OpenGALEN* has established a consistent approach: "Abnormal" indicates "clinically noteworthy"; "pathological indicates "in need of clinical management (possibly by doing nothing)". This approach is reflected at all three levels, the ontology, intermediate representations, and guidelines, but the complex interrelations and inferences are confined to the underlying ontology and hidden from domain experts.

AN OUTLINE EXAMPLE

The following is an abbreviated example of the process as applied to analysis of surgical procedure terminologies from original rubric through paraphrase, intermediate representation, transformation to generated natural language.

RUBRIC: *Insertion of pins in neck of femur*

PARAPHRASE: *Fixation of femur by insertion of pins in neck of femur*

INTERMEDIATE REPRESENTATION:

MAIN fixation
  ACTS-ON femur
  BY-MEANS-OF insertion
    ACTS-ON pins
      INTO neck
        IS-PART-OF FEMUR

GRAIL/GALEN Ontology

Performance which isOf
('SurgicalFixation' which
  <actsOn Femur
  hasSubprocedure (Performance which isOf
    'SurgicalInsertion' which
      <actsOn Pins
      hasLocation (AnatomjicalNeck which
                     isLinearDivisionOf Femur)>)>)

GENERATED LANGUAGE:

"Fixation of femur by means of insertion of pins in neck of femur"

(In the GRAIL representation, concepts in single quotes are further defined elsewhere and *which* is a keyword introducing a series of attribute-value pairs bracketed by <...>. See [15] for full details of notation.)

Note that the transform from intermediate representation to the GALEN Ontology has supplied the context specific mapping of *INTO* to *hasLocation* and *IS-PART-OF* to *IsLinearDivisionOf* based on the classification of *Pins, AnatomicalNeck,* and *Femur*. Given different categories of object and value, *INTO* might have been transformed as *contains* and *IS-PART-OF* might have been transformed as *isComponentOf, isLayerOf, contains,* etc. Note also that

the 'wrapping' *Performance* has been provided which allows for combinations which involve *NonPerformance*. Here the generated language is close to the original paraphrase, but more complex cases lead to less felicitous language.

RESULTS AND DISCUSSIONS

*Open*GALEN has been used in two major areas:

- Developing and maintaining surgical procedure classification in several European countries including being the primary development vehicle for the new classification in France reconciling previously separate systems used in public and private sectors.
- Developing a drug ontology for use in prescribing support in the UK as part of the PRODIGY project [9, 14].

The methodology for distributed loosely coupled development has been used primarily in the surgical procedure development during the EU funded GALEN-IN-USE project where it allowed nine centres in seven countries to co-operate on various aspects of developing and integrating surgical procedure classifications. The introduction of the intermediate representation reduced the training time required for domain experts to roughly three days plus telephone and email support, sometimes supplemented by a one or two further days of advanced training. This contrasts the several months training required for a knowledge engineer to be able to use the underlying ontology. Just as important, it dramatically reduced the time and effort required to reach consensus. Domain experts did not have to deal with what they regarded as arcane distinctions, and controversial decisions could be deferred until sufficient data was gathered to make choices based on evidence rather than dogma. The fear of wasting effort was reduced, because it was possible to preserve the intermediate representations and change only the transforms to the underlying description logic.

Natural language generation has proved unexpectedly to be essential to acceptance by users – no matter how intuitive the intermediate representation appears to designers, simple noun phrases are more compact and accessible to domain experts, especially for quality assurance.

At this level, the method appears cost effective compared with the alternative manual development of classifications. Replication is required, but a preliminary study by the Dutch collaborators indicated that the cost of using *Open*GALEN techniques was on the order of 25% that of using conventional techniques even including the one-time-only cost of take on, primarily because the techniques reduced the number of costly meetings of expert committees and led to more rapid consensus [P Zanstra, Personal communication].

The importance of the approach to 'untangling taxonomies' can perhaps best be illustrated by recent experience with the major medical standards body Health Level Seven (HL7). The seemingly simple problem of classifying the forms and routes by which a medication can be given –

"oral tablets", "nasal sprays", "ointments to be rubbed on the skin" had caused serious difficulties. There are at least five different axes involved. Between the various providers of drug information there are over 800 concepts – a fraction of what are involved in other *Open*GALEN knowledge bases but nonetheless, a significant number. Developing a classification manually had proved a daunting task and was still incomplete after over a year; developing the classification using *Open*GALEN's formal methods was completed with a few weeks effort with contributions from five sources, none of them with previous involvement with *Open*GALEN or any formal training [29].

The separation into of the development into design and population phase, and the separation of the design issues between those involving the underlying ontology and those involving the domain itself have improved the ability to reach consensus and vastly reduced the number of arguments – a major cost in ontology development in our experience. This approach contrasts sharply with the more centralised approach taken in the Convergent Terms Project and SNOMED-Reference-Terminology projects [4, 22].

The layered architecture seems to us almost inevitable for the design of large re-usable ontologies. The predecessor application, PEN&PAD [6, 12] based the implementation directly on the ontology without an intervening layer. As a result, the developers frequently succumbed to the temptation to change the ontology to fit the application, sacrificing re-use to expediency. GALEN's intermediate representations provide a sounder alternative.

We believe the need for such intermediate representations will become more, rather than less, critical as more powerful description logics such as FaCT and ShiQ [8] come into use. While it is tempting to believe that OIL [1] will provide a suitable vehicle for direct development, our preliminary experience suggests that it is best treated as a language of similar level to GRAIL – a better vehicle for knowledge engineers but best hidden from domain experts who will still require environments oriented to their specific needs and packaged together with specialist tools for reasoning, access to information, calculation, and other services, presented at a level which corresponds to the issues which concern them, with contact with the implementation in formal logic only where necessary.

At the same time the use of intermediate representations presents an important route for adapting re-usable ontologies to specific applications. For architectures such as PROTÉGÉ, in which applications are developed based on an ontology, the hope is that 'meta authoring' of suitable intermediate representations and views onto a more general re-usable ontology might replace the repeated development of bespoke ontologies. This is already occurring to some degree within the PRODIGY project [14, 20].

145

REFERENCES

1. The OIL Home Page. www.ontoknowledge.org/oil/
2. Artale A, Franconi E, Guarino N. Open problems for part-whole relations. In: International Workshop on Descripition Logics; 1996; Boston, MA; 1996. http://www.dl.kr.org/dl96/.
3. Boose J, Gaines B, (eds). Knowledge Acquisition Tools for Expert Systems. Academic Press; 1988.
4. Campbell K. Scalable methodologies for distributed development of logic-based convergent medical terminology. Meth Inf Med 1998;37(426-439).
5. Hahn U, Schulz S, Romacker M. Partonomic reasoning as taxonomic reasoning in medicine. In: Proc. of the 16th National Conf. on AI & 11th Innovative Applications of AI (AAAI-99/IAAI-99); 1999; Orlando FL: AAAI Press/MIT Press; 1999. p. 271-276.
6. Horan B, Rector A, Sneath E, Goble C, Howkins T, Kay S, et al. Supporting a Humanly Impossible Task: The Clinical Human-Computer Environment,. In: Diaper D, (eds). Interact 90; 1990: Elsevier Science Publishers, B.V.North-Holland; 1990. p. 247-252.
7. Horrocks I. Using an expressive description logic: FaCT or Fiction. In: Cohn AG, Schubert LK, Shapiro SC, (eds). Principles of Knowledge Representation and Reasoning: Proceedings of the Sixth Int Conf on Knowledge Representation (KR 98); Morgan Kaufmann; 1998. p. 634-647.
8. Horrocks I, Staller U, Tobies S. Practical reasoning for very expressive description logics. Journal of the Interest Group in Pure and Applied Logics (IGPL) 2000;8(3):293-323.
9. Johnson PD, Tu S, Booth N, Sugden B, Purves I. Using scenarios in chronic disease management guidelines for primary care. J Am Med Assoc 2000 (Symposium Special Issue):389-393.
10. Lenat DB, Guha RV. Building Large Knowledge-Based Systems: Representation and inference in the Cyc Project. Reading, MA: Addison-Wesley; 1989.
11. Musen M. Modern architectures for intelligent systems: reusable ontologies and problem-solving methods. J Am Med Inf Assoc 1998 (Symposium supplement):46-54.
12. Nowlan WA. Clinical workstation: Identifying clinical requirements and understanding clinical information. Intl J Bio-Med Comput 1994;34:85-94.
13. Odell JJ. Six different kinds of composition. Journal of Object Oriented Programming 1994;5(8):10-15.
14. Johnson PD, Tu S, Booth N, Sugden B, Purves I. Using scenarios in chronic disease management guidelines for primary care. J Am Med Assoc 2000 (Symposium Special Issue):389-393.
15. Rector A, Bechhofer S, Goble C, Horrocks I, Nowlan W, Solomon W. The GRAIL concept modelling language for medical terminology. AI in Medicine 1997;9:139-171.
16. Rector AL. Clinical Terminology: Why is it so hard? Meth Inf Med 1999;38:239-252.
17. Rector AL, Zanstra PE, Solomon WD, Rogers JE, Baud R, Ceusters W, et al. Reconciling Users' Needs and Formal Requirements: Issues in developing a Re-Usable Ontology for Medicine. IEEE Trans Inf Tech in BioMedicine 1999;2(4):229-242.
18. Rosse C, Shapiro IG, Brinkley JF. The Digital Anatomist foundational model: Principles for defining and structuring its concept domain. J Am Med Inf Assoc 1998(Fall Symposium Special issue):820-824.
19. Rossi Mori A, Gangemi A, Steve G, Consorti F, Galeazzi E. An ontological analysis of surgical deeds. In: AI in Medicine Europe (AIME-97); 1997; Springer Verlag; 1997. p. 361-372.
20. Solomon DS, Wroe C, Rogers JE, Rector A. A reference terminology for drugs. J Am Med Inf Assoc 1999 (Fall Symposium Special Issue):152-155.
21. Solomon W, Roberts A, Rogers J, Wroe C, Rector A. Having our cake and eating it too: How the GALEN Intermediate Representation reconciles internal complexity with users' requirements for appropriateness and simplicity. J Am Med Inf Assoc 2000 (Fall Symposium Special Issue):819-823.
22. Spackman KA, Campbell KE, Côté RA. SNOMED-RT: A reference Terminology for Health Care. J Am Med Inf Assoc (J Am Med Assoc) 1997 (Symposium special issue):640-644.
23. Staab S, Maedche A. Ontology engineering beyond the modeling of concepts and relations. In: Benjamins RV, A. Gomez-Perez, N. Guarino, Uschold M, (eds). ECAI 2000. 14th European Conference on Artificial Intelligence; Workshop on Applications of Ontologies and Problem-Solving Methods; 2000; 2000.
24. Tu S, Eriksson H, Gennari J, Shahar Y, Musen M. Ontology-based configuration of problem-solving methods and generation of knowledge-acquisition tools - application of PROTEGE-II to protocol-based decision-support. AI in Medicine 1995;7:257-289.
25. Tu SW, Musen MA. A flexible approach to guideline modelling. In: AMIA Fall Symposium; 1999; Washington DC: Hanley and Belfus; 1999. p. 420-424.
26. Uschold M, Gruninger M. Ontologies: principles, methods and applications. Knowledge Engineering Review 1996;11(2).
27. Wielinga B, Van de Velde W, Schreiber G, Akkermans H. The KADS Knowledge Modelling Approach. In: The Japanese Knowledge Acquisition Workshop (JKAW '92); 1992; 1992.
28. Winston M, Chaffin R, Hermann D. A taxonomy of part-whole relations. Cog Sci 1987;11:417-444.
29. Wroe C, Cimino J. Using openGALEN techniques to develop the HL7 drug formulation vocabulary. In: J Am Med Inf Assoc (Fall Symposium special issue); 2001; (Submitted for publication).
30. Wroe C, Solomon W, Rector A, Rogers J. Inheritance of drug information. J Am Med Inf Assoc 2000 (Annual Symposium Special Issue):1158.

Inferring the Environment in a Text-to-Scene Conversion System

Richard Sproat

Human/Computer Interaction Research
AT&T Labs — Research
180 Park Avenue
Florham Park, NJ 07932, USA
rws@research.att.com

Abstract

There has been a great deal of work over the past decade on inferring semantic information from text corpora. This paper is another instance of this kind of work, but is also slightly different in that we are interested not in extracting semantic information per se, but rather real-world knowledge. In particular, given a description of a particular action — e.g. *John was eating breakfast* — we want to know where John is likely to be, what time of day it is, and so forth. Humans on hearing this sentence would form a mental image that makes a lot of inferences about the *environment* in which this action occurs: they would probably imagine someone in their kitchen in the morning, perhaps in their dining room, seated at a table, eating a meal.

We propose a method that makes use of Dunning's likelihood ratios to extract from text corpora strong associations between particular actions and locations or times when those actions occur. We also present an evaluation of the method. The context of this work is a text-to-scene conversion system called WordsEye, where in order to depict an action such as *John was eating breakfast*, it is desirable to make reasonable inferences about where and when that action is taking place so that the resulting picture is a reasonable match to one's mental image of the action.

Keywords

Common sense knowledge; statistical natural language processing; text-to-scene conversion.

INTRODUCTION

There has been a great deal of work over the past decade on inferring semantic information from text corpora; see [9, 8, 17, 18, 2, 19, 12, 13] for some examples. This paper is another instance of this kind of work, but is also slightly different in that we are interested not in extracting seman-

tic information per se, but rather real-world knowledge. In particular, given a description of a particular action — e.g. *John was eating breakfast* — we want to know where John is likely to be, what time of day it is, and so forth. Humans on hearing this sentence would probably form a mental image of someone in their kitchen, perhaps in their dining room, seated at a table, eating a meal in the morning. But note that the sentence omits a lot of this information, and says nothing explicit about the location of the action, or the time of day. Nonetheless, people would usually make these inferences about the *environment* in which the particular action occurs.

The context of this work is a text-to-scene conversion system called WordsEye, which we describe in the next section. Subsequent sections describe the method for extracting information about the environment from text corpora, and an evaluation of the method.

THE WORDSEYE SYSTEM

WordsEye [5] is a system for converting from English text into three-dimensional graphical scenes that represent that text. WordsEye works by performing syntactic and semantic analysis on the input text, producing a description of the arrangement of objects in a scene. An image is then generated from this scene description. At the core of WordsEye is the notion of a "pose", which can be loosely defined as a figure (e.g. a human figure) in a configuration suggestive of a particular action. For example a human figure holding an object in its hand in a throwing position would be a pose that suggests actions such as *throw* or *toss*. Substituting for the figure or the object will allow one to depict different statements, such as *John threw the egg* or *Mary tossed the small toy car*.

The natural language component in the current incarnation of WordsEye is built in part on several already existing components, including Church's [3] part of speech tagger, Collins' head-driven stochastic parser [4] and the WordNet semantic hierarchy [7]. The parsed sentence is first converted into a dependency representation. Then lexical semantic rules are applied to this dependency representation to derive the com-

Figure 1: *Mary uses the crossbow. She rides the horse by the store. The store is under the large willow. The small allosaurus is in front of the horse. The dinosaur faces Mary. A gigantic teacup is in front of the store. The gigantic mushroom is in the teacup. The castle is to the right of the store.*

Figure 2: *John was eating breakfast.* **The light in the sky coming through the window is morning light (though that may be hard to see in a black and white version).**

ponents of the scene description. For instance the verb *throw* invokes a semantic rule that constructs a scene component representing an action (ultimately mapped to a pose) where the lefthand noun phrase dependent represents an actor, the righthand noun phrase dependent a patient, and some dependent prepositional phrases the path of the patient.[1] WordNet is used as part of the implementation of noun semantics, both to derive appropriate sets of objects (e.g. *the vehicle* will get all vehicle objects by inheritance from WordNet subclasses such as *car, airplane,* etc.); and in subsequent reference resolution (so that one can refer to, e.g., an *allosaurus* and subsequently use *dinosaur* to refer to the previously evoked allosaurus).

The depiction module of WordsEye interprets the scene description to produce a set of low-level depictors representing poses, spatial relations, color attributes etc. Transduction rules are applied to resolve conflicts and add implicit constraints. The resulting depictors are then used (while maintaining constraints) to manipulate the 3D objects that constitute the final, renderable 3D scene. An example of a fairly complex scene constructed with WordsEye is shown in Figure 1.

One problem that arises in such a system is how to derive the large amount of knowledge that is needed in order to give reasonable depictions. Suppose I say: *John was driving to the store.* In understanding this sentence and visualizing what it means, a human would probably assume that John was in the driver's seat of a car, on a road, possibly passing buildings, and so forth. Many of these inferences are defeasible: I can easily cancel the inference about the road, for example, by saying *John was driving to the store across the muddy field.* But without such explicit cancellation the inferences seem

fairly robust. To take another example, if we say *John ate his dinner at 7*, we assume that it is 7 in the evening (possibly near twilight), that he is in a room such as his dining room or his kitchen (or possibly in a restaurant), and that he is seated at a table. Or if *John was eating breakfast*, we would usually assume that it is morning, and that John is in his kitchen or dining room. See Figure 2. Finally, if *John is shoveling snow*, it is probably winter.

Some of this knowledge is represented in WordsEye as part of the word's meaning. For example, the depiction phase of WordsEye knows that for *drive*, the driver should be using some sort of vehicle, and will select an appropriate vehicle and place the driver in the driver's seat. But other common sense knowledge is more tenuously linked: if John is washing his face, he is probably in a bathroom, but need not be: there is nothing in the meaning of *wash face*, that implies a bathroom.

An important problem is how to acquire this kind of knowledge. One approach would of course be to do it by hand, possibly making use of already hand-built ontologies such as Cyc [14], or Mikrokosmos [15]. In this paper we explore the alternative of deriving this kind of information from text corpora.[2]

The question posed by this paper can therefore be stated as follows: if John is eating dinner, can we infer from text corpora where he is and what time of day it is? If John is raking leaves, can we infer from text corpora what season it is and where he is likely to be?

METHOD

The first step involves computing a set of concordance lines for terms that can denote elements of the set of interest. For

---

[1] We have just started investigating the use of FrameNet [10] for verbal semantics.

[2] We do not mean to imply, however, that hand-built ontologies such as Cyc and statistical methods such as the one proposed here, are at odds with one another. Rather, the two approaches complement one another, as we will suggest in the final section.

example, if one is interested in activities that can take place in various rooms of a house, one would compute concordance lines for terms like *kitchen, living room, dining room, hallway, laundry room* and so forth: so, for the key word *kitchen*, one would simply find all places in a corpus that have the word *kitchen*, and for each of these, output a line containing the key word surrounded by words in a predetermined window of that corpus location.

We used a corpus of 415 million words of English text, consisting of about nine years of the *Associated Press* newswire, the Bible, the Brown corpus [11], *Grolier's Encyclopedia*, about 70 texts of various kinds published by Harper and Row, about 2.7 million words of psychiatry texts, a corpus of short movie synopses, and 62 million words of the *Wall Street Journal*. The texts in this corpus had already been automatically tagged with a part of speech tagger [3] and so the concordance lines also contain part of speech information for each word.[3]

Sample concordance lines for various rooms from the 1996 *Associated Press* newswire are given in Figure 3. (Here we omit the part of speech information for readability.) As expected, the data are noisy: for example in the third line, *Kitchen* is a family name, not a room in the house. Note that in the actual implementation, a window of 40 words on each side of the target is used, wider than what is shown here.

Once the concordance lines are collected, and after sorting to remove duplicates (newswire text especially contains a lot of repeated stories), we extract *verb-object* (e.g. *wash face*) and *verb-preposition-object* (e.g. *get into bed*) tuples. Unlike verbs alone, verb-argument tuples of this kind are usually pretty good indicators of a particular action. Thus, whereas *wash* is consistent with many activities (e.g. washing oneself, washing one's car, washing clothes), a particular verb-object construction such as *wash clothes* is usually indicative of a particular activity. In the present system, the tuples are extracted using a simple matching algorithm that looks for verbal part-of-speech tags and then searches for what looks like the end of the following noun phrase, with a possible intervening preposition.[4] Verb-object and verb-preposition-object tuples extracted from the concordance lines in Figure 3 are shown in Figure 4.

Once again the data are noisy, and include misanalyses (*didn't window*) and complex nominals that are not instances of verb-object constructions (*swimming pool*). Apart from misanalyses of the tuples, one also finds many instances where the target term does not have the intended denotation. For example, if

a concordance line matching *kitchen* will not always have to do with kitchens. As we saw above, *Kitchen* may be a family name, but a more common instance is that it is part of a complex nominal, such as *kitchen knife*. In such instances the text is not generally talking about kitchens, but rather about kitchen knives, which can be used in rooms besides kitchens. To remove such cases we filter the concordance lines to remove the most frequent collocations (for example the 200 most frequent ones).

The next step is to compute the association between each of the tuples and the target term, such as the name of a room. For this stage we use likelihood ratios [6, 16], which compute the relative likelihood of two hypotheses concerning two events e_1 and e_2:

- Hypothesis 1: $p(e_2|e_1) = p = p(e_2|\neg e_1)$

- Hypothesis 2: $p(e_2|e_1) = p_1 \neq p_2 = p(e_2|\neg e_1)$

Hypothesis 1 simply says that the probability of e_2 occurring given e_1 is indistinguishable from the probability of e_2 occurring given something other than e_2: i.e., the e_2 is not particularly expected (or unexpected) given e_1. Hypothesis 2 says, in contrast, that there is a difference in expectation, and that e_2 is dependent on e_1.

We can estimate the probabilities p, p_1 and p_2 by the maximum likelihood estimate as follows, where c_1, c_2 and c_{12} are, respectively the frequency of e_1, of e_2, and of e_1 and e_2 cooccurring; and N is the size of the corpus:

$$p = \frac{c_2}{N}, p_1 = \frac{c_{12}}{c_1},$$
$$p_2 = \frac{c_2 - c_{12}}{N - c_1}$$

If we assume a binomial distribution

$$b(k; n, x) = \left(\begin{array}{c} n \\ k \end{array} \right) x^k (1-x)^{(n-k)}$$

then the likelihoods of the two hypotheses, given the observed counts e_1, e_2 and e_{12}, can be computed as:

$$L(H_1) = b(c_{12}; c_1, p)b(c_2 - c_{12}; N - c_1, p)$$
$$L(H_2) = b(c_{12}; c_1, p_1)b(c_2 - c_{12}; N - c_1, p_2)$$

The *log* likelihood ratio for the two hypotheses then reduces as follows:

$$
\begin{aligned}
log\lambda &= log\frac{L(H_1)}{L(H_2)} \\
&= logL(c_{12}, c_1, p) + logL(c_2 - c_{12}, N - c_1, p) \\
&\quad - logL(c_{12}, c_1, p_1) - logL(c_2 - c_{12}, N - c_1, p_2)
\end{aligned}
$$

where:

---

[3]The concordance itself is computed using a corpus encoding representation and a set of corpus tools developed at AT&T, but this could just as easily have been done with any of a number of other concordancing software packages.

[4]This is currently done with an ad hoc script, though we are investigating using Cass [1], a robust chunk parser, in the future. Note that while the Collins parser is used in the runtime version of WordsEye, it is far too slow to use to parse large amounts of text.

anything else, her books are about memories:	*kitchen*	memories, barnyard memories, family memories
both videotapes and photos of her in bathrooms and	*bedroom*	and asks for an unspecified amount of
will happen to Mr Tarkanian," said Jack	*Kitchen*	, one of the NCAA's lawyers
grounded for telling his parents he didn't open his	*bedroom*	window. He confessed in
gone, replaced by a big house with five	*bathroom*	and an indoor swimming pool.
The second child was born in a	*bedroom*	of their home near Scottsdale after Corvin
beds in semiprivate rooms at one end of a	*hallway*	separated from the "older adult"
and the couple's 15-month-old son use a downstairs	*bedroom*	that lies in Granite City along with
of the halls, equipped with microwaves and other	*kitchen*	appliances not allowed in individual rooms.

Figure 3: Sample concordance lines from the 1996 Associated Press.

asks		amount	**bedroom**
happen	to	Tarkanian	**kitchen**
grounded		parents	**bedroom**
telling		parents	**bedroom**
didn't		window	**bedroom**
replaced	by	house	**bathroom**
swimming		pool	**bathroom**
born	in	home	**bedroom**
use	in	City	**bedroom**
lies	in	City	**bedroom**
equipped	with	microwaves	**kitchen**
allowed	in	rooms	**kitchen**

Figure 4: Some verb-argument combinations extracted from Figure 3.

$$L(k, n, x) = x^k (1 - x)^{n-k}$$

Following [6, 16] we make use of the fact that $-2log\lambda$ is asymptotically χ^2 distributed, and compute $-2log\lambda$, rather than just $log\lambda$. In what follows we assume p value of 0.05, which has a critical χ^2 value of 3.84 for one degree of freedom. Thus any $-2log\lambda$ value of 3.84 or above will be considered evidence of association.

After the likelihood ratios for each tuple-term pair are computed, we then sort the tuple-term pairs, and filter to remove those that are below the significance threshold; in the process of doing this, we also lemmatize the verb forms, or in other words replace inflected verbs (e.g. *eats*) by their base forms (e.g. *eat*). A sample of the highest ranking tuple-term associations is given in Figure 5. Again, there is still noise, including a misanalyzed complex nominal (*dine room* from *dining room*), misparsed examples (*find in Simpson* from *find in Simpson's X*) and so forth.

The final stage is to filter the list for tuples that designate reasonable depictable actions. We do this by extracting activities from the set of sentences input to the WordsEye system; at the time of writing this consisted of 20K words (about 3,400 sentences). We then use these activities to filter the raw likelihood-ratio-ordered list. An example of a filtered list is shown in Figure 6. A similar example for times of day is shown in Figure 7.

EVALUATION

The system has been evaluated by human subjects on its predictions for the **rooms**, **seasons** and **times** of day in which particular actions or situations occur. Clearly these are not the only things that one would like to infer about a scene, but they are three fairly obvious ones, and serve to give us a metric for evaluating the method as a whole.

The test used sentences constructed based on verb-object or verb-preposition-object tuples from the final filtered lists for rooms, seasons and times, as described in the last section. This meant that the system would be able to predict an answer for at least one of these categories for each of the sentences, but at the same time there was no guarantee that the prediction would be correct. This resulted in 106 sentences from which 90 were randomly selected. Of these 90, 30 were submitted to the system to label the choices for the three variables listed above; 30 were given to a human for labeling; and 30 — the "baseline" system — had the answers labeled randomly.

The three sets of judgments were randomized, and presented via a web-based interface to subjects, who were informed that they were judging the output of an automatic system; subjects were not informed that some sentences had been labeled randomly, or that some had been labeled by a human. (For those who are interested, the exact instructions given to subjects are shown in the Appendix.)

150

306.585215	143	424	10227	dine room	dining room
196.753628	63	65	32243	find in Simpson	bedroom
150.457758	29	31	10227	serve in room	dining room
137.680378	35	51	10227	designate areas	dining room
117.189848	23	25	10227	eat in room	dining room
109.719646	25	29	12457	wash clothes	laundry room
107.275571	24	30	10227	cook on premises	dining room
100.616896	19	19	12457	sell in America	laundry room
96.602198	205	575	32243	live room	bedroom
79.429912	15	15	12457	cook appliances	laundry room
76.659647	43	68	28224	kill people	garage
61.528933	49	64	51214	sit at table	kitchen
61.103395	30	47	24842	give birth	bathroom
61.067298	18	18	32243	see socks	bedroom
58.542468	16	16	28224	rent van	garage
54.146381	18	21	24842	wash hands	bathroom
51.280771	21	54	10227	dine rooms	dining room
51.111709	26	28	51214	prepare meals	kitchen
49.807875	10	10	14575	push down gantlet	hallway
49.807875	10	10	14575	form gantlet	hallway
47.564595	13	13	28224	carry bomb	garage

Figure 5: Most likely actions associated with particular rooms. Columns represent, from left to right: the likelihood ratio; the frequency of the tuple/target-term pair; the frequency of the tuple; the frequency of the target term; the tuple; and the target term.

The full set of choices for each of the categories were as follows:

Room: bedroom, kitchen, laundry room, living room, bathroom, hallway, garage, ANY, NONE OF ABOVE

Time of day: morning, midday, afternoon, evening, night, ANY

Season: winter, spring, summer, autumn, ANY

"ANY" indicates that any of the choices would be acceptable. "NONE OF ABOVE", in the case of rooms, indicates that the action could not occur in any of the rooms; typically this would be because the action occurs outside.

The interpretation of the subjects' judgments are as follows:

- If the preselected choice is left alone it is assumed *correct*.
- If the preselected choice is changed to "ANY", it is assumed that the preselected choice *may be* okay.
- If the preselected choice is changed to any other selection, it is assumed to be *incorrect*.

63 subjects, all employees at AT&T Labs, participated in the experiment. Subjects were rewarded with a bar of chocolate of their choice. Results are presented in Table 1. In this table, "human" denotes the 30 human-judged sentences; "system" the sentences tagged by the system; and "baseline" the randomly tagged 30 sentences. Note that since we are dealing

with three predictions in each case (room, season and time of day), we have a total of 90 judgments for each of the human, system and baseline conditions. For each condition we report total errors, and *real errors*, which are errors where the subject changed the setting to something other than "ANY". As indicated in the table, all differences between the baseline and the system were significant at at least the $p < 0.01$ level on a two-sample t-test, except for the real errors for times, which were significant at the $p < 0.05$ level. All differences between the human and the system were significant at at least the $p < 0.01$ level.

So while the system does significantly better than the random baseline, it also does significantly worse than a human, and there is thus still room for improvement. An interesting general point is that the strength of the judgments seem to vary greatly across the three types — rooms, times and seasons. The subjects marked the most errors for rooms, but fewer errors for time of day or season. This suggests that, at least for the kinds of actions studied here, getting the location right is more important than getting the time of day or season right.

CONCLUSIONS & FUTURE WORK

The method we describe in this paper is fully implemented and forms part of the WordsEye system. While the technique clearly works better than a random assignment of environmental properties, it is still significantly worse than human judgments, so a major component of future work will be trying to close this gap. We are also working to expand the cov-

92.282095	175	433	24730	live room	bedroom
73.256801	17	21	7906	wash clothes	laundry room
51.118373	18	20	21056	wash hands	bathroom
35.438165	19	26	23479	drive car	garage
34.289413	18	26	21056	go to bathroom	bathroom
30.699638	16	23	21056	brush teeth	bathroom
16.510044	5	5	23479	run car	garage
16.107447	18	29	32408	wash dishes	kitchen
14.545979	4	6	7906	go to store	laundry room
14.284725	11	18	24730	go to bed	bedroom
13.490176	10	18	21056	take shower	bathroom
13.286761	5	5	32408	see in kitchen	kitchen
12.792577	4	4	24730	sit on sofa	bedroom
11.718897	11	20	24730	sit on bed	bedroom
10.559389	3	3	21056	sit on toilet	bathroom
10.329526	9	13	32408	sit at table	kitchen
9.594336	3	3	24730	hold knife	bedroom
9.594336	3	3	24730	climb over wall	bedroom
8.774370	5	11	12756	sit on floor	hallway
8.495289	5	6	32408	make breakfast	kitchen
8.240026	4	5	24730	play guitar	bedroom
8.177386	6	8	32408	eat meal	kitchen
7.971921	3	3	32408	cook meal	kitchen
7.945854	11	24	24730	leave house	bedroom
7.945854	11	24	24730	knock on door	bedroom

Figure 6: Most likely actions associated with particular rooms, after filtering with tuples extracted from the WordsEye input sentences. Columns are as in Figure 5.

erage of the technique, in particular by investigating other (implicit) features of the environment that can be predicted by corpus-based methods.

The evaluation of the system reported here evaluates only descriptions for which the system can make a prediction: in effect, then, we have considered the *precision* of the method. What we do not have a measure for at present is the *recall*, or in other words the percentage of sentences for which one ought to make a prediction, but for which we do not have the data to do so. In future work we hope to be able to say more about recall, and propose measures for evaluating it.

Finally, the work reported here considers only classes of information — time of day, location, and so forth — which were selected by hand. As reviewers have noted, and as we are aware, this is not ideal: one would like to be able to let a method loose on a large text corpus, and have it learn interesting associations of all kinds, not just associations that we happened to think of. At present it is not clear how to do this. One thing that is clear is that the more unconstrained the search is, the more "sanity checks" will have to be in place to make sure that the system does not learn useless or absurd associations. It is possible, as some reviewers have noted, that large ontologies such as Cyc may be of use in filtering more absurd associations.

REFERENCES

1. S. Abney. Partial parsing via finite-state cascades. In J. Carroll, editor, *Workshop on Robust Parsing*, pages 8-15. ESSLLI, Prague, 1996.

2. M. Berland and E. Charniak. Finding parts in very large corpora. In *Proceedings of the North American ACL*, pages 57-64, College Park, MD, 1999.

3. K. Church. A stochastic parts program and noun phrase parser for unrestricted text. In *Proceedings of the Second Conference on Applied Natural Language Processing*, pages 136-143. Association for Computational Linguistics, 1988.

4. M. Collins. *Head-Driven Statistical Models for Natural Language Parsing*. PhD thesis, University of Pennsylvania, Philadelphia, PA, 1999.

5. B. Coyne and R. Sproat. WordsEye: An automatic text-

35.729439	28	40	385312	read newspapers	morning
33.804553	32	50	385312	eat breakfast	morning
26.415083	15	32	166691	drink tea	evening
19.529204	38	48	743374	sleep on floor	night
17.679023	13	18	385312	look in mirror	morning
13.972083	7	8	385312	celebrate Easter	morning
11.686620	8	8	743374	play trumpet	night
11.240171	10	28	176501	eat lunch	afternoon
10.322243	126	213	743374	go to bed	night
9.572043	15	60	166691	eat dinner	evening
9.413257	6	14	166691	cook meal	evening
9.232992	16	32	385312	take shower	morning
8.673155	2	2	176501	see boat	afternoon
8.673155	2	2	176501	roll in front	afternoon
8.673155	2	2	176501	rake leaves	afternoon
8.573500	2	3	71317	sleep in chair	noon
8.325358	3	3	385312	throw egg	morning
8.325358	3	3	385312	take to hills	morning
7.824066	17	22	743374	sleep in bed	night

Figure 7: Most likely actions associated with particular times of day, after filtering with tuples extracted from the WordsEye input sentences. Columns are as in Figure 5.

	ERR	Real ERR	rERR	Real rERR	sERR	Real sERR	tERR	Real tERR
H	8.2** (6.2)	3.7** (3.3)	4.2** (3.0)	2.7** (2.4)	1.0** (1.1)	0.1** (0.3)	3.2** (3.0)	1.1** (1.7)
S	22.8 (8.3)	12.3 (5.7)	10.9 (2.7)	8.0 (2.4)	6.6 (2.6)	1.5 (0.9)	5.7 (3.8)	3.2 (3.7)
B	56.4** (19.7)	27.4** (7.9)	23.0** (4.4)	21.0** (3.8)	18.1** (6.8)	2.3** (1.2)	16.4** (8.7)	4.6* (4.2)

Table 1: Results of an evaluation on rooms, seasons, and times of day. "H(uman)" denotes the 30 human-judged sentence; "S(ystem)" the sentences tagged by the system; and "B(aseline)" the randomly tagged 30 sentences. Shown are the means and (standard deviations) for 63 subjects for: total errors ERR (= all changes including "ANY"); total real errors (= changes not counting "ANY"); and total and real errors for rooms (rERR), seasons (sERR) and times (tERR). Differences between the baseline and the system were significant at at least the $p < 0.01$ level on a two-sample t-test where indicated with a '**' on the mean for the baseline, and at the $p < 0.05$ level otherwise. Significances for the difference between the system and human are similarly indicated with asterisks on the means for the human-assigned judgments.

to-scene conversion system. In *SIGGRAPH 2001*, Los Angeles, CA, 2001.

6. T. E. Dunning. Accurate methods for the statistics of surprise and coincidence. *Computational Linguistics*, 19(1):61–74, 1993.

7. C. Fellbaum, editor. *WordNet: An Electronic Lexical Database*. MIT Press, Cambridge, MA, 1998.

8. M. Hearst. Automatic acquisition of hyponyms for large text corpora. In *Proceedings of the Fourteenth International Conference on Computational Linguistics (COLING)*, Nantes, France, 1992.

9. D. Hindle. Noun classification from predicate-argument structures. In *Proceedings of the 28th Annual Meeting of the Association for Computational Linguistics*, pages 268–275, Pittsburgh, PA, 1990. ACL.

10. C. Johnson, C. Fillmore, E. Wood, J. Ruppenhofer, M. Urban, M. Petruck, and C. Baker. The FrameNet project: Tools for lexicon building, version 0.7. Technical report, International Computer Science Institute, University of California, Berkeley, Berkeley, CA, 2001. www.icsi.berkeley.edu/~framenet/book.html.

11. H. Kucera and W. Francis. *Computational Analysis of Present-Day American English*. Brown University Press, Providence, 1967.

12. M. Lapata. The automatic interpretation of nominalizations. In *Proceedings ot the 17th National Conference on Artificial Intelligence (AAAI)*, pages 716–721, Austin, TX, 2000.

13. M. Lapata. A corpus-based account of regular polysemy: The case of context-sensitive adjectives. In *Pro-*

ceedings of the North American ACL, Pittsburgh, PA, 2001.

14. D. B. Lenat. Cyc: A large-scale investment in knowledge infrastructure. *Communications of the ACM*, 38(11), November 1995.

15. K. Mahesh and S. Nirenburg. Semantic classification for practical natural language processing. In *Proceedings of the Sixth ASIS SIG/CR Classification Research Workshop: An Interdisciplinary Meeting*, Chicago, IL, October 1995.

16. C. Manning and H. Schütze. *Foundations of Statistical Natural Language Processing*. MIT Press, Cambridge, MA, 1999.

17. F. Pereira, N. Tishby, and L. Lee. Distributional clustering of english words. In *Proceedings of the 31st Annual Meeting of the Association for Computational Linguistics*, Columbus, OH, 1993.

18. E. Riloff. Automatically generating extraction patterns from untagged text. In *Proceedings of the 13th National Conference on Artificial Intelligence (AAAI)*, pages 1044–1049, 1996.

19. E. Riloff and R. Jones. Learning dictionaries for information extraction by multi-level bootstrapping. In *Proceedings of the 16th National Conference on Artificial Intelligence (AAAI)*, pages 474–479, 1999.

APPENDIX: INSTRUCTIONS FOR THE RATING EXPERIMENT

You are helping us evaluate part of one component of a text-to-scene conversion system, specifically a part that attempts to make some "common sense" inferences about sentences.

You will be presented with a list of 90 sentences. Each of these sentences describes some action or situation. Associated with each sentence are three suggestions for answers to the following questions about the sentence:

- What room in a house does the described action or situation occur in?
- At what time of day does it occur?
- In what season does it occur?

The answers have been provided by an automatic procedure that attempts to predict the answers on the basis of the actions mentioned in the sentence. Your task is to decide if these predictions are correct. Specifically:

- If the prediction is correct — i.e. accords with your judgment — then leave the selection alone.
- If the prediction is clearly wrong then change it to what, in your view, is the correct answer.

In each case the correct answer is one of a selected group of answers (e.g. for season: **summer, winter, spring, autumn**), or **ANY**. **ANY** should be chosen if the action or situation does not seem to imply a particular location or time.

There may be some instances in which more than one, but not all of the answers are possible. For example something may be likely to take place at any time during the day, but not at night. In such cases you should be lenient with preselected answers that are in the acceptable set (i.e. don't change the selection), but if the preselected answer is not in the acceptable set, then choose **one member of the acceptable set** (rather than **ANY**) as your answer. For example if the provided answer is "night", and the activity could take place at any time during the day (but not at night), then select, say, "afternoon", or "morning".

For rooms, there is the additional option **NONE OF ABOVE**. This should be selected if in your view the action or situation must occur outside or in any event cannot be in one of the listed rooms.

You should try to base your judgments on what first comes to mind, rather than on deep introspection. For example, if the sentence is

John is washing his dog

and the first thing that comes to mind is that he must be in the bathroom, then that should be considered the correct answer. You may then reason that perhaps he has a tub of water in his living room, but you should avoid considering that as the correct answer.

SEAL — A Framework for Developing SEmantic PortALs

Nenad Stojanovic
Institute AIFB
University of Karlsruhe
D-76128 Karlsruhe, Germany
nst@aifb.uni-karlsruhe.de

Alexander Maedche
FZI Research Center for Information
Technologies
Haid-und-Neu Straße 10-14
D- 76131 Karlsruhe, Germany
maedche@fzi.de

Steffen Staab[♣],
Rudi Studer[♦], York Sure
Institute AIFB
University of Karlsruhe
D-76128 Karlsruhe, Germany
staab,studer,sure@aifb.uni-
karlsruhe.de

Abstract

The core idea of the Semantic Web is to make information accessible to human and software agents on a semantic basis. Hence, Web sites may feed directly from the Semantic Web exploiting the underlying structures for human and machine access. We have developed a domain-independent approach for developing semantic portals, *viz.* SEAL (SEmantic portAL), that exploits semantics for providing and accessing information at a portal as well as constructing and maintaining the portal. In this paper we focus on semantics-based means that make semantic Web sites accessible from the outside, i.e. that make semantic Web sites accessible from the outside, i.e. semantics-based browsing, semantic querying, querying with semantic similarity, and machine access to semantic information. In particular, we focus on methods for acquiring and structuring community information as well as methods for sharing information.

As a case study we refer to the AIFB portal — a place that is increasingly driven by Semantic Web technologies. We also discuss lessons learned from the ontology development of the AIFB portal.

Keywords: Ontology, Knowledge portal, Semantic Web

INTRODUCTION

The widely-agreed core idea of the Semantic Web is the delivery of data on a semantic basis. Intuitively the delivery of semantically processable data should help with establishing a higher quality of communication between the information provider and the consumer. The vison of the Semantic Web is closely related to ontologies as a sound semantic basis that is used to define the meaning of terms and hence to support intelligent providing and access to information on the Web.

The topic of this paper is a framework for developing ontol-

♣ Also Ontoprise GmbH, Haid-und-Neu Str. 7, D-76131 Karlsruhe

♦ Also Ontoprise GmbH and FZI Research Center for Information Technologies

ogy-based portal applications, namely SEAL (SEmantic PortAL) and its semantic mechanism for acquiring, structuring and sharing community information between human and/or machine agents. Ontologies constitute the foundation of our SEAL approach. The origins of SEAL lie in Ontobroker [3], which was conceived for semantic search of knowledge on the Web and also used for sharing knowledge on the Web [2]. It then developed into an overarching framework for search and presentation offering access at a portal site [17]. This concept was then transferred to further applications [1] and is currently extended into a commercial solution (*cf.* http://www.time2research.de). We here describe the SEAL core modules and its overall architecture (Section *SEAL Infrastructure and core modules*). As a case study we refer to the AIFB portal (*cf.* http://www.aifb.uni-karlsruhe.de). Thereafter, we go into several technical details that are important for human and machine access to a semantic portal.

In particular, we describe a general approach for semantic ranking (Section *Semantic Ranking*). The motivation for semantic ranking is that even with accurate semantic access, one will often find too much information. Underlying semantic structures, *e.g.* topic hierarchies, give an indication of what should be ranked higher on a list of results. Also, we present mechanisms to deliver and collect machine-understandable data (Section *RDF Outside*) and discuss how this approach establishes the road to the Semantic Web. These mechanisms extend previous means for better digestion of Web site data by software agents. Finally, we describe some experiences made during the development of the ontology for our AIFB portal (Section *Experience with ontology engineering*). Before we conclude, we give a short survey of related work.

SEAL INFRASTRUCTURE AND CORE MODULES

In this section, we first elaborate on the general architecture for SEAL (SEmantic PortAL) and then we explain functionalities of its core modules. As a running example we refer to the AIFB portal, which aims at presenting information to human and software agents taking advantage of semantic structures.

Architecture

The overall architecture and environment of SEAL is depicted in Figure 1. The *backbone* of the system consists of the *knowledge warehouse, i.e.* the ontology and knowledge base,

and the *Ontobroker* system [3], *i.e.* the principal inferencing mechanism. The latter functions as a kind of middleware run-time system, possibly mediating between different information sources when the environment becomes more complex than it is now.
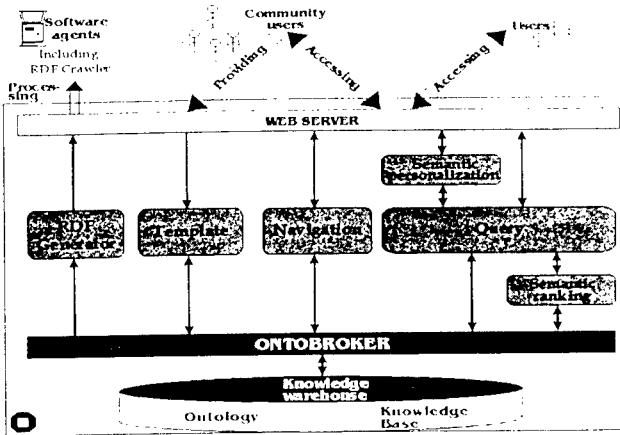

Figure 1: SEAL - System architecture

At the front end one may distinguish between three types of *agents*: *software agents*, *community users* and *general users*. All three communicate with the system through the *Web server*. The three different types of agents correspond to three primary modes of interaction with the system.

First, remote applications *(e.g.* software agents) may process information stored in the portal. For this purpose, the *RDF generator* presents RDF facts through the Web server. Software agents with *RDF crawlers* may collect the facts and, thus, have direct access to semantic knowledge stored at the Web site.

Second, community users and general users can access information contained at the Web site. Two forms of accessing are supported: navigating through the portal by exploiting hyperlink structure of documents and searching for information by posting queries. The hyperlink structure is partially given by the portal builder, but it may be extended with the help of the *navigation* module. The navigation module exploits inferencing capabilities of the inference engine in order to construct conceptual hyperlink structures. Searching and querying is performed via the *query* module. In addition, the user can personalise the search interface using the *semantic personalization* module and/or rank retrieved results according to semantic similarity (done by the module for *semantic ranking*). Queries also take advantage of the Ontobroker inferencing capabilities.

Third, only community users can provide data. In the AIFB portal application, typical information community user contribute include personal data, information about research areas, publications and other research information. For each type of information they may contribute there is (at least) one concept in the ontology. By retrieving parts of the ontology, the *template* module may semi-automatically produce suitable HTML forms for data input. The community users fill in these forms and the template module stores the data in the knowledge warehouse.

Core modules
The core modules have been extensively described in [17]. In order to give the reader a compact overview we here shortly survey their function. In the remainder of the paper we delve deeper into those aspects that have been added or considerably extended recently, *viz.* semantic ranking (Section *Semantic Ranking*), and semantic access by software agents (Section *RDF Outside*).

Ontobroker
The Ontobroker system [3] is a deductive, object-oriented database system operating either in main memory or on a relational database (via JDBC). It provides compilers for different languages to describe ontologies, rules and facts. Beside other usage, it is also used as an inference engine (server) within SEAL. It reads input files containing the knowledge base and the ontology, evaluates incoming queries, and returns the results derived from the combination of ontology, knowledge base and query. The possibility to derive additional factual knowledge from given facts and background knowledge considerably facilitates the life of the knowledge providers and the knowledge seekers. For instance, one may specify that if a person belongs to a research group of the institute AIFB, he also belongs to AIFB. Thus, it is unnecessary to specify the membership to a research group *and* to AIFB. Conversely, the info seeker does not have to take care of inconsistent assignments, *e.g.* ones that specify membership to an AIFB research group, but that have erroneously left out the membership to AIFB.

Knowledge warehouse
The knowledge warehouse [17] serves as repository for data represented in the form of F-Logic statements [6]. It hosts the ontology, as well as the data proper. From the point of view of inferencing the difference is negligible, but from the point of view of maintaining the system the difference between ontology definition and its instantiation is useful. The knowledge warehouse is organised around a relational database, where facts and concepts are stored in a reified format. It states relations and concepts as first-order objects and it is therefore very flexible with regard to changes and amendments of the ontology.

Navigation module
Beside the hierarchical, tree-based hyperlink structure which corresponds to the hierarchical decomposition of the domain, the navigation module enables complex graph-based semantic hyperlinking, based on ontological relations between concepts (nodes) in the domain. The conceptual approach to hyperlinking is based on the assumption that semantically relevant hyperlinks from a Web page correspond to conceptual relations, such as memberOf or hasPart, or to attributes, like hasName. Thus, instances in the knowledge base may be presented by automatically generating links to all related instances. For example, on personal Web pages there are, among others,

hyperlinks to Web pages that describe the corresponding research groups, secretary and professional activities (*cf.* Figure 2, higher part).
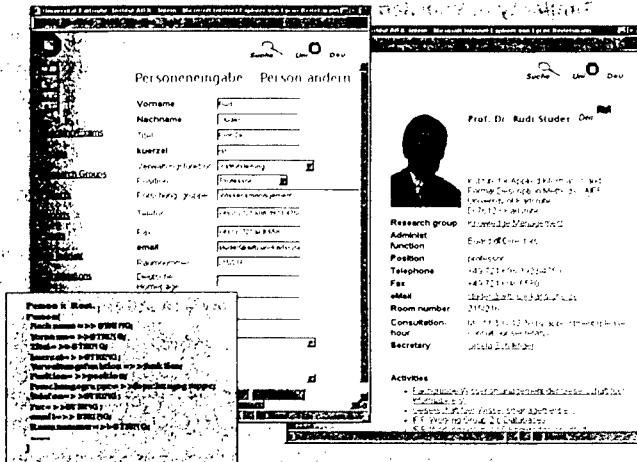

Figure 2: Templates generated from concept definitions

Query module

The query module puts an easy-to-use interface on the query capabilities of the F-Logic query interface of Ontobroker. The portal builder models Web pages that serve particular query needs. For this purpose, selection lists that restrict query possibilities are offered to the user.


Figure 3: Query form based on definition of concept Person

The selection lists are compiled using knowledge from the ontology and/or the knowledge base. For instance, the query interface for persons on the AIFB portal, allows to search for people according to research groups they are members of. The list of research groups is dynamically filled by an F-Logic query and presented to the user for easy choice by a drop-down list (*cf.* snapshot in Figure 3).

Even simpler, one may associate a hyperlink with an F-Logic query that is dynamically evaluated when the link is hit. More complex, one may construct an isA, a hasPart, or a hasSub-

topic tree, from which query events are triggered when particular nodes in the tree are selected.

Template module

In order to facilitate the contribution of information by community users, the template module generates an HTML form for each concept that a user may instantiate. For instance, the AIFB portal includes an input template (*cf.* Figure 2, left upper part) generated from the concept definition of person (*cf.* Figure 2, lower left). The data is later on used by the navigation module to produce the corresponding person Web page (*cf.* Figure 2, right part). In order to reduce the data required for input, the portal builder specifies which attributes and relations are derived from other templates. For example, in our case the portal builder has specified that project membership is defined in the project template. The coordinator of a project enters information which persons are participants of the project and this info is used when generating the person Web page taking advantage of a corresponding inverse relationship, between relations worksIn and memberOf.

Ontology lexicon

The different modules described here make extensive use of the lexicon component of the ontology (*cf. Section Experience with ontology engineering*). The most prevalent use is the distinction between English and German. In the future we envision that one may produce more adaptive Web sites making use of the explicit lexicon. For instance, we will be able to produce short descriptions when the context is sufficiently narrow, *e.g.* working with ambiguous acronyms like ASP (e.g. active server pages *vs.* active service providers).

SEMANTIC RANKING

This section describes the architecture component „Semantic Ranking" which has been developed in the context of our framework. First, we will introduce and motivate the requirement for a ranking approach with a small example. Second, we will show how the problem of semantic ranking may be reduced to the comparison of two knowledge bases. Query results are reinterpreted as „query knowledge bases" and their similarity to the original knowledge base without axioms yields the basis for semantic ranking. Thereoy, we reduce our notion of similarity between two knowledge bases to the similarity of concept pairs.

Let us assume the following ontology:

```
1:  Person::Object [worksIn =>> Project].
2:  Project::Object [hasTopic =>> Topic].
3:  Topic::Object [subtopicOf =>> Topic].          (1)
4:  FORALL X,Y,Z  Z[hasTopic ->>Y] <-
      X[subtopicOf ->>Y] and Z[hasTopic ->>X].
```

To give an intuition of the semantic of the F-Logic statements, in line 1 one finds a concept definition for a Person being an Object with a relation worksIn. The range of the relation is restricted to Project.

Ontology axioms like the one given in line 4 (1) use this syntax to describe regularities. Line 4 states that if some Z has

157

topic X and X is a subtopic of Y then Z also has topic Y. Let us further assume the following knowledge base:

```
5:  KnowledgeManagement:Topic.
6:  KnowledgeDiscovery:
    Topic[subtopicOf ->>KnowledgeManagement].
7:  Gerd:Person[worksIn ->>OntoWise].       (2)
8:  OntoWise:
    Project[hasTopic ->>KnowledgeManagement].
9:  Andreas:Person[worksIn ->>TelekomProject].
10: TelekomProject:
    Project[hasTopic ->>KnowledgeDiscovery].
```

Definitions of instances in the knowledge base are syntactically very similar to the concept definition in F-Logic. In line 6 the instance KnowledgeDiscovery of the concept Topic is defined. Furthermore, the relation subtopicOf is instantiated between KnowledgeDiscovery and KnowledgeManagement. Similarly in line 7, it is stated that Gerd is a Person working in the project OntoWise.

Now, an F-Logic query may ask for all people who work in a knowledge management project by:

```
FORALL Y,Z <- Y[worksIn ->> Z] and      (3)
Z:Project[hasTopic ->> KnowledgeManagement]
```

which may result in the tuples (Gerd, OntoWise) and (Andres, TelekomProjet). Obviously, both answers are correct with regard to the given knowledge base and ontology, but the question is, what would be a plausible ranking for the correct answers. This ranking should be produced from a given query without assuming any modification of the query.

Reinterpreting queries

Our principal consideration builds on the definition of semantic similarity that we have first described in [10]. There, we have developed a measure for the similarity of two knowledge bases. Here, our basic idea is to reinterprete possible query results as a „query knowledge base" and compute its similarity to the original knowledge base while abstracting from semantic inferences. The result of an F-Logic query may be reinterpreted as a *query knowledge base* (*QKB*) by the following approach.

An F-Logic query is of the form or can be rewritten into the form (*cf.* negation requires special treatment):

$$\text{FORALL } \overline{X} \leftarrow \overline{P}(\overline{X}, \overline{k}) \qquad (4)$$

With $\overline{X}$ being a vector of variables $(X_1, ..., X_n)$, $\overline{k}$ being a vector of constants, and $\overline{P}$ being a vector of conjoined predicates. The result of a query is a two-dimensional matrix M of size $m \times n$, with n being the number of result tuples and m being the length of $\overline{X}$ and, hence, the length of the result tuples. Hence, in our example above: $\overline{X} := (Y, Z)$, $\overline{k} := (\text{"Knowl-edgeManagement"})$, $\overline{P} := (P_1, P_2)$, $P_1(a,b,c) := a[\text{worksIn} ->> b]$, $P_2(a,b,c) := b[\text{hasTopic} ->> c]$ and

$$M := (M_1, M_2) = \begin{pmatrix} \text{Gerd} & \text{Andreas} \\ \text{OntoWise} & \text{TelekomProjekt} \end{pmatrix}. \qquad (5)$$

Now, we may define the query knowledge base $i(QKB_i)$ by

$$QKB_i := \overline{P}(M_i, \overline{k}). \qquad (6)$$

Similarity of knowledge bases

The similarity between two objects (concepts and or instances) may be computed by considering their relative place in a common hierarchy H. H may, but need not be a taxonomy H. For instance, in above example we have a categorization of research topics, which is not a taxonomy.

Our principal measures are based on the cotopies of the corresponding objects as defined by a given hierarchy H, *e.g.* an ISA hierarchy H, a part-whole hierarchy, or a categorization of topics. Here, we use the *upwards cotopy* (*UC*) defined as follows:

$$UC(O_i, H) := \{O_j \mid H(O_i, O_j) \vee O_j = O_i\} \qquad (7)$$

Concepts are taxonomically related by the irreflexive, acyclic, transitive relation $H, (H \subset C \times C)$. $H(C_1, C_2)$ means that C_1 is a subconcept of C_2. UC is overloaded in order to allow for a set of objects M as input, *viz.*

$$UC(M, H) := \bigcup_{O_i \in M} \{O_j \mid H(O_i, O_j) \vee O_j = O_i\} \qquad (8)$$

Based on the definition of the upwards cotopy (*UC*) the object match (*OM*) is defined by:

$$OM(O_1, O_2, H) := \frac{|UC(O_1, H) \cap UC(O_2, H)|}{|UC(O_1, H) \cup UC(O_2, H)|} \qquad (9)$$

Basically, *OM* reaches when two concepts coincide (number of intersections of the respective upwards cotopies and number of unions of the respective cotopies is equal); it degrades to the extent to which the discrepancy between intersections and unions increases (a *OM* between concepts that do not share common super-concepts yields value 0).

The match introduced above may easily be generalized to relations using a relation hierarchy H_R into account. Hence, it may also be generalized to instantiated relations. Thus, the predicate match (PM) for two n-ary predicates P_1, P_2 is defined by a mean value. Thereby, we use the geometric mean in order to reflect the intuition that if the similarity of one of the components approaches 0 the overall similarity between two predicates should approach 0 as well — which need not be the case for the arithmetic mean:

$$PM(P_1(I_1, ..., I_n), P_2(J_1, J_n)) := \qquad (10)$$

$$\sqrt[n+1]{OM(P_1, P_2, H_R) \cdot OM(I_1, J_1, H) \cdot ... \cdot OM(I_n, J_n, H)}$$

This result may be averaged over an array of predicates. We here simply give the formula for our actual needs, where a query knowledge base is compared against a given knowledge base KB:

$$Simil(QKB_i, KB) = Simil(\overline{P}(M_i, \overline{k}), KB) := \qquad (11)$$

$$\frac{1}{|\overline{P}|} \sum_{P_j \in \overline{P}} \max_{Q(M_i, \overline{k}) \in KB} PM(P_j(M_i, k), Q(M_i, \overline{k})).$$

158

Example. We here give a small example for computing UC and OM based on a given categorization of objects *H*. Figure 4 depicts the example scenario.
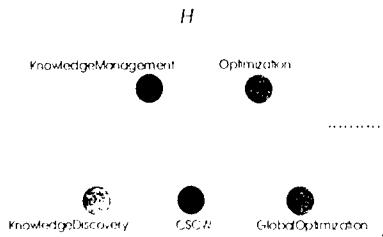

Figure 4: Example for computing UC and OM

The upwards cotopy UC(KnowledgeDiscovery, *H*) is given by {KnowledgeDiscovery, KnowledgeManagement}. The upwards cotopy UC(Optimization, *H*) computes to {Optimization}. Computing the object match OM between KnowledgeManagement and Optimization results in 0, the object match between CSCW and KnowledgeDiscovery computes to 1/3 .

For instance, we compare the two result tuples from our example above with the given knowledge base: Our first result tuple is $M_1^T :=$ (Gerd, OntoWise). Then, we have the query knowledge base (QKB_1) :

        Gerd[worksIn ->>OntoWise] .   (12)

        OntoWise[hasTopic ->>KnowledgeManagement] .

Its relevant counterpart predicates in the given knowledge base (KB) are

        Gerd[worksIn->>OntoWise] .   (13)

        OntoWise[hasTopic->>KnowledgeManagement]

This is a perfect fit. $Simil(QKB_1, KB)$ computes to 1.

Our second result tuple is $M_2^T :=$ (Andreas, TelekomProject). Then, we have the query knowledge base (QKB_2) :

        Andreas:Person[worksIn ->>TelekomProject]   (14)

        TelekomProject[hasTopic->>KnowledgeManagement] .

Its relevant counterpart predicates in the given knowledge base (KB) are

        Andreas[worksIn ->>TelekomProject] .   (15)

        TelekomProject[hasTopic ->>KnowledgeDiscovery] .

Hence, the similarity of the first predicate indicates a perfect fit and evaluates to 1, but the congruency of TelekomProject[hasTopic->>KnowledgeManagement] with TelekomProject[hasTopic->>KnowledgeDiscovery] measures less than 1. The instance match of KnowledgeDiscovery and KnowledgeManagement returns ½ in the given topic hierarchy. Therefore, the predicate match PM returns $\sqrt[3]{1 \cdot 1 \cdot \frac{1}{2}} \approx 0.79$. Thus, overall ranking of the second result is based on ½(1+0.79)=0.895. Therefore, the AIFB portal will display (Gerd, OntoWise) as the first result and (Andreas, TelekomProject) as the second one.

Remarks on semantic ranking. The reader may note some basic properties of the ranking: *(i)* similarity of knowledge bases is an asymmetric measure, *(ii)* the ontology defines a conceptual structure useful for defining similarity, *(iii)* the core concept for evaluating semantic similarity is cotopy defined by a dedicated hierarchy. The actual computation of similarity depends on which conceptual structures *(e.g.* hierarchies like taxonomy, part-whole hierarchies, or topic hierarchies) are selected for evaluating conceptual nearness. Thus, similarity of knowledge bases depends on the view selected for the similarity measure. Ranking of semantic queries using underlying ontological structures is an important means in order to allow users a more specific view onto the underlying knowledge base. The method that we propose is based on a few basic principles:

- Reinterprete the combination of query and results as query knowledge bases that may be compared with the explicitly given information.
- Give a measure for comparing two knowledge bases, thus allowing rankings of query results.

The reader should be aware that our measure may produce some rankings for results that are hardly comparable. For instance, results may differ slightly because of imbalances in a given hierarchy or due to rather random differences of depth of branches. In this case, ranking may perhaps produce results that are not better than unranked ones — but the results will not be any worse either

RDF OUTSIDE — FROM A SEMANTIC WEB SITE TO THE SEMANTIC WEB

In the preceding sections we have described the components and the underlying techniques of the SEAL framework and its instantination in the AIFB portal. Since we want to embed SEAL-based portals into the Semantic Web, we have developed means for RDF-capable software agents to process the portal data. Therefore, we have built an automatic RDF GENERATOR that dynamically generates RDF statements on each of the static and dynamic pages of the semantic knowledge portal.

Our current AIFB portal is „Semantic Web-ized" using RDF facts instantiated and defined according to the underlying AIFB ontology. This means, that *e.g.* for each person from the institute, contact information (telephone, fax, e-mail address) as well as professional information (research-area, research-group, projects involved in) are available for processing from world-wide software agents which understand this form of metadata representation.

The RDFMAKER established in the Ontobroker framework (*cf.* [3]) was a starting point for building the RDF GENERATOR. The idea of RDFMAKER was, that from Ontobroker's internal knowledge warehouse RDF statements are generated.

RDF GENERATOR follows a similar approach and extends the principal ideas. In a first step it generates an RDF(S)-

based ontology that is stored on a specific XML namespace, *e.g.* in our concrete application:

```
xmlns:aifb= "http://ontobroker.semanticweb.org/
        ontologies/aifb-2001-01-01.rdfs#"
```

Additionally, it queries the knowledge warehouse. Data, *e.g.* for a person, is checked for consistency, and, if possible, completed by applying the given F-Logic rules. We here give a short example of information, namely name, phone, fax, e-mail and supervisor, which may be generated and stored on a specific home-page of a researcher in the position of PhD student :

```
<rdf:RDF    xmlns:rdf="http://www.w3.org/1999/02/22-
rdf-syntax-ns#"
        xmlns:aifb="http://ontobroker.    semantic-
web.org/ontologies/aifb-2001-01-01.rdfs#">
  <aifb:PhDStudent rdf:ID="per:ama">
    <aifb:name>Alexander Maedche</aifb:name>
    <aifb:phone>+49-(0)721-608 558</aifb:phone>
    <aifb:fax>+49-(0)721-608 6580</aifb:fax>
<aifb:email>maedche@fzi.de</aifb:email>
        <aifb:supervisor rdf:resource="http://www.
aifb.uni-karlsruhe.de/studer.html#per:rst"/>
    </aifb:PhDStudent>
</rdf:RDF>
```

RDF GENERATOR is a configurable tool, in some cases one may want to use inferences to generate materialized, complete RDF descriptions on a home page, in other cases one may want to generate only ground facts of RDF. Therefore, RDF GENERATOR allows to switch axioms on and off in order to adopt the generation of results to varying needs. In order to collect RDF annotateted information from dedicated sources, software agents have to crawl that portion of the Web – by using RDF CRAWLER.

The RDF CRAWLER (*cf.* RDF CRAWLER free downloadable at http://ontobroker.semanticweb.org/rdfcrawler) is a tool which downloads interconnected fragments of RDF from the internet and builds a knowledge base from this data. Building an external knowledge base for the AIFB portal (its researcher, its projects, its publications, ...) becomes easy using the RDF CRAWLER and the machine-processable RDF data currently defined on AIFB portal. In general, RDF data may appear in Web documents in several ways. We distinguish between pure RDF (files that have an extension like „*.rdf"), RDF embedded in HTML and RDF embedded in XML. Our RDF CRAWLER uses RDF-API (*cf.* RDF-API free downloadable at http://www-db.stanford.edu /~melnik/rdf/api.html) that can deal with the different embeddings of RDF as described above.

EXPERIENCES WITH ONTOLOGY ENGINEERING

The conceptual backbone of our SEAL approach is an ontology. For our AIFB portal application, we had to model concepts relevant in this setting. As SEAL has been maturing, we have developed a methodology for setting up ontology-based knowledge systems [18]. Our approach (*cf.* Figure 5) is mainly based on [16] and [5] but focuses on the application-driven development of ontologies. We here describe some

experiences made during the ontology development for our AIFB portal.
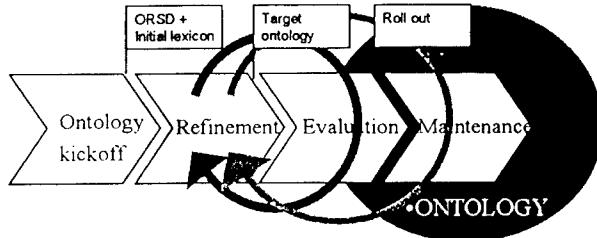
Figure 5: Ontology Development

Kickoff phase for ontology development

Setting up requirements for the AIFB ontology we had to deal mainly with modeling the research and teaching topics addressed by different groups of our institute and personal information about members of our institute. We took ourselves as an „input source" and collected a large set of lexical entries for research topics, teaching related topics and personal information, which represent the lexicon component of the ontology. By the sheer nature of these lexical entries, the ontology developers were not able to come up with all relevant lexical entries by themselves. It was necessary to go through several steps with domain experts *(viz.* our colleagues) in the refinement phase.

Refinement phase

We started to develop a baseline taxonomy that contained a heterarchy of research topics identified during the kickoff phase. An important result for us was to recognize that categorization was not based on an isA-taxonomy, but on a much weaker HasSubtopic relationship. *E.g.* „KDD" is a subtopic of „Knowledge Management", which means that it covers some aspects of „Knowledge Management" — but it does not reflect inheritance provided by an isA-taxonomy. It then took us three steps to model the currently active ontology. In the first step, lexical entries were collected by all members from the institute. Though we had already given the possibility to provide a rough categorization, the categories modeled by non-knowledge engineers were not oriented towards a model of the world, but rather towards the way people worked in their daily routine. Thus, their categorization reflected a particular rather than a shared view onto the domain. A lesson learned from this was that people need an idea about the nature of ontologies to make sound modeling suggestions. It was helpful to show existing prototypes of ontology-based systems to the domain experts.

In the second step, we worked towards a common understanding of the categorization and the derivation of implicit knowledge, such as „someone who works in logic also works in theoretical computer science" and inverseness of relations, *e.g.* „an author has a publication" is inverse to „a publication is written by an author".

In the third step, we mapped the gathered lexical entries to concepts and relations and organized them at a middle level.

Naturally, this level involved the introduction of more generic concepts that people would usually not use when characterizing their work (such as „optimization"), but it also included „politically desired concepts", because one own's ontology exhibits one's view onto the world. Thus, the ontology may become a political issue.

Modeling during early stages of the refinement phase was done with pen and paper, but soon we took advantage of our ontology environment OntoEdit (cf. free downloadable at http://www.ontoprise.de/) that supports graphical ontology engineering at an epistemological level as well as formalization of the ontology. Like in other ontology engineering projects, the formalization of the ontology is a non-trivial process where the ontology engineer has to draw the line between ontology and knowledge base. Therefore our final decisions were much disputed.

Evaluation phase

After all we found that participation by users in the construction of the ontology was very good and met the previously defined requirements, as people were very interested to see their work adequately represented. Some people even took the time to learn about OntoEdit. However, the practical problem we had was that our environment does not yet support an ontology management module for cooperative ontology engineering. We embedded the ontology into our AIFB portal. It contains around 170 concepts (including 110 research topics) and 75 relations. This version is still running, but we expect maintenance to be a relevant topic soon. Therefore we are collecting feedback from our users - basically colleagues and students from our institute.

RELATED WORK

This section positions our work in the context of existing Web portals and also relates our work to other basic methods and tools that are or could be deployed for the construction of community Web portals, especially to related work in the area of semantic ranking of query results.

Related Work on Knowledge Portals. One of the well-established Web portals on the Web is Yahoo (cf. http://www.yahoo.com). In contrast to our approach Yahoo only utilizes a very light-weight ontology that solely consists of categories arranged in a hierarchical manner. Yahoo offers keyword search (local to a selected topic or global) in addition to hierarchical navigation, but is only able to retrieve complete documents, i.e. it is not able to answer queries concerning the contents of documents, not to mention to combine facts being found in different documents or to include facts that could be derived through ontological axioms. We get rid of these shortcomings since our portal is built upon a rich ontology enabling the portal to give integrated answers to queries.

The Ontobroker project [3] lays the technological foundations for the AIFB portal. On top of Ontobroker the portal has been built and organizational structures for developing and maintaining it have been established.

The approach closest to Ontobroker is SHOE [8]. In SHOE, HTML pages are annotated via ontologies to support information retrieval based on semantic information. Besides the use of ontologies and the annotation of Web pages the underlying philosophy of both systems differs significantly: SHOE uses description logic as its basic representation formalism, but it offers only very limited inferencing capabilities. Ontobroker relies on Frame-Logic and supports complex inferencing for query answering. Furthermore, the SHOE search tool does not provide means for a semantic ranking of query results. A more detailed comparison to other portal approaches may be found in [17].

Related Work on Semantic Similarity. Since our semantic ranking is based on the comparison of the query knowledge base with the given ontology and knowledge base, we relate our work to the comparison of ontological structures and knowledge bases (covering the same domain) and to measuring the similarity between concepts in a hierarchy. Although there has been a long discussion in the literature about evaluating knowledge-bases [12], we have not found any discussion about comparing two knowledge bases covering the same domain that corresponds to our ranking approach. Similarity measures for ontological structures have been investigated in areas like cognitive science, databases or knowledge engineering (cf. e.g., [14, 13, 15, 9]). However, all these approaches are restricted to similarity measures between lexical entries, concepts, and template slots within one ontology.

Closest to our measure of similarity is work in the NLP community, named semantic similarity [14] which refers to similarity between two concepts in a isA-taxonomy such as the WordNet or CYC upper ontology. Our approach differs in two main aspects from this notion of similarity: Firstly, our similarity measure is applicable to a hierarchy which may, but not need be a taxonomy and secondly it is taking into account not only commonalties but also differences between the items being compared, expressing both in semantic-cotopy terms. This second property enables the measuring of self-similarity and subclass-relationship similarity, which are crucial for comparing results derived from the inferencing processes, executed in the background.

Conceptually, instead of measuring similarity between isolated terms (words), that does not take into account the relationship among word senses that matters, we measure similarity between „words in context", by measuring similarity between Object-Attribute-Value pairs, where each term corresponds to a concept in the ontology. This enables us to exploit the ontological background knowledge (relations between concepts) in measuring the similarity, which expands our approach to a methodology for comparing knowledge bases.

From our point of view, our SEAL framework is rather unique with respect to the collection of methods used and the functionality provided. We have extended our community portal approach that provides flexible means for providing, integrating and accessing information [17], semantic ranking of generated answers and a smooth integration with the evolv-

ing Semantic Web. All these methods are integrated into one uniform environment, the SEAL framework.

CONCLUSION

In this paper we have shown our comprehensive approach SEAL for building semantic portals. In particular, we have focused on three issues.

First, we have described the general architecture of the SEAL framework, which is also used for our real-world case study, the AIFB portal. The architecture integrates a number of components that we have also used in other applications, like Ontobroker, the navigation and query module. Second, we have extended our semantic modules to include a larger diversity of intelligent means for accessing the Web site, *viz.* semantic ranking and machine access by crawling. Third, we have presented some experiences made during the ontology development for our case study - AIFB portal.

For the future, we see a number of new important topics appearing on the horizon. For instance, we consider approaches for ontology learning in order to semi-automatically adapt to changes in the world and to facilitate the engineering of ontologies [11].

Currently, we work on providing intelligent means for providing semantic information, *i.e.* we elaborate on a semantic annotation framework that balances between manual provisioning from legacy texts *(e.g.* Web pages) and information extraction [4], [7].

Finally, we envision that once semantic Web sites are widely available, their automatic exploitation may be brought to new levels. Semantic Web mining considers the level of mining Web site structures, Web site content, and Web site usage on a semantic rather than at a syntactic level yielding new possibilities, *e.g.* for intelligent navigation, personalization, or summarization, to name but a few objectives for semantic Web sites.

REFERENCES

[1]Angele, J., Schnurr, H.-P., Staab, S., and Studer, R. The times they are changing- the corporate history analyzer. In D. Mahling, U. Reimer, editors, *Proceedings of the Third International PAKM Conference*, pages 1.1-1.11, Basel, October, 2000.

[2] Benjamins, V.R., Fensel, D., Decker, S., and Perez, A.G. (KA): Building ontologies for the internet. *International Journal of Human-Computer Studies*, 51(1):687–712, 1999

Systems, 16(1):26–34, 2001.

[3] Decker, S., Erdmann, M., Fensel, D., and Studer, R. Ontobroker: Ontology Based Access to Distributed and Semi-Structured Information. In R. Meersman et al., editors, *Database Semantics: Semantic Issues in Multimedia Systems*, pages 351–369. Kluwer Academic Publisher, 1999.

[4] Erdmann, M., Maedche, A., Schnurr H.-P., and Staab, S. From manual to semi-automatic semantic annotation: About ontology-based text annotation tools. *ETAI Journal - Section on Semantic Web*, 6(2001).

[5] Gomez-Perez, A. A framework to verify knowledge sharing technology. *Expert Systems with Application*, 11(4):519–529, 1996.

[6] Kifer, M., Lausen, G., Wu, J. Logical Foundations of Object-Oriented and Frame-Based Languages. *Journal of the ACM* , Vol. 42, 741-883, 1995

[7] Handschuh, S., Staab, S., Maedche, A. CREAM - Creating relational metadata with a component-based, ontology-driven annotation framework. *ACM K-CAP 2001*. October, Vancouver, in press

[8] Heflin, J. and Hendler, J. Searching the Web with SHOE. In *Artificial Intelligence for Web Search*. AAAI Workshop.WS-00-01,pages 35–40. AAAI Press, 2000.

[9] Hovy, E. Combining and standardizing large-scale, practical ontologies for machine translation and other uses. In *Proc. Of the First Int. Conf. on Language Resources and Evaluation (LREC)*, 1998.

[10] Maedche, A. and Staab, S. Discovering conceptual relations from text. In *Proceedings of ECAI-2000*, pages 321-324 IOS Press, Amsterdam, 2000.

[11] Maedche, A., Staab, S. Ontology Learning for the Semantic Web. *IEEE Intelligent Systems*, 16(2):72–79, 2001.

[12] Menzis, T.J. Knowledge maintenance: The state of the art. *The Knowledge Engineering Review*, 10(2), 1998.

[13] Rada, R., Mili, H., Bicknell, E. and Blettner, M. Development and application of a metric on semantic nets. *IEEE Trans. on Systems, Man, and Cybernetics*, 19(1), pages 17-30, 1989.

[14] Resnik, P. Using information content to evaluate semantic similarity in a taxonomy. In *Proceedings of IJCAI-95*, pages 448-453, Montreal, Canada, 1995.

[15] Richardson, R., Smeaton, A.F. and Murphy, J. Using Wordnet as knowledge base for measuring semantic similarity between words. Technical Report CA-1294, Dublin City University, School of Computer Applications, 1994.

[16] Schreiber, G., Akkermans, H., Anjewierden, A., de Hoog, R, Shadbolt, N., Van de Velde, W., and Wielinga, B. *Knowledge Engineering and Management — The CommonKADS Methodology*. The MIT Press, 1999.

[17] Staab, S., Angele, J., Decker, S., Hotho, A., Maedche, A., Schnurr, H.-P., Studer, R. and Sure, Y. AI for the Web — ontology-based community Web portals. In *AAAI 2000/IAAI 2000*, pages 1034-1039, AAAI Press/MIT Press, 2000.

[18] Staab, S., Schnurr, H.-P., Studer, R. and Sure, Y. Knowledge processes and ontologies. *IEEE Intelligent*.

Ontology-Based Metadata Generation from Semi-Structured Information

Heiner Stuckenschmidt
Center for Computing Technologies
University of Bremen
P.O.B: 33 04 40 D-28334 Bremen, Germany
heiner@tzi.de

Frank van Harmelen
AIdministrator BV, Amerfoort, and
AI Department, Vrije Universiteit Amsterdam
De Boelelaan 1081a, 1081HV Amsterdam, The Netherlands
frankh@cs.vu.nl

Abstract

Content-related metadata plays an important role in intelligent information systems. Especially on the world-wide web meaningful metadata describing the contents of a web-site is the key to intelligent retrieval and access of information. Metadata description standards like RDF and RDF schema have been developed and work in progress addresses the use of ontologies to provide a logical foundation for metadata. However, the acquisition of appropriate metadata is still a problem. The main part of the paper is concerned with the specification of ontologies and metadata models. We describe the Spectacle approach, a knowledge-based approach for metadata validation and generation as well as tools related to the ontology language OIL. We conclude that the specification of ontologies and the generation of metadata models are processes that supplement each other and propose a method for semi-automatic generation of metadata models on the basis of ontologies.

Motivation

The information society demands large-scale availability of data and information. With the advent of the World Wide Web, huge amounts of information is available in principle, but the size and the inherent heterogeneity of the Web make it difficult to find and to access useful information. A suitable information source must be located which contains the data needed for a given task. Once the information source has been found, access to the data therein has to be provided. A common approach to this problem is to provide so-called metadata, i.e. data about the actual information. This data may cover very different aspects of information: technical data about storage facilities and access methods co-exist with content descriptions and information about intended uses, suitability and data quality. Concerning the problem of finding and accessing information, metadata help to find, to access and to interpret information.

In this paper we focus on a specific type of metadata, namely metadata related to the contents of a web-page. A typical approach to capture this kind of metadata is so-called web-page categorization [10]. Here, web pages as a whole are assigned to a set of classes representing a certain topic area the page belongs to. In order to apply this approach there has to be a set of classes to be used as targets for the classification task. The idea of using ontologies in order to define these classes is straightforward and does not need too much argumentation.

A problem that remains is the classification itself which can be a tremendous effort considering the size of normal websites or even the web itself. There is a need for automatic or semi-automatic support for the classification process that has already been observed by others. Jenkins and others, for example, use text mining technology in order to generate RDF models describing the contents of web-pages [9]. It has been argued that web page classification can be significantly improved by using additional information like other kinds of metadata [10] or linguistic features [1]. We propose an approach that exploits another kind of additional information namely the syntactic structure of a web page. This can be done because it has been shown that it is possible to identify syntactic commonalities between web-pages information about similar topics [5]. We use an existing approach for classifying web-pages on the basis of their structure and show how this approach can be used to relate web-pages to a pre-existing ontology in such a way that the formal semantics of the ontology remains available for consistency checking and filtering of web-pages.

The paper is organized as follows. In section we introduce the Spectacle approach for semi-automatically classifying individual web-pages based on their structure. In section we present the use of Spectacle for the generation of contents metadata on the basis of ontologies in some more details. The current state of the technology used as well as two case studies using the approach are the topic of section . We conclude with a critical view on the scaleability of the approach and topics for further research brought up by the case studies.

The Spectacle Approach

We have developed an approach to solve the problems of completeness, consistency and accessibility of metadata identified above. This is done on the basis of rules which must hold for the information found in the Web site, both the actual information and the metadata (and possibly their relationship). This means that besides providing Web site contents and metadata, an information providers also formulate classification rules (also called: integrity constraints) which should hold on this information. An inference engine then applies these integrity constraints to identify the places in the Web site which violate these constraints. This approach has been implemented in the Spectacle Workbench, developed by the Dutch company AIdministrator (http://www.aidministrator.nl). In this section, we will describe the different steps of our approach. Formulating and applying classification criteria and integrity constraints is done in a three step process [13].

Constructing a Web-site Model

The first step in our approach to content-based verification and visualization of web-pages is to define a model of the contents of the web-site. Such a model identifies classes of objects on our web-site, and defines subclass relationships between these classes. For example, pages can be about water, soil, air, energy, etc. Each of these types of pages can again be subdivided into new subclasses: water-pages can be about waste-water, drinking water, river-drainage, etc. This leads to a hierarchy of pages which is based on page-contents, such as the example shown in Figure 1.
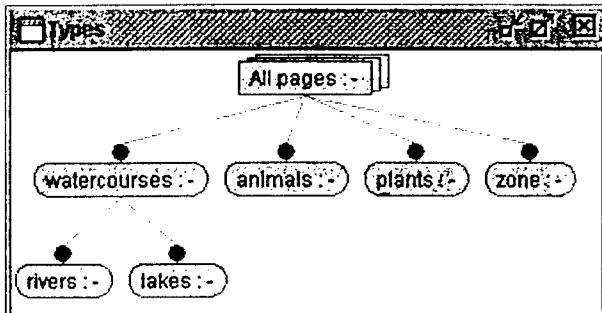
Figure 1: An Example Classification Tree

A subtle point to emphasize is that the objects in this ontology are objects on the web-site, and not objects in the real-world described by the web-site. For example, the elements in the class "rivers" are not (denotations of) different rivers in a specific region, but they are *web-pages* (in this case: web-pages talking about rivers). As a result, any properties we can validate for these objects are properties of the *pages on the web-site*, as desired for our validation purposes.

Defining Syntactic Criteria for Classes

The first step only defines the classes of our ontology, but does not tell us which instances belong to which class. In the second step, the user defines rules (compare [11]) that determine which Web pages will be members of which class. In this section, we will briefly illustrate these rules by means of three examples.

Figure 2 specifies that a rule is about "watercourses" if the keyword "Gewsser" appears in the meta-information of the web-page. The rule succeeds if for example the following code appears in the web-page:

```
<META NAME="Keywords" CONTENT="Gewsser, Bericht"/
```

In the typical case, a page belongs to a class if the rule defined for that class succeeds for the page. However, it is also possible to define classes by negation: a page belongs to a class when the corresponding rule fails on that page. This is indicated by a rectangle in the class-hierarchy (instead of a rounded box).
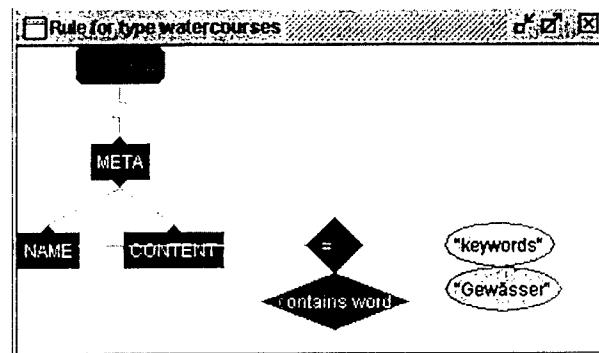
Figure 2: Example of a Classification Rule Using Metadata

Classifying Individual Pages

While the human user of Spectacle performs the previous steps, the next step is automatic. The definition of the hierarchy in the first step and the rules in the second step allow the Spectacle inference engine to automatically classify each page in the class hierarchy. Note that classes may overlap (a single page may belong to multiple classes).

The rule format has been defined in such a way as to provide sufficient expressive power while still making it possible to perform such classification inference on large numbers of pages (many thousands in human-acceptable response time).

164

Generating Metadata

After these three steps, we have a class hierarchy that is populated with all the pages of a given site. Such a populated class hierarchy can be stored in a combined RDF and RDF Schema format [3]. The following statements are taken from the RDF Schema encoding of the Spectacle type hierarchy. The first three show how of the types "watercourses", "lake" and "river" and their sub-type relationship are encoded in standard RDF Schema.

```
<rdfs:Class rdf:ID="watercourses"/>

<rdfs:Class rdf:ID="lake">
    <rdfs:subClassOf rdf:resource="#water"/>
</rdfs:Class>

<rdfs:Class rdf:ID="river">
    <rdfs:subClassOf rdf:resource="#water"/>
</rdfs:Class>
...
```

The following is an example of an RDF encoding of instance information: the URL mentioned in the "about" attribute is declared to be a member of the class "water" (and consequently of all its super-types, by virtue of the RDF Schema semantics).

```
<rdf:Description about=
    "http://www.umwelt.bremen.de/buisy/scripts/buisy.asp?
    doc=Badegewaesserguete+Bremen">
    <rdf:type resource="#watercourses"/>
</rdf:Description>
...
```

These automatically generated annotations constitute an aggregated description of a web site that can be used to get an overview of its content.

Ontology-Based Metadata Generation

In this section we propose a method to generate content-related metadata in terms of a web-page categorization. The idea behind the method is based on the following observations: Ontologies are intentional models of information contents with a well-defined logical basis which can be used for reasoning. Metadata, on the other hand, are extensional models summarizing existing information and can therefore be extracted from an information source. We conclude that both can supplement each other in the process of generating metadata. In the following we describe an integrated method to generate metadata models on the basis of content ontologies. We illustrate the method with experiments conducted using an existing information system.

Building Content Ontologies

Ontologies have set out to overcome the problem of implicit and hidden knowledge by making the conceptualization of a domain (e.g. environmental protection) explicit. This corresponds to one of the definitions of the term ontology most popular in computer science [7]:

"An ontology is an explicit specification of a conceptualization."

An ontology is used to make assumptions available about the meaning of a term. In the context of the general metadata architecture this means that terms are specified by restrictions on their interpretation and their relation to other terms used in the metadata description. In this section we describe how an ontology about the contents of a web-site can be built and used for reasoning.

The OIL language has been developed in the context of the On-To-Knowledge Project (www.ontoknowledge.org) as a proposal for a language for the specification and exchange of ontologies [6]. OIL tries to provide a core set of features that have been widely accepted to be useful for the description of vocabularies and terminologies. OIL combines object-oriented modeling primitives, reasoning facilities from Description Logics, and a tight interaction with RDF and XML.

A couple of tools have been developed to support the application of the OIL language on the World-Wide Web, including the Ontology Editor OILed which has already been proven useful for real applications [12]. OILed can be used to develop ontologies that contain the following language elements.

Class Hierarchies: The basic structure of an OIL ontology is a set of classes arranged in a subclass hierarchy. Each class is a place-holder for a specific set of entities. We can use class hierarchies to define different sub-disciplines of for example environmental protection, namely emission control, nature preservation, soil protection and water pollution control. These disciplines might be used to structure an information system and can provide guidance for content-based search or navigation. Therefore a clear notion of these terms is important to provide meaningful metadata.

Slot Definitions: OIL is capable of defining binary relations (so-called slots) between classes in the hierarchy. Range and domain of these relations can be restricted to special classes described by their name or an intentional description of their members (see below). Further it is possible to define inverse relations, hierarchies of relations and to assign a couple of mathematical properties (e.g. transitivity) to relations. In our case an 'about' relation, for example, connects disciplines (referred to as *topic area*) to specific contents or spatial locations. Note that we can use Boolean operations on class names to describe this fact.

Concept Definitions: Classes can not only be defined by their position in the class hierarchy, but also by constraints on objects they may relate to. Figure 3 shows a simple class definition.
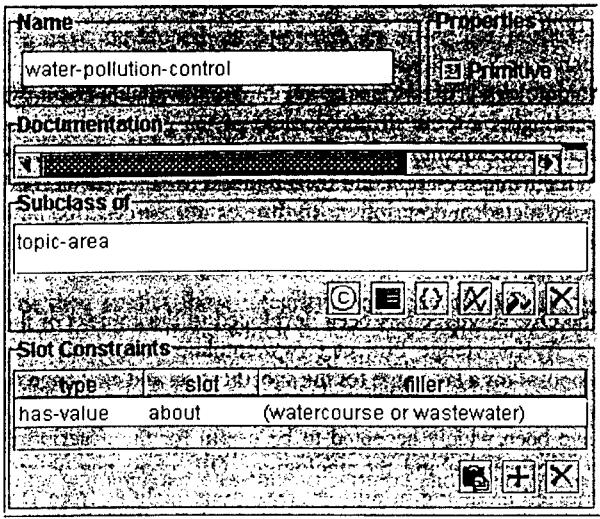
Figure 3: The Class Definition Editor

The figure shows the (strongly simplified) definition of the topic area *water-pollution-control*. The definition claims that the contents of each instance of that topic area are concerned with *watercourses* or with *wastewater*. This definition restricts the way of how a piece of information can be interpreted. For example, it does not allow us to classify an information item which is only concerned with animals as belonging to the topic area of water pollution control.

Individuals: The last feature of the OIL language we need in order to build an ontology about the vocabulary used to describe the contents of a web-based information system (in our case an environmental information system) are instances of classes. In our case, we can use individuals to describe existing objects in the world like real watercourses, lakes and rivers, but also to refer to pages in the information system and relate these pages to real world objects they contain information about. The dummy page shown in the picture, for example, is said to be about the 'Sodenmattsee', a lake in the district 'Huchting'.

Assigning Pages to Classes

The definition of a content ontology provided us with an intentional model of the domain. The next step is to relate this model to real information from the system. This step, also referred to as grounding, is a crucial one, because it is time-consuming and error-prone. The size of modern information systems forces us to provide some tool support. We claim that the Spectacle Workbench is a very helpful tool for this task, because it automatically classifies pages into ontological concepts on the basis of syntactic rules. In order to make use of the system's capabilities we have to import the previously built contents ontology and define classification rules for each concept from the ontology. Note that the ontology already defines criteria for class membership, but does not

define criteria that can be checked on a web page. Consequently, we define two sets of criteria for each class in the ontology.

- **Intentional Criteria:** Restrictions on the way a term might be interpreted. This is done in the content ontology.
- **Extensional Criteria:** Properties of information related to that class allowing us to find it in on a web-site: These criteria are specified using the Spectacle System.

In order to use Spectacle for the definition of syntactic criteria, we import the subclass hierarchy from the content ontology into the Workbench and proceed by defining syntactic classification rules for each class. In principle, we have three possibilities:

1. using metadata to classify web-pages

2. using arbitrary page-contents to classify web-pages

3. using external properties of web-pages for classification

The first possibility applies if already some kind of metadata have been included in the system. A typical example is the use of keywords. We discussed this example in section . The rule displayed in figure 2, for example, can be used as a syntactic criterion for the class watercourses. In order to distinguish the sub-types of watercourses we can no longer use existing metadata. In this case, we can use Spectacle to perform a free-text search in the body of web-pages and look for the German terms corresponding to lake and river. Figure 4 shows the result of the search.

Figure 4: Pages classified to belong to the class 'lake'

From the results displayed in figure 4 we can easily generate a metadata description of the pages classified to belong to the class lakes. The metadata description of the page highlighted in the screenshot is the following:

166

```
<rdf:Description about=
"http://www.umwelt.bremen.de/buisy/scripts/bgb/bewertung.asp?
        See-Sodenmattsee&Jahr=2000"
    <rdf:type resource="#lakes"/>
</rdf:Description>
```

Corresponding descriptions are generated for all pages on the web-site which could be classified. In parallel, we supplement the contents ontology by creating an individual for every page and assigning it to the corresponding concepts that have been detected.

Ontology-Based Post-Processing

One of the major benefits of using the OIL language for specifying ontologies is the availability of reasoning support for a limited number of tasks concerned with ontology management. The reasoning support is based on Description Logic, i.e. on the correspondence of OIL with the language $\mathcal{SHIQ}$. OIL specifications are translated into this logic and standard reasoning techniques are used to support the following tasks:

Consistency Checking: The reasoner is able to check the satisfiability of the logical model of the ontology. In particular, inconsistent concept definitions are detected. If we, for example, defined animals to have four legs and we try to include an instance of the class animal with five legs, the reasoner will find the contradiction.

Computation of Subclass Relations: An ontology normally contains two different kinds of sub-class relations: explicitly defined relations from the class hierarchy and implicit subclass relations implied by the logical definitions of concepts. The latter can be detected using the reasoning support of the OIL language and included into the ontology thus completing it.

A special case of the computation of subclass relation is the automatic classification of individuals. OIL allows us to describe an individual by its relation to other individuals without naming all classes it belongs to. The reasoner will find the classes we omitted in the definition. An exmaple would be if we only defined our dummy page to be about the 'Sodenmattsee' without assigning it to a special topic area. However, we stated that the domain of the 'about' relation is the class topic area and we defined water-pollution-control to be concerned with watercourses. This information provided and the fact that the 'Sodenmattsee' is a lake and therefore a watercourse enables the reasoner to decide that our dummy page should be classified as belonging to the topic area 'water pollution control'.

OIL uses the FaCT reasoner, a system which implements highly optimized algorithms for providing the above mentioned reasoning support [8]. FaCT is implemented in LISP, but it offers a CORBA interface that allows easy access to the system using a well-defined interface [2]. The OILed Editor can be directly connected with the reasoner providing reasoning support at development time. Inconsistencies are

highlighted and missing subclass relations are added. Therefore, OILed and FaCT offer a comfortable development environment for ontologies.

Using this environment we can check the result of the metadata generation for consistency. This is necessary because the criteria used to describe classes in the Spectacle systems only refer to syntactic structures of the page contents. Especially, Spectacle has no possibility to check whether the classification of a page makes sense from a logical point of view. For example, we can include a description of the administrative units in our ontology and classify pages according to the unit which is concerned with the specific topic of the page. We will define the units to be mutually disjoint because the competency is strictly separated. If we now classify one page to belong to both units we get a clash in the logical model. In this case, we have to check the page and assign the right administrative unit by hand. Thus the logical models helps us to find shortcomings of the generated model.

The second benefit of the logical grounding of the metadata model is the possibility to derive hidden class memberships. This is important because the RDF metadata schema makes some assumptions about implicit knowledge. Examples of these assumptions can be found in [4]. We use the following axiom as an example:

$$\frac{\mathcal{T}(r, \mathtt{rdf:type}, c_1) \wedge \mathcal{T}(c_1, \mathtt{rdfs:subClassOf}. c_2)}{\mathcal{T}(r. \mathtt{rdf:type}. c_2)}$$

The equation states that every resource r (i.e. web-page) that is member of class c_1 (indicated by the triple $\mathcal{T}(r, \mathtt{rdf:type}, c_1)$) is also member of class c_2 ($\mathcal{T}(r, \mathtt{rdf:type}. c_2)$) if c_1 is a subclass of c_2 ($\mathcal{T}(c_1, \mathtt{rdfs:subClassOf}, c_2)$). This correlation can easily be computed using the FaCT reasoner by querying all super-concepts of a given concept. The result of this query can be used to supplement the description of a page. The description of the page referred to above, for example, will be extended with the following statement.

```
...
<rdf:type resource="#watercourse"/>
...
```

In the same way, other axiomatic properties of RDF schema can be implemented in order to produce a more complete metadata model.

Applying the Method

The generated metadata model can be used in various ways. In the introduction, we already mentioned the general application areas search, access and interpretation. In this section, we will briefly discuss the use of metadata for intelligent search for web pages. We implemented a universal search engine which relies on an ontology-based metadata model

in order to search for web resources with certain properties. The search engine imports the content ontology and asks the user for a concept to be queried. Based on the definition of that concept (i.e. the attached slots) a query interface is generated that allows the user to specify restrictions on the slot fillers. The query engine searches the metadata model and returns all pages that fall under these restrictions.

The search engine is intended to be used as a component in web-based information systems rather than the complete web. In such a system we can assume the existence of a common ontology which can be used as a basis for generating the metadata model necessary to support the search process.

Tool Support and Interaction

We are currently implementing the approach described above making use of mostly pre-existing technology already mentioned in the previous sections. Figure 5 shows the interaction of these tools, namely the OILed ontology editor, the FaCT reasoning system, the spectacle workbench and our own search engine ASK-Me (Automatic Selection of Knowledge resources based on Metadata).
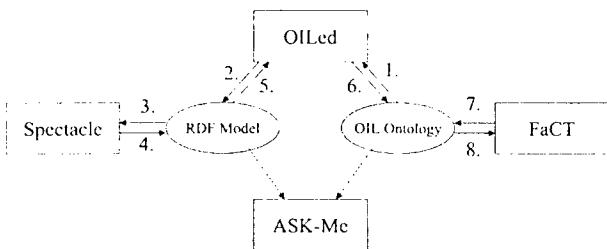
Figure 5: Interaction of Tools in the Overall Process

The figure depicts a typical run through the metadata generation process that contains the following steps.

1. Import of Content Ontology into the ontology editor.

2. Export of the ontology as RDFS model.

3. Import of the Class Hierarchy in to the Spectacle workbench as basis for the classification.

4. Export of instantiated ontology, where each web-page is described and assigned to one or more classes in the hierarchy

5. Import of the instances into the editor in order to supplement the content ontology.

6. Export of the instantiated ontology in OIL format

7. Import of the ontology into the FaCT reasoner for consistency checking and computation of subsumption relations.

8. Export of the verified and completed ontology in OIL

Finally the search engine is supplied with the metadata model as well as with the ontology in order to provide a content filtering service on the basis of a target concept specified by the user. The system uses the Ontology in order to relate the query concept to concepts assigned to web-pages as well as the RDF model in order to retrieve the web-pages assigned to these classes.
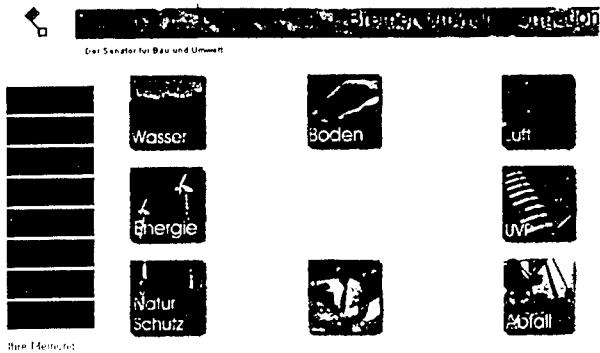
At the moment, the right hand side of the figure, namely the interaction between editor, reasoner and search engine is completely implemented. We are currently working on the RDF part. Open tasks include the alignment of the RDF Models supported by Spectacle one and OILed on the other hands. Further we have to extend the search engine to completely work on RDF instead of a relational database we use at the moment.

Case Studies

We have two different case studies we use in order to evaluate the approach presented. The first one the examples found in this paper are taken from is concerned with the environmental information system of the City of Bremen and has already been finalized. The second one is a rather new attempt to provide an integrated information system for scientific services provided by organizations in the city of Bremen. This project called City-of-Science has just started. We briefly describe these case studies in the following.

BUISY: An Environmental Information System The advent of web-based information systems came with an attractive solution to the problem of providing integrated access to environmental information according to the duties and needs of modern environmental protection. Many information systems were set up either on the Internet in order to provide access to environmental information for everybody or in intranets to support monitoring, assessment and exchange of information within an organization. One of the most recent developments in Germany is BUISY, an environmental information system for the city of Bremen which has been developed by the Center for Computing Technologies of the University of Bremen in cooperation with the public authorities. The development of the system was aimed at providing unified access to the information existing in the different organizational units for internal use as well as for the publication of approved information on the internet.

Figure 6 shows the main screen of the BUISY system with the main topic area covered by the system. The first step of the proposed method now consists of the development of an ontology about the domain. The definition of the topic areas and the vocabulary used within these areas is of major interest. In the previous section we already sketched the idea of how such an ontology could be built and showed some example definitions as screenshots from the OILed Editor. The result of this first step will be an extended RDF model that contains additional modeling primitives of the OIL language. Such a model can be generated by OILed without

Figure 6: The main topic areas of the BUISY System

further modeling effort. Below is a corresponding definition of the topic-area water pollution control that we already mentioned in the last section.

```
<rdfs:Class rdf:ID="water pollution control">
    <rdfs:subClassOf>
        <rdfs:Class rdf:about "#topic area"></rdfs:Class>
    </rdfs:subClassOf>
    <oil:hasPropertyRestriction>
        <oil:HasValue>
            <oil:onProperty rdf:resource="#about">
            </oil:onProperty>
            <oil:toClass>
                <oil:Or>
                    <oil:hasOperand>
                        <rdfs:Class rdf:about="#watercourse">
                        </rdfs:Class>
                    </oil:hasOperand>
                    <oil:hasOperand>
                        <rdfs:Class rdf:about="#wastewater">
                        </rdfs:Class>
                    </oil:hasOperand>
                </oil:Or>
            </oil:toClass>
        </oil:HasValue>
    </oil:hasPropertyRestriction>
</rdfs:Class>
```

In the course of our case study on the BUISY system eight groups of AI students with some experience in knowledge representation and knowledge-based systems independently built ontologies covering the contents of the BUISY system. They used the Spectacle System in order to assign web-pages to concepts from the ontology and conducted experiments with querying the system using concept expressions.

City Of Science: An Information System for Scientific Services
The government of the City of Bremen recently recognized the need to support technology transfer from research organizations to the local industry. One of the activities started in connection with this goal is the establishment of an information system for scientific services. The idea is to provide a uniform interface and intelligent access methods to profiles of potential providers of scientific services. A standard profile has been created which each provider has to specify according to the kinds of services he wants to advertise.

Examples of information given by each organization are the following:

Type of Organization: Providers of scientific services are categorized due to their legal status and organizational nature. Categories include universities, research institutes, consortia and companies.

Area of Expertise: A rough description of the areas of research the corresponding service provider works in and claims to have expertise.

Technical Equipment: Non standard equipment needed to perform special tasks. Typical examples are laboratories but this notion also includes special function buildings.

There are also other kinds of information like previous projects or mode of funding. However, we only refer to the three properties above.

In a case study, we investigate how contents related metadata can improve the search methods provided to the user in order to find the service he needs. We just finished the development of the content ontology defining the properties mentioned above on a logical basis. Each service provider is modeled as an instance of the concept service-provider with further specifications of the properties using properties from the city of science namespace denoted by coc. An example is the following:

```
<coc:ResearchConsortium rdf:about="MARUM">
    <coc:equipment rdf:resource="ResearchShip"/>
    <coc:equpment rdf:resource="ResearchPost"/>
    <coc:expertise rdf:resource="Climate"/>
    <coc:equipment rdf:resource="Laboratories"/>
    <coc:expertise rdf:resource="MarineReserach"/>
    <coc:expertise rdf:resource="EnvironmentalResearch"/>
    <coc:part-of rdf:resource="UniversityOfBremen"/>
</coc:ResearchConsortium>
```

The next step in this case study will be an investigation of how the Spectacle system can be used in order to semi-automatically describe new profiles added by service providers by annotating descriptions like the one shown above.

Discussion and Future Work
We discussed the role of metadata for intelligent search, access, and interpretation of information in web-based information systems. We described the Spectacle approach for the generation of metadata models and the OIL approach to ontology building. We concluded that both approaches can be combined to achieve a semi-automatic procedure to build metadata models. We also described the current status of the implementation and two applications of the approach.

Lessons learned from the case studies: Our approach of combining ontology building with metadata generation comes with benefits for both previously existing approaches. On one hand, metadata generation with Spectacle takes advantage of the logical foundation provided by the ontology in terms of consistency checking and subsumption reasoning. On the other hand it helps to acquire ontological knowledge by providing a tool for the automatic population of the ontology with individuals. The BUISY Case study showed that users with some knowledge in AI are able to build content ontologies and to apply the Spectacle system for generating metadata. However, the definition of syntactical criteria for web-pages of a certain class is still a difficult and time-consuming task which requires some knowledge about the information to be annotated. In order to avoid the effort of analyzing the whole web-site we are currently developing an approach for automatically learning page structure from examples and partial specifications. The city of science case study revealed that it is often not enough to analyze web-pages as a whole. In the case of the city of science project, metadata related to special aspects described in single paragraphs has to be generated. We therefore have to refine the analysis to include single elements on a web-page.

In general, an open problem of the approach is the application to arbitrary web resources. The approach relies on the existence of a single ontology all pages can be related to. At the moment this can only be achieved by restricting the application to information systems with a well-defined domain. Intranets, for example, fulfill this requirement.

REFERENCES

1. Roberto Basili, Alessandro Moschitti, and Maria Teresa Pazienza. Nlp-driven ir: Evaluating performances over a text classification task. In B. Nebel, editor, *Proceedings of the 13th International Joint Conference on Artificial Intelligence IJCAI-01*, 2001.

2. S. Bechhofer, I. Horrocks, P. F. Patel-Schneider, and S. Tessaris. A proposal for a description logic interface. In *Proc. of DL'99*, pages 33–36, 1999.

3. D. Brickley, R. Guha, and A. Layman. Resource description framework (rdf) schema specification. Working draft, W3C, August 1998. http://www.w3c.org/TR/WD-rdf-schema.

4. Pierre-Antoine Champin. Rdf tutorial. Available at http://www710.univ-lyon1.fr/ champin/rdf-tutorial/, June 2000.

5. M. Craven, D. DiPasquo, D. Freitag, A. McCallum, T. Mitchell, K. Nigam, and S. Slattery. Learning to construct knowledge bases from the world wide web. *Artificial Intelligence*, 118(1-2):69–113, 2000.

6. D. Fensel, I. Horrocks, F. Van Harmelen, S. Decker, M. Erdmann, and M. Klein. Oil in a nutshell. In *12th International Conference on Knowledge Engineering and Knowledge Management EKAW 2000*, Juan-les-Pins, France, 2000.

7. T.R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2), 1993.

8. I. Horrocks. The FaCT system. In H. de Swart, editor, *Automated Reasoning with Analytic Tableaux and Related Methods: International Conference Tableaux'98*, number 1397 in Lecture Notes in Artificial Intelligence, pages 307–312. Springer-Verlag, Berlin, May 1998.

9. C. Jenkins, M. Jackson, P. Burdon, and J. Wallis. Automatic rdf metadata generation for resource discovery. *Computer Networks*, 31:1305–1320, 1999.

10. John M. Pierre. On the automated classification of web sites. *LInkping Electronic Articles in Computer and Information Science*, 6, 2001.

11. M.-C. Rousset. Verifying the world wide web: a position statement. In F. van Harmelen and J. van Thienen, editors, *Pro-ceedings of the Fourth European Symposium on the Validation and Verification of Knowledge Based Systems (EUROVAV97)*, 1997.

12. R. Stevens, I. Horrocks, C. Goble, and S. Bechofer. Building a reason-bale bioinformatics ontology using oil. In A. Gomez-Perez, M. Gruninger, H. Stuckenschmidt, and M. Uschold, editors, *Proceedings of the IJCAI-01 Workshop on Ontologies and Information Sharing*, August 2001.

13. Frank van Harmelen and Jos van der Meer. Webmaster: Knowledge-based verification of web-pages. In M. Ali and I. Imam, editors, *Proceedings of the Twelfth International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, (IEA/AEI99)*, LNAI. Springer Verlang, 1999.

Discovery of Ontologies from Knowledge Bases[†]

Hendra Suryanto and Paul Compton

School of Computer Science and Engineering
University of New South Wales
Sydney 2052, Australia
{hendras, compton}@cse.unsw.edu.au

Abstract

Current approaches to building knowledge-based systems propose the development of an ontology as a precursor to building the problem-solver. This paper outlines an attempt to do the reverse and discover interesting ontologies from systems built without the ontology being explicit. In particular the paper considers large classification knowledge bases used for the interpretation of medical chemical pathology results and built using Ripple-Down Rules (RDR). The rule conclusions in these knowledge bases provide free-text interpretations of the results rather than explicit classes. The goal is to discover implicit ontological relationships between these interpretations as the system evolves. RDR allows for incremental development and the goal is that the ontology emerges as the system evolves. The results suggest that approach has potential, but further investigation is required before strong claims can be made

Keywords

Knowledge acquisition, machine learning, ontology.

INTRODUCTION

Current knowledge acquisition (KA) methodologies, based on knowledge-level modelling frameworks, attempt to build a number of models, before building the particular problem solver; e.g KADS and CommonKADS [22], Protege2000 [26]. There is an increasingly strong emphasis in this on developing an ontology, and modern KA tools are increasingly tools for developing an ontology. Although this approach facilitates re-use there are still major maintenance issues in making additions and corrections.

With RDR the emphasis is not on the preparation, but in making it as simple as possible to make corrections. Hence, the only domain model is the data representation scheme[4]. The assumption is, that if sufficient heuristics are added, they will compensate for the lack of a substantial

ontology. The focus of the approach is therefore to make the addition of each incrementally added piece of knowledge as simple and as reliable as possible. Although this approach facilitates KA and maintenance, it does not facilitate reuse.

Our aim in this study is to see if some aspects of an ontology can be automatically learned from the rules of an RDR knowledge base. Our particular focus here is to discover an ontology for the rule conclusions. Our strategy is to compare the bodies of rules that give different classifications. Because the same conclusion may be given by more than one rule, the rule comparisons are combined to give overall or 'average' relationships between classes.

There is no class structure for these conclusions. When a Multiple Classification RDR (MCRDR) KB makes an error, the task of the expert is to specify the correct conclusion and identify the attributes and values that justify this conclusion. In adding the conclusion, the expert can select from a list of pre-existing conclusions organized into broad categories, but can also simply type in a new conclusion. In medical pathology result interpretation, the evaluation domain here, the conclusions added by the pathologist may provide advice to the referring clinician on patient diagnosis, management, how treatment is progressing, whether the tests ordered were appropriate, what tests might still be necessary or any combination of the above. It is quite clear to both the expert and the receiver of the advice what information is being provided in the free text interpretation, but these interpretations are a long way from the well defined classes of a formal ontology. A task analysis would assess this domain as a classification problem, but this does not imply well defined classes. Hence the problem is not only that disjuncts for a class (separate rule paths) may be scattered across the KB, but that the same class may be represented by different text strings. Such text strings may cover a combination of different classes.
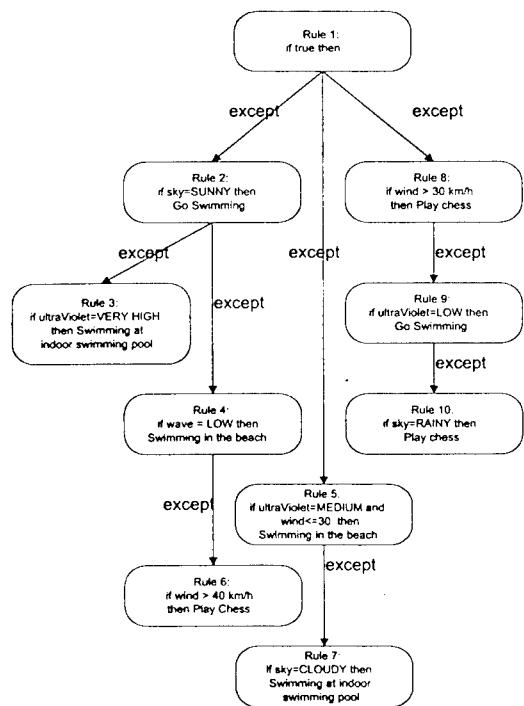
The question perhaps arises of whether it would be more appropriate to start with a well-developed ontology, as suggested by most KA researchers, rather than allow this situation to evolve. There have been a number of RDR papers

[†] An earlier version of this paper appeared in the ECAI'2000 Workshop on Ontology Learning

over the years presenting data on the practical advantages of the incremental KA approach provided by RDR [7]. Commercial RDR systems for Clinical Pathology are now available from Pacific Knowledge Systems Pty. Ltd and in routine use in a number of laboratories. These systems are used to add clinical comments to pathology reports to provide advice to the referring clinician. The knowledge bases can be built and maintained while in routine clinical use. Pathologists require about two hours training to be able to build rules and the KA time is about one minute per rule, and high levels of accuracy are rapidly achieved. Knowledge bases range in size from hundreds to thousands of rules. Since development is incremental there is minimal impact on the laboratory's normal work-flow. These informal, but industrial-use results, strongly confirm the previous RDR research results. Rather than leading to the conclusion that it would be better to start with a well-developed ontology, this experience perhaps suggests that it may be simpler to re-develop rather than re-use! However, if we can discover the ontologies implicit in these incrementally developed systems, we may be able to have the best of both worlds.

These studies have used knowledge bases made available by Pacific Knowledge Systems (http://www.pks.com.au).

Figure 1. Example of the simple MCRDR knowledge based system

ONTOLOGY LEARNING OVERVIEW
The ontology learning (OL) approach we use could be applied to any rule-based system; it is specified here in terms

of MCRDR. A class in MCRDR is the set of disjunct rule paths giving the same conclusion.

The ontology learning method discovers three type of relation between these classes: mutual-exclusivity, similarity and subsumption. All of these relations are assessed by comparing the rule paths between classes and are based on the commonalities and differences between the conditions used in rules in the rule paths

Mutual-exclusivity is the relation between 2 classes such that both of them can not appear together. (In MCRDR, we can have more that one rule path providing an interpretation for a case). A simple example of how this relation might be used is as follows. One might discover that the interpretations "bald" and "pregnant" are 90% mutually exclusive. (We will discuss how to calculate this 90% value later).

Subsumption is the relation between two classes such that class2 is a specialization of class1. In current implementations of MCRDR, it is possible for the expert to define intermediate classes. This suggests two kinds of subsumption in MCRDR, i.e. semantic subsumption and syntactic subsumption. An intermediate conclusion gives syntactic subsumption. Although formal semantic subsumption is not necessary in MCRDR, the exception structure of MCRDR generally implies semantic subsumption. That is, if class2 is always an exception of class1, we say that semantically class1 subsumes class2.

Similarity between two classes is a measure of how alike two classes are. If class1 is very similar to class2, the system might suggest to the expert that class1 is perhaps the same as class2. Another motivation for a similarity metric is to visualize the classes in 2D space, to support the expert if they wish to split a large RDR-KBS into several more homogenous KBS.

Similar notions, but unrelated to ontology learning have been investigated by Hamming [12] for coding and information theory. Flax [8] also investigated the hamming distance between 2 logical formulae. Our investigation is based on similar notions since we compute a distance between two classes by taking the average of all the pairs of difference between the disjunctions for each class.

A complexity in this is that attributes tend to be multi-valued rather than boolean. So that in rules, conditions can subsume each other, be disjoint, etc. For example age >10 subsumes age 50, whereas age >40 and age <10 are clearly disjoint. The measures to be proposed need to address the way in which conditions based on multi-valued attributes interrelate.

ONTOLOGY LEARNING AND MCRDR
A rule path consists of all conditions from all predecessor rules plus conditions of this particular node's rule. For example, from figure 1 :

rule path 6: class *Play Chess* if wind > 40, wave=LOW, sky=SUNNY

Firstly we discover the class relations between rules for subsumption, mutual-exclusivity and similarity. Secondly we specify some compound relations using these three basic relations. We use these compound relations to extract matching relations from all the basic relations. Finally we build a class model using the matched compound relations.

The central idea of the technique is to group all rules of the same class and compute a quantitative measurement (from 0 to 1) for each relation (subsumption, mutual-exclusivity, similarity) between every pair of classes. We use this quantitative measurement as an informal confidence measure as to whether these relations exist. The algorithm will be discussed in detail below, but when applied to the example in figure 1 it gives: class *Go Swimming* subsumes class *Swimming in the beach* with degree of confidence 0.83; class *Play Chess* and class *Go Swimming* are mutually exclusive with degree of confidence 0.17; class *Go Swimming* and class *Play Chess* have degree of similarity 0.50.

This quantitative measure enables us to group different examples of the class and provides information on whether across these examples, a class tends to subsume another class (for example, class *Go Swimming* subsumed class *Swimming in the beach* with degree of confidence 0.83. Using this quantitative measure we can say class *Go Swimming* tends to subsume or almost subsumes class *Swimming in the beach*, rather than simply saying class *Go Swimming* subsumes *class Swimming in the beach* or class *Go Swimming* does not subsume class *Swimming in the beach*.

These measures become interesting when applied to real examples such as: class *Mild hypothyroidism may contribute to hyperlipidaemia* is subsumed by class *Hypothyroidism may exacerbate hyperlipidaemia* with degree of confidence 0.75.

Boolean values are inadequate for subsumption, mutual-exclusivity, and similarity, relations in real domains. We found that in a 3710 rule KBS we considered, there were only 4 subsumption relations with degree of confidence 1.0; 181 mutual-exclusive relations with degree of confidence 1.0 and no similarity relations with degree of confidence 1.0.

One of the advantages of learning from rules is that we can assume that irrelevant attributes have already been discarded. This is significant as in our application domain there are hundreds of attributes. Gaines [9] argues that a rule in a knowledge base is worth many cases for learning. We adopt the same viewpoint and note that although there is research on combining KA and machine learning and using background knowledge in machine learning, there seems little research so far in learning from a KBS rather than from cases [19], [10].

The immediate precursor of this work [19] applied formal concept analysis to ontology discovery in knowledge bases. This provided a useful way to explore concepts in a knowledge base, but because of the complexity of the conceptual lattice it was necessary to consider sub-sections of the lattice, selected by the user or by a simple nearest neighbour algorithm [20]. The critical difference from the work here, is that in formal concept analysis the difference between individual concepts is emphasised. Here we attempt to combine all the concepts that represent a class and consider relations between classes rather than between concepts. In the knowledge bases we considered, there was an average of about 10 rule paths (concepts) per conclusion, making some sort of merging very desirable. Some earlier results using this approach have been published in [25]

RIPPLE DOWN RULES (RDR)

Although our technique could be applied to probably all rule-based classification KBS, we have developed it specifically to deal with RDR knowledge bases. RDR is an attempt to deal with the problem that experts never explain how they reach a conclusion, rather they justify why their conclusion is appropriate and this justification varies with the context in which it is given [4]. With the RDR approach the expert only corrects errors made by the KB. The expert decides a case should have a different conclusion from the one given by the KBS and justifies this by indicating features in the case which distinguish it from a case for which the conclusion provided by the KBS would have been appropriate. This results in the exception structure shown in figure 1 illustrating where rule 10, for example, has been added to correct the error that rule 9's conclusion was inappropriate for a case. This particular structure supports multiple conclusions for a case. Any case for which a rule is added is also stored with that rule and is known as a 'cornerstone case'. When a new rule is added, any 'cornerstone cases' that can reach the parent rule are retrieved. The expert has to add a rule which excludes all of these cornerstone cases or he or she can decide to add this conclusion as an extra conclusion for these cases. In practice, even with hundreds of cornerstone cases that might reach the rule, the expert selects conditions to make a sufficiently precise rule after no more than two or three cornerstone case have been shown to the expert. This is essentially a verification and validation technique incorporated into knowledge acquisition [15].

The RDR exception structure provides a compact representation of knowledge [3], [11], [16], [21], [23]. In particular, despite the random order in which cases are presented, and the likelihood of experts providing less than ideal rules, manually built RDR systems produce compact KBs [5], [13], [24]. Initial RDR development was concerned with classification tasks, first single and later multiple classification (The knowledge bases considered here ae multiple classification). RDR has been extended to configuration [18], heuristic search [2], document retrieval [14] and a more general RDR system for construction tasks has been proposed [6].

THE CLASS RELATIONS

The class relations model shows the relations subsumption, mutual-exclusivity and similarity between classes and the degree of confidence that the particular relation exists. We note that the measures we derive are strictly heuristic. Other superior and perhaps more well founded measures may be possible. The results here represent simply a first attempt at carrying out this type of analysis. The second point to note is that these relations have to deal with non-boolean as well as boolean data.

The technique is based on set theory. If we have two sets A and B, then the following relations between them are possible. $\{A \subset B, B \subset A, A \cap B \neq \varnothing, A \cap B = \varnothing, A = B, A \neq B\}$.

Figure 2
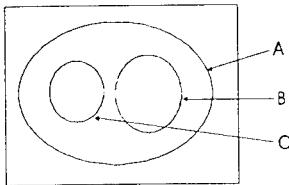
Figure 2 shows a particular example where: $B \subset A$, $C \subset A$, $B, B \cap C = \varnothing$. That is, B and C are mutually exclusive.

We use the following definitions:

Let X be a class in the MCRDR framework. $\{X_0...X_m\}$ is a set of rules which have class X as the conclusion. $\{X_{i0} ... X_{in}\}$ is a set of conditions for rule X_i where $i = 0...n$, n is the number of distinct conditions in the rule path; m is number of rule paths for class X. In the MCRDR framework the class is given as a disjunction of rule paths (Richards and Compton 1997). Then:

$$\text{class } X = \overset{m}{\underset{i=0}{\vee}} (\overset{n}{\underset{j=0}{\wedge}} X_{ij})$$

That is, X_{ij} stands for an individual condition in a rule path for the class X.

We further define a quality measure Q for a rule. This measure was introduced to counteract the significance of rules which are a gross overgeneralisation so that after corrections are added few conclusions are provided by the original rule.

$Q(X_i)$ = number of cases with conclusions given by X_i / (number of cases given by descendants of X_i + number of cases with conclusions given by X_i).

If we have no information about the number of cases, the default for Q is 1.

If the quality of a rule is close to 100%, it means that nearly all cases reaching this rule are processed by the the rule with few cases being processed by child rules. On the other hand if the quality of a rule is 10%, it means 90% of the

cases that satisfy the rule are passed to its children. That is the rule is too general and can be regarded as not being a particularly good rule and so it should not be given as strong consideration in developing the relations in the system.

Similarity

If X is a class and Y is also a class, we could define a similarity measure as follows:

$\text{Sim}(X_{ij}, Y_{ij}) = 0$ if X_{ij}, Y_{ij} are different

$\text{Sim}(X_{ij}, Y_{ij}) = 1$ if X_{ij}, Y_{ij} are same

If α is set of distinct attributes in rule path X_i, β is set of distinct attributes in rule path Y_i, then we can define:

$$\text{Similar}(X_i, Y_i) = \frac{\sum\limits_{j=0}^{n} \text{Sim}(X_{ij}, Y_{ij})}{|\alpha \cup \beta|} * Q(X_i) * Q(Y_i)$$

For example, from figure 1 (and assuming Q is 1):

Similar(rulepath-2, rulepath-6) = 1/3, Similar(rulepath-8, rulepath-9) = 1/2, Similar(rulepath-9, rulepath-10)= 2/3. Function Similar() measures a similarity between 2 node (each node contains a rule path).

Figure 3. Similarity of 2 nodes.
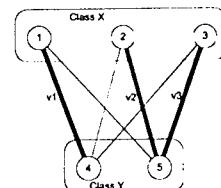
Figure 3 suggests how from rule similarities we can find a similarity measure between 2 classes. It shows that Class X is the disjunction of nodes 1,2 and 3 and Class Y is the disjunction of nodes 4 and 5. We propose that ClassSimilarity(X,Y) = (v1 + v2 + v3) / 3 ,where we choose the v such that all nodes are covered by at least one edge and the sum of v (eg. v1 + v2 + v3) is maximal. Note that v stands for the Similar() function. In later similar diagrams v stands for the Subsume() and MutualEx() measures. Eg. ClassSimilarity(Go swimming, Play chess) = ((1/3) + (1/2) + (2/3)) / 3 = 0.5.

Subsumption

We can define a subsumption measure as follows with X_{ij} and Y_{ij} standing for individual conditions in rule paths for the relevant classes as above.

$\text{Sub}(X_{ij}, Y_{ij}) = 0$ if X_{ij} does not subsume Y_{ij}
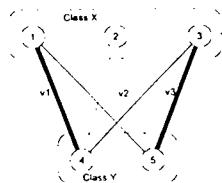
Sub (X_{ij}, Y_{ij}) = 1 if X_{ij} subsumes or same Y_{ij} (for example A>5 subsumes A>10)

If α is set of distinct attributes in rule X_i, β is set of distinct attributes in rule Y_i, then we can define:

$$Subsume(X_i, Y_i) = \frac{\sum_{j=0}^{n} Sub(X_{ij}, Y_{ij})}{|\alpha \cup \beta|} * Q(X_i) * Q(Y_i)$$

Function Subsume () measures a degree of confidence that the first rule path subsumes the second rule path.

Figure 4. Subsumption of 2 nodes

For example subsume (rulepath-2,rulepath-4) = 2/2, (rulepath-2 sky=SUNNY, rulepath-4 sky=SUNNY, wave=LOW), $|\alpha \cup \beta|$ = 2, that is {sky, wave}. Since rulepath-2 does not have an attribute wave, we consider rulepath-2 is more general than rulepath-4 with an attribute wave. Therefore there are 2 conditions in rulepath-2 which are same or more general than rulepath-4. Similarly, subsume (rulepath-2, rulepath-5) =2/3.

Figure 4 illustrates how from rule subsumption we can find a subsumption measure between 2 classes. It shows Class X as a disjunction of nodes 1,2 and 3 and Class Y as a disjunction of nodes 4, 5 (each node contains a rule path).

We compute ClassSubsume(X,Y) = (v1 + v3) / 2. We choose the v such that all nodes of class Y are covered by at least one edge and sum of v (eg. v1 + v3) is maximal. Eg. classSubsume (Go swimming, Swimming in the beach) = (1+0.667)/2. We then compute TotalSubsume (X,Y) = ClassSubsume (X,Y)-ClassSubsume (Y,X). If the value of TotalSubsume (X,Y) is negative, then we exchange X and Y, so we can convert the value to positive. Since classSubsume (Swimming in the beach, Go swimming) = 0, Total-Subsume(Go Swimming, Swimming in the beach) = 0.83.

Mutual Exclusivity
We can define a mutually exclusive measure as follows with X_{ij} and Y_{ij} standing for individual conditions in rule paths for the relevant classes as above.
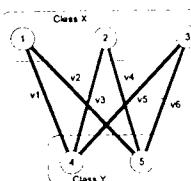
Mut (X_{ij}, Y_{ij}) = 0 if X_{ij} and Y_{ij} are not mutually exclusive.

Mut (X_{ij}, Y_{ij}) = 1 if X_{ij} and Y_{ij} are mutually exclusive (for example A>5 and A<2).

MutualEx(X_i, Y_i) = 1 , if at least one of Mut(X_{ij}, Y_{ij})=1

MutualEx(X_i, Y_i) = 0 , otherwise

Figure 5. MutualExclusivity of 2 nodes

Function MutualEx() measures a degree of confidence that the first rule subsume the second rule. For example

MutualEx(rulepath-2, rulepath-10) = 1.0,

Figure 5 illustrate how from rule mutual-exclusivity can find a mutual exclusivity measure between 2 classes. It shows that Class X is a disjunction of nodes 1,2 and 3 and Class Y is a disjunction of nodes 4 and 5. We compute ClassMutualEx(X,Y) = (v1 + v2 +v3 +v4 + v5 + v6) / 6. X and Y are mutually exclusive iff all nodes of X and Y are mutually exclusive with respect to each other (see Figure 5). Therefore ClassMutualEx (Go swimming, Play chess) = 1/6, since Go Swimming has 2 rulepaths, Play chess has 3 rulepaths and all MutualEx() between those rulepaths are 0.0, except for MutualEx(rulepath-2, rulepath-10) = 1.0.

EXPERIMENTAL RESULTS

Class relations model
Results from an endocrine knowledge base are shown in the table 1. The results shown in each table are the class pairs with the highest similarity, subsumption, or mutual exclusivity measures. Only results with high values for these relations are shown.

Extracting patterns from the class relation graph
Since there are many classes (from 25 to 100), it is impossible to consider all possible pairs of relations between the classes.

We therefore extract specific patterns which seem likely to be components of a meaningful taxonomy. For example:

Subsume(A,B)=1.0,
Subsume(A,C)=1.0,
MutualExclusive(B,C)>0.5

for all sets of three classes A,B,C from a knowledge bases

175

We can then combine such elements. E.g we may join Triangle(A,B,C) and Triangle(D,E,F) if A=D and MutualExclusive({B,C}, {E,F}, > 0.5). If MutualExclusive(B, C, == 1.0), we could say {B,C} are exhaustive subclass partitions of A [23].

We applied this technique to the thyroid knowledge base and obtained the result in figure-6.

Table 1. some examples of class relations.

Hormone knowledge bases system similarity-value > 0.66	
Class description	Class description
5. Satisfactory prolactin level.	12. Satisfatory prolactin level.
14. Consistent with premature ovarian failure.	28 Consistent with premature ovarian failure. Suggest repeat FSH and oestradiol in 2-3 months to confirm.

Hormone knowledge bases system subsumption-value = 1	
Class description	Class description
6. Elevated prolactin persists. Suggest TSH.	33. Elevated prolactin persists. Primary hypothyroidism has been excluded. IV sampling through an in-dwelling cannula can help exclude stress-related elevations of prolactin. Pituitary imaging may be required.
	36 Elevated prolactin persists. Await TSH.
7. Raised prolactin in females is commonly due to medication, strees or lactation. Suggest TSH to exclude hypothyroidism and repeat prolactin after 30 minutes rest.	23. Raised prolactin in men is commonly due to stress, medications and occasionally hypothyroidism. Suggest TSH and repeat prolactin after 30 minutes rest.
	30. Raised prolactin in females - is commonly due to medication, stress or lactation. Hypothyroidism has been excluded. Suggest review medications and repeat prolactin after 30 minutes rest.

Hormone knowledge bases system mutual-exclusivity-value=1	
Class description	Class description
4. Consistent with perimenopause	9. No evidence of perimenopause, unless patient is on oestrogen therapy.
5. Satisfactory prolactin level.	5 Elevated prolactin persists. Suggest TSH

DISCUSSION
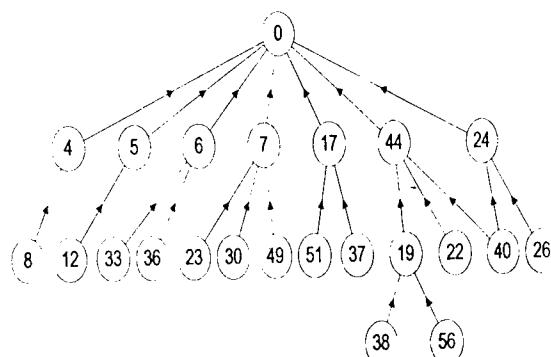
It is beyond the scope of this paper (and the authors) to provide a detailed medical analysis of the ontologies produced by these techniques; however, it is worth noting some lay observations.

The mutually-exclusive classes in Table 1 seem reasonable. However, we also have some cases where very similar conclusions are identified as mutually exclusive. This occurs when experts make up rules that include different values for the same attribute. For example, some mutually exclusive conclusions seem to give the same clinical advice but specifically refer to a patient being male or female. This may or may not be of clinical importance, but there is an obvious opportunity to have a further superclass.

The most interesting issues arise with the nature of subsumption. A superclass subclass relation may arise where one rule specifies a value for an attribute and another does not. For example a key factor in comments 6, 7, 23, 30, 33 and 36 is whether or not a TSH measurement should be ordered and whether or not primary hypothyroidism has been excluded. TSH results are important in the diagnosis of primary hypothyroidism. The generic comment suggesting a TSH measurement is given when there is no TSH result available. The comment also suggests the clinical cause of the high prolactin level remains unknown. When there is a TSH result available this provides some evidence to confirm or exclude one of the causes of the high prolactin. These relations appears to us as ontologically reasonable. However, the wording of the actual comments does not readily indicate such relationships. It would be interesting to know how the expert would react and how comment might be worded if this ontological information was available as rules were being developed.

A more general example of this pattern is the comment [0]:"patient has ovulated" which is at the top level of th taxonomy in Figure 6. This subsumes a whole range o more specific comments related to other attributes. Again would be interesting to see if this taxonomic informatio influenced the expert's wording.

Figure 6. Partial taxonomy of the thyroid knowledge base

FURTHER WORK

The results above seem promising. However, we should note that in other knowledge bases we have examined, the ontological relations seem much more idiosyncratic. It was expected that the free-text interpretations might combine classes and other complexities. However, some seem to be best described as part of a conversation. For example in a comment suggesting a specific follow-up procedure the pathologist may note that this procedure has been suggested previously. The present methods we are using are simply heuristics that seem appropriate to a classification system. To deal with more idiosyncratic relationships, we will need to develop further heuristics. We anticipate that these will include ways of clustering similar classes and then looking at the relationships between the clusters, as well as within.

Finally, the present technique considers only the conditions in rule paths. It does not consider any other information about the classes themselves. The refinement structure for RDR does not assume any ontological refinement; the expert is simply indicating that a conclusion is inappropriate and so should be replaced by another. It may be possible in some domains to provide some constraints on how refinements are added so that the ontological relationships that emerge are more well defined.

CONCLUSION

The work we have presented is an attempt to develop techniques to discover the ontologies implicit in knowledge bases. We believe it will be of increasing importance to carry out this particular kind of knowledge discovery as larger and larger knowledge bases come into use and we seek to exploit the knowledge in the knowledge bases in different ways. We do not make any particular claim for the techniques we have developed to date, except that they suggest that such ontology discovery is possible. The key idea in the techniques we have developed is that is seems reasonable to use heuristic quantitative measures to group classes and class relations. This then enables possible ontologies to be explored on a reasonable scale

Acknowledgement

REFERENCES

[1] Agrawal, R., H. Mannila, et al. (1996). *Fast Discovery of Association Rules*, AAAI/MIT Press chapter 12, 307-328.

[2] Beydoun, G. and A. Hoffmann (1997). *NRDR for the Acquisition of Search Knowledge*. 10th Australian Conference on Artificial Intelligence. 1-16.

[3] Catlett, J. (1992). *Ripple Down Rules as a Mediating Representation in Interactive Induction*. Proceedings

of the Second Japanese Knowledge Acquisition for Knowledge Based Systems Workshop, Kobe, Japan. 155-170

[4] Compton, P. and R. Jansen. (1990). *A philosophical basis for knowledge acquisition*. Knowledge Acquisition 2: 241-257.

[5] Compton, P., P. Preston, et al. (1995). *The Use of Simulated Experts in Evaluating Knowledge Acquisition*. 9th AAAI-sponsored Banff Knowledge Acquisition for Knowledge Base System Workshop, Canada. 12.1-12.18

[6] Compton, P. and D. Richards (2000). *Generalising Ripple-Down Rules*. Knowledge Engineering and Knowledge Management (12th International Conference EKAW2000). Springer, Berlin. 380-386.

[7] Edwards, G., P. Compton, R. Malor, A. Srinivasan, and L. Lazarus (1993) *PEIRS: a pathologist maintained expert system for the interpretation of chemical pathology reports* Pathology 25: 27—34.

[8] Flax, Lee. *Application of hamming distance between logical formulae to statistical sontingency tables*. Technical Report, Computing Departement, Macquarie University, Australia.

[9] Gaines, B. R. (1989). *Ounce of Knowledge is Worth a Ton of Data: Quantitative Studies of the Trade-off between Expertise and Data based on Statistically Well-founded Empirical Induction*, Proceeding of the sixth International Workshop on Machine Learning. San Mateo, California: Morgan Kaufmann, 1989. 156-159

[10] Gaines, B. R. (1995). *Transforming Rules and Trees into Comprehensible Knowledge Structures*, Advance in Knowledge discovery and Data Mining. AAAI/MIT Press. 205-226

[11] Gaines, B. R. and P. J. Compton (1992). *Induction of Ripple Down Rules*. Fifth Australian Conference on Artificial Intelligence, World Scientific, Singapore. Hobart. 349-354

[12] Hamming, R. W. *Coding and infromation theory*. Prentice Hall 1986.

[13] Kang, B., P. Compton, et al. (1998). *Simulated Expert Evaluation of Multiple Classification Ripple Down Rules*. 11th Banff knowledge acquisition for knowledge-based systems workshop, Banff, SRDG Publications, University of Calgary. 1-19

[14] Kang, B., Yoshida, et al. (1997). *A help desk system with intelligent interface*. Applied Artificial Intelligence 11((7-8)): 611-631.

[15] Kang, B. H., W. Gambetta, et al. (1996). *Verification and Validation with Ripple Down rules*. International Journal of Human-Computer Studies 44: 257-269.

[16] Kivinen, J., H. Mannila, et al. (1993). *Learning Rules with Local Exceptions*. European Conference on Computational Theory.

[17] Perez, A. G. (1999). *Evaluation of Taxonomic Knowledge in Ontologies and Knowledge Bases*. KAW'99, Twelfth Workshop on Knowledge Acquisition, Modeling and Management, Voyager Inn, Banff, Alberta, Canada. 6.1.1 - 6.1.18

[18] Ramadan, Z., P. Compton, et al. (1998). *From Multiple Classification RDR to Configuration RDR*. 11th Banff Knowledge Acquisition for Knowledge Base System Workshop, Canada. 1-19

[19] Richards, D. and P. Compton (1997). *Uncovering the conceptual models in RDR KBS*. International Conference on Conceptual Structures ICCS'97, Seattle, Springer Verlag. 198-212

[20] Richards, D. (1998). *Ripple Down Rules with Formal Concept Analysis: A Comparison to Personal Construct Psychology*. Proceedings of KAW'98, Eleventh Workshop on Knowledge Acquisition, Modelling and Management, Banff, Alberta, Canada. 1-20

[21] Scheffer, T. (1996). *Algebraic foundations and improved methods of induction or ripple-down rules*. 2nd Pacific Rim Knowledge Acquisition Workshop. 279-292

[22] Schreiber, G., B. Wielinga, et al. (1993). *KADS A Principled Approach to Knowledge-Based System Development*, Academic Press.

[23] Siromoney, A. and R. Siromoney (1993). *Variations and Local Exception in Inductive Logic Programming*. Machine Intelligence - Applied Machine Intelligence. K. Furukawa, D. Michie and S. Muggleton. 14: 213 - 234.

[24] Suryanto, H., D. Richards, et al. (1999). *The automatic compression of Multiple Classification Ripple Down Rule Knowledged Based Systems: Preliminary Experiments*. Knowledge-based Intelligence Information Engineering Systems, Adelaide, South Australia, IEEE. 203-206

[25] Suryanto, H., Compton, P. (2000). Discovery of Class Relations in Exception Structured Knowledge Bases. International Conference on Conceptual Structures, ICCS 2000. 113-126

[26] William E. Grosso, H. E., Ray W Fergeson (1999). *Knowledge Modeling at the Millenium (The Design and Evolution of Protege-2000)*. Twelfth Workshop on Knowledge Acquisition, Modeling and Management, Voyager Inn, Banff, Alberta, Canada. 7.4.1 – 7.4.36

Learning Procedural Knowledge through Observation

Michael van Lent
Institute for Creative Technologies
University of Southern California
13274 Fiji Way
Marina del Rey, CA 90292
vanlent@ict.usc.edu

John E. Laird
Artificial Intelligence Laboratory
University of Michigan
1101 Beal Ave.
Ann Arbor, MI 48109
laird@umich.edu

ABSTRACT

The research presented here describes a framework that provides the necessary infrastructure to learn procedural knowledge from observation traces annotated with goal transition information. One instance of a learning-by-observation system, called KnoMic (Knowledge Mimic), is developed within this framework and evaluated in a complex domain. This evaluation demonstrates that learning by observation can acquire procedural knowledge and can acquire that knowledge more efficiently than standard knowledge acquisition.

Keywords

Machine learning, knowledge acquisition, rule learning, user modeling

INTRODUCTION

Software agents are being used to perform a wide range of tasks in a variety of applications such as military simulations [6], on-line training exercises [10] and computer games [11]. As with most intelligent systems, these agents require large amounts of knowledge to successfully perform their tasks. Acquiring procedural knowledge from experts and encoding it into a suitable representation is a costly process. In the TacAir-Soar project [10] more than ten person years of effort were required to acquire and encode the knowledge necessary for a medium fidelity agent. The research presented here explores the application of machine learning techniques to this problem of acquiring and encoding procedural knowledge.

Machine-learning approaches can be viewed as points along a continuum of expert and knowledge engineer effort vs. research effort (see Figure 1). Moving to the right on this continuum represents a long term improvement in efficiency, since the research needs only be done once while the experts and engineers must put in the requisite effort for each new task. At one end of the continuum lies the standard knowledge-acquisition approach [5]. This approach requires
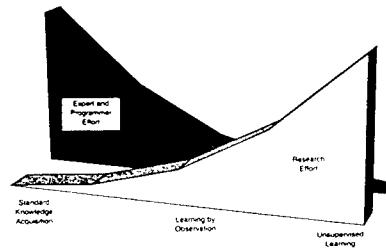
Figure 1: Expert effort vs research effort for a variety of knowledge acquisition approaches.

a great deal of effort from the expert to organize and teach the knowledge to the knowledge engineer who must then encode the knowledge. At the other end of the continuum lie a variety of unsupervised machine-learning techniques [17]. Unsupervised machine learning requires no effort from an expert but is an extremely challenging research problem for complex procedural knowledge. Moreover, these techniques do not necessarily result in behavior matching a human expert, an important consideration in training tasks and military simulations. Our research explores the middle of this expert effort vs. research effort continuum. Our hypothesis is that learning procedural knowledge from observations of an expert is more efficient than the standard knowledge-acquisition approach and is a more tractable research problem than the unsupervised learning approach. The observations of the expert's behavior consist of the sensor inputs seen by the expert, the expert's operator selections, and the actions performed to achieve these operators. Learning by observation does not require a knowledge engineer to encode the knowledge or the expert to organize and communicate knowledge to an engineer. Instead, the expert simply performs the task at which he or she is an expert [18].

The primary goal of this research is to explore and evaluate observation as a knowledge source for learning procedural knowledge. The first step is to develop a general framework for learning-by-observation systems that casts the issue of acquiring knowledge from observations as a machine-learning problem. Plugging different components (learn-

ing algorithms, execution architectures,...) into this framework results in different learning-by-observation systems. Based on this framework, we have developed a learning-by-observation system called KnoMic (Knowledge Mimic) that uses a simple inductive learning algorithm and then the Soar architecture to execute the learned knowledge. KnoMic is evaluated in a complex real-world domain to show that procedural knowledge can be learned from observation even with a simple learning algorithm. As the complexity of the domain increases, especially when observations contain a great deal of noise, more powerful learning algorithms can be substituted. However, the learning-by-observation framework will apply equally well to these complex domains and powerful learning algorithms.

RELATED WORK

The research described here expands on previous research into learning by observation such as behavioral cloning [1, 7, 15], OBSERVER [19]. and TRAIL [3]. KnoMic combines aspects of these three efforts and includes a number of novel contributions to significantly extend the capabilities of the learning-by-observation paradigm. There has also been some research in the robotics community looking at learning robot motor control by observation and imitation [4, 16]. However. the research is only peripherally related as the learned motor control knowledge isn't procedural and is usually non-symbolic.

Behavioral Cloning

Bain and Sammut [1, 15] used behavioral cloning, their term for learning by observation, to learn the knowledge necessary to fly a simulated airplane along a specific flight plan in the Silicon Graphics flight simulator. Behavioral cloning applies the C4.5 induction algorithm [14] to situation/action examples taken from observations of an expert. These observations are used to build decision trees that decide which actions to take based on current sensor input. These decision trees then mimic the expert's behavior and fly the plane by setting each control to the value specified by applying the current sensor inputs to the decision tree. One of the most impressive aspects of behavioral cloning is its effectiveness in a complex, non-deterministic, and dynamic environment.

Behavioral cloning manually segments the flight plan into seven stages and a separate decision tree is learned for each stage. Separate decision trees are needed because each stage represents a different step in the task, requiring a different procedure to perform that step. Therefore, the expert's responses to sensor inputs differ in the different stages. Behavioral cloning does not learn knowledge that allows the agent to dynamically select goals and procedures. The seven phases of the task are hard-coded into the interface between the agent and the environment. If the agent's task were changed even slightly the decision trees would have to be relearned from a new set of observations.

The OBSERVER system

The OBSERVER system [19] learns STRIPS-style operators [8] using a learning method similar to version spaces. Wang developed OBSERVER and applied it to a process-planning domain, in which the task is to generate a plan to produce machine parts meeting a set of specifications. This design task is less complex and does not share all of the same

properties as most of the procedural tasks we have examined such as dynamic, non-deterministic environments and durative actions. An advantage the OBSERVER system has over behavioral cloning is that it learns a more expressive knowledge format. Unlike behavioral cloning. OBSERVER's STRIPS-style operators are procedural. Each operator includes pre-conditions allowing the agent to dynamically select which operator (procedure) to execute based on the current sensor input. However. the STRIPS-style operators assume that operator actions are always performed in a single time step and without error. This is acceptable for OBSERVER's simulated design task but is likely to cause problems in more complex tasks where the agent's actions may fail or only gradually achieve the desired effect over multiple time steps. Since the design task contains no noise or dynamic changes. OBSERVER can and does generate knowledge based on a single observation.

The TRAIL system

Scott Benson's TRAIL system [3] combines a learning algorithm and a planning algorithm to create teleo-operators (TOPs) which are similar to STRIPS operators. The learning algorithm uses traces of both undirected exploration by the agent and directed exploration based on observation. The behavioral traces are annotated to indicate which TOP the expert is performing for each segment. From these traces TRAIL learns the pre-conditions and effects of the procedural teleo-operators. TRAIL uses inductive logic programming to learn TOPs based on positive and negative examples from the traces. Positive instances are steps where the actions achieved the TOP's post-conditions and negative instances include time steps in which the post-conditions were not achieved. As with OBSERVER, TRAIL's TOPs can have difficultly in complex domains with uncertain action outcomes and durative actions. However, TRAIL's segmentation of the observation traces according to the expert's goal selection helps focus the learning algorithm. The segmentation allows TRAIL to easily locate transitions between TOPs and identify the positive and negative instances of TOP selection.

KNOMIC

KnoMic is a learning-by-observation system based on a general framework for learning procedural knowledge from observations of an expert. The knowledge KnoMic learns is represented as a fairly general hierarchy of operators which can be formatted into a number of specific representations including production rules for intelligent architectures and decision trees. This section first describes KnoMic's general knowledge representation, followed by a high-level description of the learning-by-observation framework. Following this description, each component of the framework is described in more detail.

Knowledge Representation
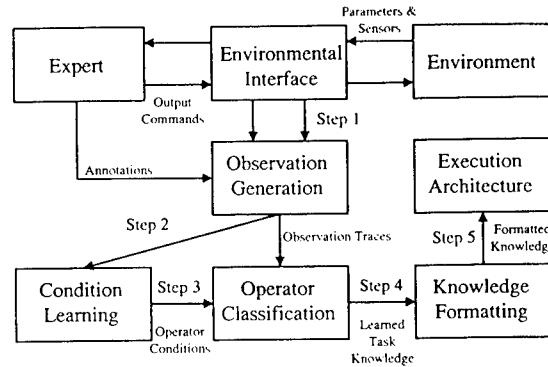
To perform complex, real-world tasks agents must constantly react to changes in complicated, dynamic environments by selecting appropriate goals and performing actions to achieve and maintain those goals. Frequently, procedural knowledge is represented by a collection of operators composed of pre-conditions and effects (as used in STRIPS [8]). KnoMic utilizes a modified, more robust version of this standard opera-

tor representation. Where STRIPS operators include effects, denoting direct changes to the current state, KnoMic operators include actions which are commands to be implemented in the environment by the environmental interface. Unlike STRIPS operators, the effects of KnoMic's commands are determined by the environment and may or may not have the expected outcome. In addition, KnoMic is designed to be effective in domains that include non-deterministic environments and other agents who are often unpredictable. For these multiple reasons, KnoMic's operators must be able to recover when the actions and environment don't behave as expected. This is handled by expanding the operator representation to include a set of goal conditions. A KnoMic operator will remain active while its pre-conditions are satisfied and until it achieves its goal conditions. Thus, if the operator's actions don't have the intended effect of achieving the goal conditions, the operator will remain active and alternate approaches to achieving the goal can be attempted. Since the operator's actions are conditional each operator can include multiple approaches to achieving the goal. Additionally, operators are arranged into an operator hierarchy with sub-operators only being candidates for activation when their parent operators are active. At any given time only one operator at each level of the hierarchy can be active. An operator can achieve its goal conditions through its actions or through the actions of sub-operators. Often sub-operators will represent either a sequence of steps that achieve the super-operator's goal or a number of alternate approaches to achieving the super-operator's goal. Thus, each of KnoMic's operators can have multiple procedures for achieving its goals, represented either as conditional actions or sub-operators. An example of a learned operator is shown in figures 7, 8, and 9.

A second aspect of KnoMic's knowledge representation is the classification of each operator as homeostatic, one-time, or repeatable. These classifications denote how the operator will behave after its goals conditions are achieved. A homeostatic operator will always attempt to maintain its goal conditions as true once they are achieved. Thus, if a homeostatic operator's goal conditions become untrue, the operator can be immediately re-activated to re-achieve the goal. A one-time operator, on the other hand, will only achieve its goal conditions once and then never be re-activated. A repeatable operator will not be immediately re-activated if its goal becomes unachieved but can be reselected at a later time if triggered by another operator. Our experience in applying the Soar architecture to a variety of complex domains has shown these three classes of operator behavior to be necessary in many cases [10, 11, 18].

Learning-by-Observation Framework

The learning-by-observation framework, shown in Figure 2, consists of modular components that work in concert to learn task knowledge from annotated observations. The arrows labeled with step numbers in Figure 2 do not represent a constant flow of information but rather a series of five discrete steps. A high-level description of each step is presented here with more detailed descriptions in the following subsections. In the first step, a number of observation traces are generated. The environmental interface sits between the expert and the environment and sends sensor and parameter information to the expert and returns the expert's commands to the environment. The observation

Figure 2: The learning-by-observation framework.

generation component gathers the sensor and parameter inputs, the expert's output commands, and the expert's goal-change annotations (which segment the traces in a fashion similar to TRAIL) and creates observation traces. The expert annotates the observation trace by specifying the points where he/she changed goals because a goal was achieved or abandoned. These goals correspond to the operator goals in the learned knowledge. The expert can work out a goal hierarchy for the task in advance to aid in specifying goal transitions. KnoMic can then learn operators that fit into this hierarchy. Although observation is the primary source of knowledge, the framework could also be viewed as a hybrid observation/automated knowledge acquisition system due to the annotations the expert adds. These annotations require the expert to consider how the task knowledge will be represented, a hallmark of automated knowledge acquisition. The future work section will discuss some approaches to eliminating the need for annotations in the observation trace.

Once a set of observation traces (typically 4-8 are required) are available, the second step is to learn the operator conditions using the condition learning algorithm. After the operator conditions have been learned, the operator classification component uses the conditions and the observation traces to classify each operator and determine the persistence of an internal feature representing that the operator's goal has been achieved. Once the classification step is complete, the fourth step takes the learned task knowledge and formats it in the representation required by the execution architecture. This step maintains the independence between the learning system and the specific execution architecture. Once the knowledge is in the correct representation, the execution architecture can replace the expert and perform the task by interacting through the environmental interface.

A major advantage of the learning-by-observation framework is its modularity. The observation generation, condition learning, and operator classification components all act relatively independently. Thus, it is possible to change learning algorithms, execution architectures, or domains without modifying the other components in the framework. Ob-

Figure 4: The hierarchical display used by the expert to annotate his behavior while performing the task.
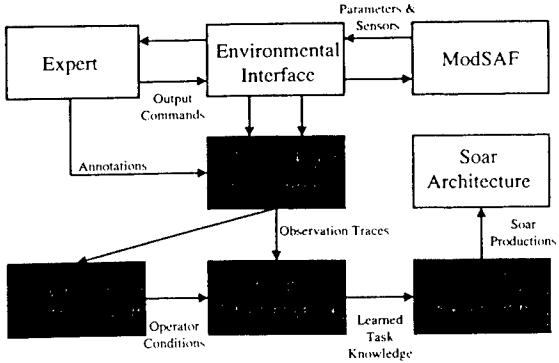
Figure 3: The KnoMic system, which is an instantiation of the learning-by-observation framework.

viously, the knowledge formatting component and the execution architecture are closely related but these two components can also be modified independently of the previous three. KnoMic is one instantiation of the learning-by-observation framework. As shown in Figure 3. KnoMic uses a simple specific-to-general inductive learning algorithm and generates productions for the Soar architecture from the learned procedural knowledge.

Observation Trace Generation

An expert's interaction with the environment while performing a task is a communication loop. The interface to the environment sends symbolic sensor and task parameter information to the expert and the expert reacts to this information by sending symbolic actions to the environmental interface. The interface causes these actions to be performed in the environment and the sensor information changes to reflect the effects of these actions. The observation generation component records the sensor inputs the expert receives and the actions the expert performs each cycle through the loop. In addition, the expert annotates the observation trace whenever the goal he/she is seeking to achieve changes - an approach similar to that used by the TRAIL system. The expert can add annotations as the task is being performed or during a review of his/her behavior to avoid interrupting the performance of the task. For the experiments described later the expert annotated the observation traces while performing the task by clicking on the active operators in a hierarchical display as shown in Figure 4. The problem of segmenting observation traces into chunks corresponding to individual procedures is faced by many systems and an area of active research [2].

Specific-to-General Condition Learning

The condition learning component takes a list of observation traces and incrementally learns the operator pre-conditions, action conditions, and goal conditions. As the focus of the research here is the development of the framework, not the
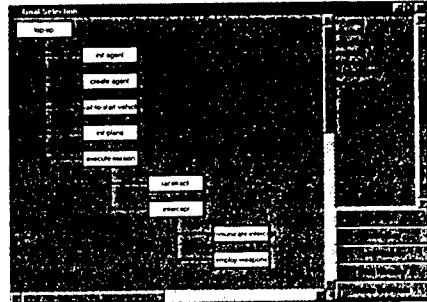
development of new learning algorithms, KnoMic is based on a simple, efficient and well-known learning algorithm. The learning algorithm is a slightly improved version of the Find-S specific-to-general induction algorithm [12]. Although this algorithm is fairly successful, it is doubtful it will work for all domains. However, due to the modularity of the learning-by-observation framework, it can easily be replaced with more powerful learning algorithms without requiring major changes to the rest of the system. The observation trace is examined cycle by cycle to extract the instances used to learn the conditions. Each step in the observation trace that includes an operator change annotation is used as a positive instance of that operator's pre-conditions and also a positive instance of the previous operator's goal conditions. A positive instance includes all the current sensor inputs as well as any relations ($\leq, \geq$, and $=$) between sensors or between sensors and task parameters that recently (as defined by a parameter to the learning system) became true. This bias towards recent sensor/parameter relations seeks to reduce the vast number of relations that would otherwise have to be considered. In addition, each sensor and parameter is classified according to the unit (feet, seconds...) it is represented in and relations are only allowed between inputs represented in the same type of units. The Find-S learning algorithm treats the first positive instance as an initial, most-specific hypothesis and generalizes that hypothesis to cover each additional positive instance. If generalizing to cover a new instance results in an empty set of pre-conditions, a second, disjunctive set of operator pre-conditions is created. Each new positive instance is then applied to the set of conditions that requires the least generalization to cover it. Once all the observation traces have been processed the result is one or more sets of pre-conditions (disjunctive pre-conditions if more than one set) for each operator the expert was observed to perform. Goal conditions are learned in a similar fashion. The only difference is that only the sensor inputs that changed recently are included in the goal condition positive instances. Since goal conditions represent changes made by the operator, they should only test for changes in the environment due to the recent actions of the operator. Finally, action conditions and values are learned using each time step in which the expert performs an action as a positive instance of that action's conditions and values.

182

Operator Classi£cation

Once the conditions and actions for each operator are learned, KnoMic makes another pass through the observation trace to classify each operator as homeostatic, one-time or repeatable. Operators are classified by examining if and when the expert reselects an operator as its goal conditions change from achieved to unachieved. If the expert reselects the operator each time the goal conditions are no longer true the operator is homeostatic. If the operator is not reselected immediately, it is repeatable and if the operator is never reselected it is a one-time operator. This classification determines whether each operator's internal goal-achieved feature should be persistent or non-persistent. Persistent goal-achieved features allow the agent to recall that past goals have been achieved, and should not be pursued again, even if the goal conditions of those operators are no longer satisfied. Non-persistent goal-achieved features are useful if a set of goal features should be achieved and maintained. Homeostatic operators will have non-persistent goal features while one-time and repeatable operators have persistent goal features. In the case of repeatable operators, operators that can trigger re-activation are given new actions to remove the repeatable operator's goal-achievement feature.

Soar Production Generation

The task-performance knowledge that KnoMic learns is independent of any specific execution architecture although it is inspired by the Soar architecture. It is a hierarchical, symbolic, propositional representation allowing some numerical relations. Before the knowledge can be used, it must be translated by the knowledge generation component into a representation appropriate for an available architecture. Production generation takes the learned knowledge and creates three classes of Soar productions: operator precondition productions, goal-achieved productions and action application productions. Although KnoMic currently only generates productions formatted for the Soar architecture, it would be possible to generate knowledge for other similar architectures.

EXPERIMENTS AND RESULTS

This section presents two experiments showing that all the elements of the procedural knowledge representation described previously can be learned and reporting on the accuracy of the learned knowledge. The first experiment explores the accuracy of the KnoMic system when provided with error-free observation traces. The second experiment explores KnoMic's accuracy with more realistic observation traces generated from observations of a human expert. Previous research has compared the efficiency of learning by observation to standard knowledge acquisition showing that learning by observation is almost four times faster than the estimated typical knowledge engineer [18]. The domain used in these experiments is the air combat domain, although similar experiments have also been run in a second domain using the commercial computer game Quake II.

Air Combat Domain

The air combat domain is a complex domain that includes properties of particular interest, such as a highly dynamic environment and the need for agents that behave in a realistic, human-like fashion. In the air combat domain, the
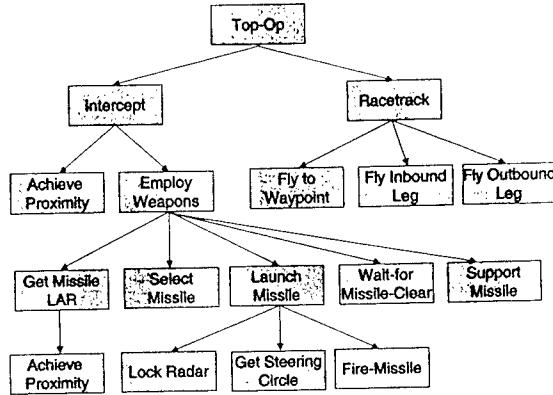

Figure 5: The operator hierarchy (except initialization operators) learned by KnoMic in the air combat domain.

expert, or an intelligent agent, controls a military aircraft in the ModSAF battlefield simulator [6]. The specific task studied here involves taking off, flying to a specified waypoint, and flying a patrol pattern called a racetrack. If an enemy plane is detected and satisfies the commit criteria parameters, the expert or agent must intercept the enemy plane, shoot it down, and return to flying the racetrack. The environmental interface provides the expert with 54 sensor inputs, 23 task parameters and 22 output commands. On a standard workstation (300 MHz Pentium II), ModSAF updates the sensor inputs and accepts output commands 8-10 times a second. The time granularity of actions varies from seconds (when turning during the patrol) to tenths of a second (when firing missiles). However, for the learning system to be effective, the expert must perform actions within half a second of the triggering sensor changes. The knowledge required for the air combat domain consists of 31 operators in a four level hierarchy. The operator hierarchy, shown in Figure 5, includes ten initialization operators, one takeoff operator, one execute mission operator, four operators used to perform the racetrack, and twelve operators for the intercept.

The air combat domain is well suited to learning by observation for a number of reasons. First, the operator conditions are, for the most part, conjunctive and when disjunctive there isn't any overlap between components of the disjunction. The current learning algorithm would not be effective in a domain that included partially overlapping disjunctive condition sets. Second, the operator transitions and actions required by the domain are triggered by external events sensed by the expert. Tasks involving extensive internal planning or time delays between trigger and action are difficult since these aspects of the task aren't apparent to the observation system. Third, the task doesn't involve many negated conditions. The simple specific-to-general learning algorithm KnoMic currently doesn't make use of negative instances and as a result is only able to learned negated conditions in a very limited sense. Of these three limita-
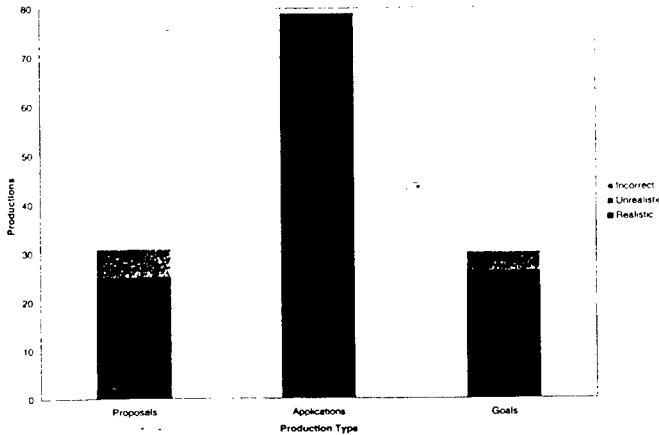
Figure 6: The number of correct (realistic), functionally correct (unrealistic) and incorrect productions after four observation of a software expert.

tions, only the first is an inherent limitation of the learning-by-observation approach. The other two limitations can be overcome with more sophisticated learning algorithms.

Evaluation Criteria

The procedural knowledge learned by KnoMic is checked for correctness in two different ways. First, using the learned knowledge, the agent performs the task in the same situations used to generate the observation traces. Due to the non-determinism of the environment the agent won't encounter the exact observed situations but to some extent this constitutes a test of the learned knowledge on the training data. Second, the actual learned rules are compared to rules for the same task created by a human programmer using the standard knowledge acquisition approach. This second evaluation criteria serves two purposes. The direct comparison with hand-coded, verified knowledge is a rigorous test of the learned knowledge that will discover errors that aren't apparent in the previous test. This direct comparison also highlights the fact that the knowledge learned by KnoMic is easily understood and modified by a knowledge engineer. This is important as learning by observation is unlikely to ever provide fully correct knowledge and some debugging will always be necessary. The two tests classify each learned rule as fully correct, functionally correct, or incorrect. Functionally correct knowledge doesn't cause errors in the observed situations (first test) but doesn't match the hand-coded knowledge (second test) and therefore may cause errors in novel situations.

Experiment 1

The first experiment evaluates the correctness of the knowledge learned by KnoMic from the error-free observation traces provided by an expert system hand-coded to perform the air combat task. The expert system is able to generate perfect observation traces, in that the timing and content of all output commands and operator annotations are correct. Variations between traces include differences in altitudes, speeds, relative angles, and similar factors due to variations in starting conditions and non-determinism in the environ-

ment. This experiment is an expanded version of a previously reported experiment [18]. The results here include an expanded task requiring more task knowledge (31 operators rather than 22 previously) and improvements to the learning algorithm to support disjunctive pre-conditions and goals conditions as well as sensor/sensor relation conditions. Four observation traces were generated and used by KnoMic to learn task knowledge. The task takes about 30 minutes for the expert system to perform. Each observation trace includes approximately 19,000 decision cycles. There are approximately 25,000 sensor input changes recorded, 40 goal annotations and 140 output commands issued. From the four observation traces, KnoMic successfully learns the 3 operators in the four level hierarchy. Encoded as Soar productions, the learned task knowledge consists of 140 productions. Of these 140 productions, 101 are fully correct and an additional 29 are functionally correct (see Figure 6) The productions that are incorrect fall into three categories. Six of the 10 incorrect productions have extraneous conditions representing over-specialized condition sets. Three of the remaining incorrect rules are due to missing sensor in the environmental interface. The final incorrect production requires a negated test for a sensor input that KnoMic is unable to learn from only positive instances. As shown in Figure 6, 23 of the 29 functionally correct production are operator proposal productions. All of these production vary from the hand-coded productions due to extraneous conditions testing either internal goal-achieved features or external sensors. Figure 7 is an example of an learned operator proposal production with extraneous conditions (figure 8 and figure 9 show the rest of the productions learned for the fly-to-racetrack operator). In this case the extraneous conditions include both internal goal-achieved feature and external sensors (such as the radar condition). These extraneous conditions don't cause problems in the observed situations (which is why they were not generalized away) but may cause errors in unobserved situations. Because the recent-changes heuristic is not used when learning operator pre-conditions, the space of potential pre-conditions is much larger. The larger space of pre-conditions includes sensor that change infrequently and therefore are likely to have the same values or relationships with parameters each time a operator is selected. This makes it easier for extraneous conditions to remain in operator pre-conditions than action or goal conditions. It is possible that more observation trace perhaps with greater variations in behavior, could eliminate some of these extraneous conditions.

Experiment 2

The second experiment evaluates the correctness of the knowledge learned by KnoMic from observation traces generated by observing a human expert. Two observation traces were generated from observations of a human expert performing the initialization, takeoff and racetrack part of the task. Because it is difficult to successfully perform the intercept portion of the task consistently, this experiment will focus on subset of the full task. These traces included greater variations in the factors mentioned above as well as variation in the timing of the actions and annotations due to the human's limited reaction time. These traces do not include any outright errors in task performance or annotations. Traces generated from observations of human experts include more variability in the task performance than the expert system

184

```
sp {propose*fly-to-racetrack
(state <ts> ^superstate nil
^goal-features <gf>)
(state <s> ^superstate <ss>)
(<ss> ^operator.name racetrack)
(<ts> ^observe.station-4 loaded)
(<ts> ^observe.cannon loaded)
(<ts> ^observe.station-1 loaded)
(<ts> ^observe.station-3 loaded)
(<ts> ^observe.station-5 loaded)
(<ts> ^observe.station-9 loaded)
(<ts> ^observe.station-11 loaded)
(<ts> ^observe.initialized *yes*)
(<ts> ^observe.io.input.vehicle.radar tws-auto)
(<gf> ^init-agent.goal achieved)
(<gf1> ^station-4.goal achieved)
(<gf> ^init-agent <gf1>)
(<gf2> ^cannon.goal achieved)
(<gf> ^init-agent <gf2>)
(<gf3> ^station-1.goal achieved)
(<gf> ^init-agent <gf3>)
(<gf4> ^station-5.goal achieved)
(<gf> ^init-agent <gf4>)
(<gf5> ^station-9.goal achieved)
(<gf> ^init-agent <gf5>)
(<gf6> ^station-11.goal achieved)
(<gf> ^init-agent <gf6>)
(<gf> ^wait-to-start-vehicle.goal achieved)
(<gf> ^init-plane.goal achieved)
-(<gf10> ^fly-to-racetrack.goal achieved)
(<gf11> ^racetrack <gf10>)
(<gf> ^execute-mission <gf11>)
-->
(<s> ^operator <o>)
(<o> ^name fly-to-racetrack)
}
```

Figure 7: The operator proposal production learned for the fly-to-racetrack operator. This proposal includes extraneous internal goal-achieved features and external sensor features.

```
sp {fly-to-racetrack*apply*desired-turn-rate
(state <ts> ^superstate nil)
(state <s> ^operator.name fly-to-racetrack)
(<ts> ^io.output-link <out>)
-->
(<out> ^desired-turn-rate moderately-hard)
}

sp {fly-to-racetrack*apply*desired-heading
(state <ts> ^superstate nil)
(state <s> ^operator.name fly-to-racetrack)
(<ts> ^io.output-link <out>)
(<ts> ^input.waypoint-comp.bearing.round <val>)
-->
(<out> ^desired-heading <val>)
}
```

Figure 8: Two Soar productions that issue actions associated with the fly-to-racetrack operator.

```
sp {goal*achieved*fly-to-racetrack*persistent
(state <ts> ^superstate nil
^goal-features <gf>)
(state <s> ^operator.name fly-to-racetrack)
(<ts> ^observe.param.waypoint-range <id0>)
-(<ts> ^input.waypoint-comp.range.rnd { > <id0> })
(<gf1> ^racetrack <gf0>)
(<gf> ^execute-mission <gf1>)
(<gf0> ^fly-to-racetrack <goal>)
-->
(<goal> ^goal achieved)
}
```
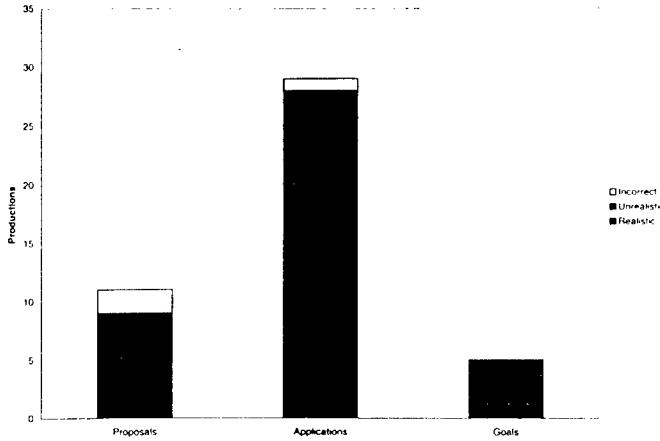
Figure 9: The goal-achieved production for the fly-to-racetrack operator.

generated traces. In some cases this extra variability aids KnoMic in generalizing away extraneous conditions but it also makes KnoMic's job more difficult as the positive instances taken from human observations aren't always consistent. The human expert controls an aircraft in the ModSAF simulator using an instrument panel that gets sensor input from ModSAF and passes the human's commands to Mod-SAF. The human expert also uses a graphical goal selection tool (shown in Figure 4) to aid in generating the operator annotations in the observation trace. Of the 45 productions learned by KnoMic for this subset of the task, 29 are correct, 13 are functionally correct and 3 are incorrect. As shown in Figure 10, four of the 13 unrealistic productions are operator proposal productions while 5 are applications and 4 are goal productions. In some cases. mostly involving pre-conditions, the extra variability inherent in the human behavior helped KnoMic eliminate unnecessary conditions. However, in other cases the variability in the timing of the annotations and actions caused errors that did not occur while learning from the expert system. This experiment shows that KnoMic can successfully learn from observations of human experts but additional research needs to explore a more robust learning algorithm.

FUTURE WORK

The next clear step in this research is to study the impact of more complex learning algorithms on the learning-by-observation framework. Some work has begun exploring the use of some inductive logic programming systems in the framework. A better learning algorithm, perhaps in conjunction with more powerful biases, should aid in removing the additional extraneous conditions that caused most of the errors in the first experiment and will be more robust to the noise and variability in the human generated observations. This research is also seeking to expand the learned knowledge representation to include structured sensors, operator parameters, and the incorporation of prior, hand-coded knowledge.

The process of annotating the observation traces with operator transition information is tedious. A technique for automatically segmenting the observation traces based on changes in the expert's behavior could make this process much easier. One possible approach is to have the expert annotate a subset of the behavior traces then use the initial operator conditions learned from these initial annotations

Figure 10: The number of correct (realistic), functionally correct (unrealistic), and incorrect productions after learning from two observations of a human expert.

to detect when operators would be selected and terminated. Based on these preliminary operators, the rest of the traces could be automatically annotated. It might also be possible to detect shifts in behavior through statistical analysis of the behavior traces. It would also be interesting to include separate annotations for "goal-complete" and "goal-abandoned." Currently these two cases are not distinguished which can result in unnecessarily complex goal conditions.

As stated previously, it is unlikely that learning by observation will ever result in perfect knowledge. A number of systems exist that take partially correct knowledge and seek to verify and correct that knowledge [9, 13]. We are beginning to look at these systems with the goal of applying them to procedural knowledge with the aid of the learning-by-observation concepts.

CONCLUSION

The research presented here focuses on the application of a relatively simple learning algorithm to a real world problem. Much of the recent research in the field of machine learning has focused on incremental improvements to state of the art learning algorithms that are tested on benchmark data sets. Surprisingly, there is not a great deal of research on the infrastructure necessary to move the application of these learning algorithms beyond the benchmarks. As demonstrated here, even a simple learning algorithm can be effective when embedded in a carefully designed framework. Hopefully, future research will find that more sophisticated learning algorithms are even more effective.

REFERENCES

[1] M. Bain and C. Sammut. A framework for behavioral cloning. In S. Muggleton, K. Furakawa, and D. Michie. editors, *Machine Intelligence 14*. Oxford University Press, 1995.

[2] M. Bauer and C. Rich. Learning how to do things: Papers from the 2000 aaai fall symposium. Technical Report Technical Report FS-00-02, AAAI, 2000.

[3] S. S. Benson. *Learning Action Models for Reactive Autonomous Agents*. PhD thesis, Computer Science Dept., Stanford University, 1996.

[4] A. Billard and M. J. Mataric. A biologically inspired robotic model for learning by imitation. In *Proceedings of the Fourth International Conference on Autonomous Agents*, pages 373–380, Barcelona, Catalonia, Spain, 2000. ACM Press.

[5] B. G. Buchanan and D. C. Wilkins, editors. *Readings in Knowledge Acquisition and Learning*. Morgan Kaufman, 1993.

[6] R. Calder, J. Smith, A. Courtemanche, J. Mar, and A. Ceranowicz. Modsaf behavior simulation and control. In *Proceedings of the Third Conference on Computer-Generated Forces and Behavioral Representation*, 1993.

[7] R. Chambers and D. Michie. Boxes: An experiment on adaptive control. In E. Dale and D. Michie, editors, *Machine Intelligence 2*. pages 125–133. 1968.

[8] R. E. Fikes and N. J. Nilsson. Strips: a new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.

[9] Y. Gil and E. Melz. Explicit representations of problem-solving strategies to support knowledge acquisition. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 469–476, 1996.

[10] R. M. Jones, J. E. Laird, P. E. Nielsen, K. J. Coulter, P. Kenny, and F. V. Koss. Automated intelligent pilots for combat flight simulation. *AI Magazine*, 20:27–41, 1999.

[11] J. E. Laird and M. C. van Lent. Human-level ai's killer application: Interactive computer games. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence*. 2000.

[12] T. M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[13] D. J. Pearson and J. E. Laird. Towards incremental knowledge correction for agents in complex environments. In *Machine Intelligence*, volume 15. Oxford University Press, 1996.

[14] J. R. Quinlan and R. M. Cameron-Jones. Combining instance-based and model-based learning. In *Proceedings of the Tenth International Conference on Machine Learning*, pages 236–243, 1993.

[15] C. Sammut, S. Hurst. D. Kedzier, and D. Michie. Learning to fly. In *Proceedings of the Ninth International Conference on Machine Learning*, pages 385–393, 1992.

[16] S. Schaal. Is imitation learning the route to humanoid robots? *Trends in Cognitive Sciences*, 3:233–242, 1999.

[17] W. Shen and H. A. Simon. Rule creation and rule learning through environmental exploration. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 675–680, 1989.

[18] M. C. van Lent and J. E. Laird. Learning hierarchical performance knowledge by observation. In *Proceedings of the 1999 International Conference on Machine Learning*, pages 229–238, 1999.

[19] X. Wang. *Learning Planning Operators by Observation and Practice*. PhD thesis, Computer Science Dept., Carnegie Mellon University, 1996.

A Grammar-Driven Knowledge Acquisition Tool that incorporates Constraint Propagation

Simon White[*]
Accelrys Ltd.,
230/250 The Quorum,
Barnwell Road, Cambridge, CB5 8RE.
England, UK.
email swhite@accelrys.com

Derek Sleeman
Computing Science Department,
University of Aberdeen.
Old Aberdeen, AB24 3FX
Scotland, UK.
email dsleeman@csd.abdn.ac.uk

Abstract
To acquire knowledge that is fit for a specific purpose, it is very desirable to have a structured, declarative expression of the knowledge that is needed. This paper introduces a stand-alone knowledge acquisition tool, called COCKATOO (Constraint-Capable Knowledge Acquisition Tool), which uses constraint technology to specify the knowledge it requires. The language in which these specifications are given is based on the meta-language notation of context-free grammars. However, we also took the opportunity to build a tool that is both more flexible and powerful by augmenting context-free grammars with the expressiveness of constraints. COCKATOO was implemented using the SCREAMER+ declarative constraints package.

Keywords
Knowledge Acquisition, Formal Grammars, Constraints, Constraint-Augmented Grammars, SCREAMER+

INTRODUCTION
Previous work has addressed the problem of determining whether existing KBs (Knowledge Bases) can be used together with a selected problem-solver to satisfy a given problem-solving goal [15], [16]. We refer to this task as assessing the *fitness-for-purpose* of a KB. When the assessment identifies a mismatch between the given KBs and the problem-solver's expected KBs, we recognise two possible responses. Either the available KBs are totally inappropriate, in which case it is necessary to acquire them *ab initio*, or the existing KBs are close to being usable but need to be modified (possibly in a number of ways). This paper addresses the first of these actions; namely, to *acquire knowledge ab initio such that it meets the problem solver's requirements*. In current practice, a knowledge engineer uses a knowledge acquisition tool, or other elicitation method(s), to acquire the knowledge needed by a problem solver. Afterwards, the knowledge must

usually be transformed, because the output of the knowledge acquisition tool cannot be used directly as input to the problem solver. We call such transformations *post-acquisitional transformations*. In this paper, we introduce the COCKATOO knowledge acquisition tool, which aims to minimise the need for post-acquisitional transformations. Our tool is *generic*, in the sense that it is independent of task, problem solver, and domain. On the other hand, it is highly configurable, and can be configured to acquire knowledge suited to a particular purpose (i.e., a problem-solving role).

The paper is organised as follows. In the next section, 'Grammar-Driven Knowledge Elicitation', we argue the benefits of using a context-free grammar as the basis for the specification of knowledge to satisfy a problem-solving role. In the section on 'Constraint-Augmented Grammars', we highlight some of the limitations of a purely grammar-driven approach to this task, suggesting the judicial use of constraints within the grammar to overcome some of the difficulties. We call this formalism a *constraint-augmented grammar*. The constraints are expressed using the declarative constraints package SCREAMER+ [14], [16], an extension of SCREAMER [11], [12]. The section on 'Grammar Development and Maintenance' outlines a process for building a constraint-augmented grammar that supports knowledge capture. Finally, we discuss the benefits and limitations of our work, and relate it to other work in the field.

GRAMMAR-DRIVEN KNOWLEDGE ELICITATION
When eliciting knowledge, it is desirable to have a structured, declarative specification of the body of knowledge that needs to be acquired. This can be used as both the target of the knowledge acquisition process and the criterion by which the acquired knowledge is assessed. Formal grammars provide a means for specifying knowledge to be acquired, are structured and declarative, and are also widely understood by knowledge engineers and computer scientists. However, there is an important difference in the way that formal grammars are "traditionally" used, and the way that they have been applied here. Traditionally, grammars are used to solve the *parsing problem*; that is, to determine whether some *given*

---

[*] also affiliated with the Computing Science Department, University of Aberdeen, Old Aberdeen, AB24 3FX, Scotland, UK.

```
formation      → <lithology>+
lithology      → (<rock> <lithology-depth> [<lithology-length>])
rock           → (<rock-type> <rock-hardness>)
rock-type      → (shale | clay | chalk | granite | other)
rock-hardness  → (very-soft | soft | medium | hard | very-hard)
```

Figure 1: EBNF Grammar for acquiring knowledge of rock formations

```
(defclause formation ::= (repeat+ <lithology>))
(defclause lithology ::= (seq <rock> <lithology-depth> (optional <lithology-length>)))
(defclause rock      ::= (seq <rock-type> <rock-hardness>)
(defclause rock-type ::= (one-of 'shale 'clay 'chalk 'granite 'other))
(defclause rock-hardness ::= (one-of 'very-soft 'soft 'medium 'hard 'very-hard))
```

Figure 2: COCKATOO Grammar for acquiring knowledge of rock formations

text conforms to some *given* formal grammar. For example, a C compiler must determine whether a given program consists entirely of legal C syntax. In grammar-driven knowledge elicitation, however, one attempts to *acquire* structured text *such that* it conforms to the given grammar.

We chose to represent EBNF grammars using a "LISPified" equivalent to the meta-notation of EBNF. This meta-language needs to:

- provide for the definition of grammar clauses of the target language;

- differentiate between a grammar's terminal and non-terminal symbols;

- provide the standard operators of an EBNF grammar, namely, *sequential composition*, the expression of *alternatives*, *repetition*, and *optionality*.

We illustrate our ideas with a simplified example from the domain of petroleum geology, and, in particular, the acquisition of a case base of oil well drilling experiences. The knowledge captured in this way is used to support subsequent drill bit run modelling and optimisation; for example, to help choose the right drill bit for a given formation sequence [8]. The EBNF grammar in figure 1 both describes and specifies a rock formation and its constituent lithologies (basic rock-types). The same grammar can be expressed in COCKATOO's syntax as shown in figure 2. (The correctness of the domain knowledge in our example has *not* been verified by a domain expert.)

Note that the non-terminal symbols lithology-depth and lithology-length have numeric values, and are more difficult to specify concisely with a grammar. We return to this issue in the following section. Note also that although our simple example illustrates only repetitions of 'one or more' (in this case, lithologies), COCKATOO also provides for repetitions of 'zero or more' with the keyword 're-peat*'.

COCKATOO grammars are interpreted top-down, left to right. Usually, a special parameter to the defgrammar macro (not described here for lack of space) informs COCKATOO which is the 'top-most' grammar clause. So, for example, in the grammar of figure 2, we would tell COCKATOO to start with the formation clause. The interpretation of this clause leads to the acquisition of a repetition of lithologies, each in turn consisting of a sequence of a rock, a lithology-depth, and an optional lithology-length. A rock, in turn, consists of a sequence of a rock-type and a rock-hardness. The acquisition of either of these two non-terminal symbols involves the capture of a decision from the user among a number of distinct options (e.g., shale, clay, chalk, granite or other). These options are presented on-screen to the user by COCKATOO, so that a choice can be made and recorded. COCKATOO is sensitive to the number of possible values available. If there are too many values to be listed (i.e., more than a configurable upper limit), then the upper and lower bounds of the symbol (internally, a constraint variable) are provided to the user as additional support. If these values are not available at acquisition time, then the user is dependent upon the guidance provided by the knowledge engineer in the form of comments and questions (see below).

It is unrealistic to expect users to base their interaction with a knowledge acquisition tool on their understanding of an EBNF grammar. To help the user understand what information is required, and how it can be supplied, each clause of a grammar can be "decorated" with a question and/or a comment. A *question* should be a request for feedback which is directed at the user, such as "What is the rock-type?". A *comment* provides additional information, such as the meaning of particular terms, the exact format of the input, or other explanatory or "small-print" material. An example comment for the lithology clause might be "A lithology consists of a rock-type, a depth, an optional length, and a hardness".

188

Even when the expert provides the knowledge acquisition tool with the *knowledge content* required by a problem solver, the format of the expert's inputs are seldom exactly the same as the format required by the problem solver. Usually, some kind of syntactic transformation needs to be performed. To achieve this functionality, COCKATOO allows a post-processing function to be specified for each clause. This is a single argument function that is applied to the value acquired by the clause. As a simple example, a question which the expert answered with 'yes' or 'no' is more likely to be represented by a LISP program with t (true) or nil (false). The post-processing function converts. the terminology/representation of the expert to that of the problem solver. Note that the mechanism accommodates *arbitrary* post-acquisitional transformations. We have used this mechanism for simple *syntactic* transformations; we believe it could also be used as the call-out mechanism for supporting deeper *semantic* transformations. Currently, the full power of this mechanism is available only to knowledge engineers who are competent in LISP; later, we may devise a more user friendly interface for the description of such transformations. Additionally, we may allow adapters written in other languages to be linked in.

CONSTRAINT-AUGMENTED GRAMMARS

This section shows how a knowledge elicitation grammar can be augmented with constraint expressions. We claim that this can improve the conciseness and readability of the grammar, reduce its development time, and enhance its expressiveness. This view of knowledge elicitation is not inconsistent with the definition of a constraint satisfaction problem (CSP). (A CSP is defined by a set of *variables*, each of which has a known *domain*, and a set of *constraints* on some or all of these variables. The solution to a CSP is a set of *variable-value assignments* that satisfy all the constraints.) For example, consider a structured interview in which the answers to the knowledge engineer's agenda of questions are the variables of the problem, and there are concrete expectations about what their allowable values (the variables' domains) might be.

As we have seen, Grammar-Driven Knowledge Elicitation is a precise and powerful mechanism for acquiring knowledge. However, by combining the grammar-driven approach with constraint technology, we gain the following advantages.

Concise Specifications — Knowledge specifications for some tasks can be written much more concisely, thus giving a more readable specification, and also saving development time.

Single-Input Property Checking — The required properties of each user input can be checked at *acquisition time*, rather than prior to problem solving or at problem-solving time. That is, inadmissible values are identified early in the knowledge acquisition cycle. The properties that help to identify the admissibility of an input value are expressed naturally as constraints.

Multiple-Input Property Checking — Required properties of *multiple* inputs can also be checked at *acquisition time*. A property of this kind is expressed as a constraint among *multiple* inputs.

Reactive User-Interfaces — Constraints among multiple inputs can be constructed in such a way that the user-interface appears to react to the user's inputs. For example, the choice of a particular value for one input might narrow the domain of another.

Concise Specifications

The value of a COCKATOO clause can be specified by combining concrete values with the keywords seq, one-of, optional, repeat+ and repeat*. Alternatively, a clause can be defined as an *arbitrary* LISP expression, such as a constraint expression. For example, the following concise clause accepts only an integer in the (inclusive) range 10 to 5000:

```
(defclause lithology-depth ::=
  (an-integer-betweenv 10 5000)
  :comment "The depth is given in metres (10 <=
depth <= 5000)")
```

With a purely grammar-driven approach, a part of the acquisition grammar would have to be dedicated to accepting either the sequence of characters that compose the integers of the range, or the enumeration of all acceptable values. For problems such as this, the simple constraint-based clause is much more maintainable than the equivalent grammar-based solutions without constraints.

Single-Input Property Checking

In the previous section, we argued that a grammar would be capable of describing the set of integers in the range 10-5000, but the introduction of constraints made the solution much more concise. For that problem, the constraint-based approach was no more *powerful* (in terms of expressiveness) than the purely grammar-based approach[1], though it clearly offered advantages. However, a constraint-augmented grammar also provides for the verification of properties beyond the power of a purely grammar-driven approach. As an example, consider prime numbers. It is not possible to define a formal grammar that admits any prime number, but disallows non-primes. However, a constraint-augmented grammar can include a clause that admits only prime numbers by constraining the input value to satisfy a predicate that tests for primeness[2].

In LISP, membership of a type can be subject to satisfaction of a *arbitrary* LISP predicate, so the mechanism for checking the properties of a single input value is general and powerful.

---

[1] Both approaches solved the problem.

[2] The example is given in full in [16].

Multiple-Input Property Checking

Another way of specifying values that could not be expressed by a context-free grammar is by asserting constraints across multiple input values. For example, a context-free grammar would not be able to constrain two variables to have different values unless it *explicitly* represented all those situations in which the values were different. At best, this represents much work for the implementer of the grammar. If the variables have an infinite domain, however, it is not even possible. The following clause returns a sequence of two rock-types which are constrained to be different.

```
(defclause rock-types ::=
   (let ((type-1 (make-variable))
         (type-2 (make-variable)))
      (assert! (not-equalv type-1 type-2))
      (seq type-1 type-2)))
```

A similar technique can be used in the grammar given earlier to prevent the rock-types of consecutively acquired lithologies to be the same (if consecutive rock-types were the same, it would be a single lithology). When acquiring a value for this clause, the second value must be different to the first:

```
LISP> (acquire (find-clause 'rock-types))
Input a value: granite
Input a value: granite
That value causes a conflict. Please try another
value...

Input a value: shale
(GRANITE SHALE)
```

Here, (GRANITE SHALE), is the return value of the rock-types clause.

Reactive User-Interfaces

Constraints can also be used to modify the behaviour of the acquisition tool, depending on the values supplied by the expert. The idea is that inputting a value in answer to one question may trigger a reduction in the set of possible answers to a different question. This is the issue of *reactive knowledge acquisition* mentioned earlier. Reconsider the example of acquiring a pair of rock types that are constrained to be different. This time, we reuse the clause first given in figure 2 that acquires a single rock type:

```
(defclause rock-type ::=
   (one-of 'shale 'clay 'chalk 'granite 'other))
```

When this clause is interpreted, it creates a constraint variable whose domain (set of possible values) consists of the rock types shale, clay, chalk, granite and other. COCKATOO uses the domain of a variable when displaying the possible input values to the user. The following clause uses the rock-type clause to return a sequence of two rock types. The values of the rock types type-1 and type-2, which are not known until acquisition time, are constrained to be different:

```
(defclause rock-types ::=
   (let ((type-1 (find-clause 'rock-type))
         (type-2 (find-clause 'rock-type)))
      (assert! (not-equalv (acquired-valuev type-1)
                           (acquired-valuev type-2)))
      (seq type-1 type-2)))
```

Note that the constraint on type-1 and type-2, imposed by the assert! function, is declarative and therefore symmetrical, allowing either of the two values to be acquired first. The return value, on the other hand, is a sequence that is interpreted such that type-1 is acquired before type-2. If the expert inputs shale as the first value of the pair, it should not be offered as a possible value for the second value of the pair. Instead, the user-interface should react to the expert's inputs, as illustrated below:

```
LISP> (acquire (find-clause 'rock-types))

The possible values are:
   A.  SHALE
   B.  CLAY
   C.  CHALK
   D.  GRANITE
   E.  OTHER
Which value? : granite

The possible values are:
   A.  SHALE
   B.  CLAY
   C.  CHALK
   D.  OTHER
Which value? : shale
(GRANITE SHALE)
```

When acquiring the second rock-type, GRANITE was not offered as a possible value because choosing that value would be inconsistent with the disequality constraint. Such behaviour cannot be realised by a context-free grammar because the rock-type is not known until acquisition time. A context-free grammar cannot build in such conditions at 'compile time'.

We have shown that the determination of a variable's value at acquisition time can cause the domain of another variable to be reduced. Sometimes, the domain of a variable might be reduced to a single value, causing that variable to become bound. When this happens, the value of that variable need not be acquired from the expert, as it has already been inferred.

GRAMMAR DEVELOPMENT AND MAINTENANCE

It is important for a knowledge specification to be easily readable so that persons other than the KA tool developer can understand it. Readable specifications tend to be easier to write, discuss, and maintain. COCKATOO has two main features that enhance both the readability and maintainability of its knowledge specifications. Firstly, it has developed an approach based on the use of (EBNF-like) formal grammars, which are a well-known type of formal specification, and in widespread use. Secondly, COCKATOO makes a clear separa-

tion between the knowledge specification and the acquisition engine that acquires the knowledge. This leads to concise specifications, which contain only pertinent material, as well as a general purpose acquisition tool which is reusable across domains. By way of contrast, consider a custom-tailored acquisition tool that embeds the knowledge specification within its program's source code. Such a tool would not be reusable across different problem domains, and even with optimal coding style, only those with knowledge of the programming language would be able to understand the specification.

COCKATOO already provides mechanisms for specifying the required knowledge at a "high level". That is, during (grammar) development, the knowledge engineer can concentrate on the nature of the knowledge to be acquired, rather than the program that acquires it. Grammar development using COCKATOO is a (cyclic) refinement process, which includes the following chronological stages.

Knowledge Analysis — The aim of this stage is to capture the most important concepts of the domain and the relationships between their instances. In effect, we aim to derive a basic domain ontology.

Grammar Construction — In this stage, we decide which of the ontological elements from the previous stage will be included in the knowledge capture. A further analysis of these elements (for example, a structural decomposition) leads to a grammar that captures the basic knowledge requirements in terms of those elements and their multiplicities (e.g., one-one, one-many, many-many).

Adding Constraints — An optional stage to enhance the grammar with *constraints*. When used, the aim of the stage is twofold: firstly, to remove unwanted or nonsensical input combinations from the specification; and secondly, to eliminate redundant questions.

Embellishment — Embellishing the grammar with questions and comments.

Notice that the system's communication with the expert is not considered until the final stage of development, reflecting the attention paid to the correctness of the knowledge specification in the early stages.

The examples presented throughout this paper have demonstrated COCKATOO's flexibility for use in many different domains. COCKATOO can also be configured *quickly* for use in a new domain. For example, three sample grammars presented in [16] were developed (and refined) in less than a day each! The main reason for the ease with which COCKATOO can be reused is the clear separation between the data that drives knowledge acquisition (the grammar) and the more generic tool that processes the data (the COCKATOO acquisition engine). COCKATOO is, in effect, a knowledge acquisition *shell* that supports the building of custom-tailored KA tools.

Although it was developed to acquire *knowledge bases* for use within the MUSKRAT toolbox [16], a KA tool such as

COCKATOO has the potential to be applied to a very wide range of application domains. Not only can it acquire simple knowledge elements, it can manage complex constraint relationships between them, and post-process user inputs for further compatibility with other tools. With regard to COCKATOO's suitability for different acquisition tasks, it could be used in most situations that involve a substantial amount of numerical, textual or symbolic user input. It is well-suited to supporting knowledge acquisition for both classification and configuration (or limited design) tasks. For classification tasks, we must acquire example cases and their associated class; for configuration tasks, the building blocks of the design are well-known, but their combinations may be explored. Although all the design decisions are made by the human user, the output is nevertheless constrained to be within the "space" specified by the grammar.

DISCUSSION

This paper has argued the value of a declarative specification of the knowledge to be acquired, and introduced the COCKATOO tool, which acquires knowledge by interpreting a constraint-augmented grammar. This approach offers enhanced readability, eased maintenance, and a reduced initial development effort compared with the construction of multiple customised tools for different domains. Augmenting a context-free grammar with constraints increases both the expressiveness and conciseness of the notation. The power of the tool that interprets the notation is also increased because in some situations its behaviour can be altered by the user's responses to questions. Conciseness of the notation is improved because admissible values do not always have to be detailed down to the level of individual characters or symbols.

Related Work

COCKATOO is an automated knowledge elicitation tool, but differs considerably from current knowledge elicitation tools based on repertory grids, sorting, and laddering (see, for example, [1], [10], and PC-PACK[3]), because they cannot be tightly coupled with an application. The knowledge acquired using these tools must be post-processed "by hand" before they can be used by a problem solver or other application program. COCKATOO, on the other hand, provides a very general post-processing facility which allows the acquired knowledge to be packaged in a form suitable for subsequent use.

Generalised Directive Models (GDMs) [13], [7] also use grammars, but for a different purpose to COCKATOO. COCKATOO uses a grammar to guide the acquisition of *domain knowledge* from a *domain expert*, such that the knowledge can be used by an *existing* problem solver. GDMs, on the other hand, apply grammars to assist *knowledge engineers* with *task decomposition* when *building* a knowledge-based

---

[3] PC-PACK is a software package marketed by Epistemics Ltd. See www.epistemics.co.uk

system. The purpose of a GDM grammar is to guide the knowledge engineer to classify the task(s) at hand, so that an appropriate knowledge elicitation tool can be selected. Thus, the grammars of the two approaches describe sentences of quite different natures: a COCKATOO sentence describes a domain structure, whereas a GDM sentence describes the decomposition of a task into subtasks and their respective types. It may also help to consider the meanings of the terminal symbols in each of the two formalisms. A terminal symbol of a COCKATOO grammar represents a domain concept or value; a terminal symbol of a GDM represents a task type whose association with a knowledge elicitation tool is known. The two approaches are complementary, and could even be used together – a terminal node of the GDM grammar could be associated with COCKATOO as the most appropriate elicitation tool for the node's task type.

It is interesting to compare COCKATOO with the Protégé project and, in particular, the Maître tool [2]. Protégé, like COCKATOO, is a general tool (a "knowledge acquisition shell") whose output is in a format that can be read by other programs (CLIPS expert systems). Also, before using Protégé to acquire knowledge, the knowledge engineer must first configure it with an "Ontology Editor" subsystem, called Maître. This tool enables the knowledge engineer to define an ontology, which is then used as the basis for knowledge acquisition. The ontology plays the same role in Protégé as the constraint-augmented grammar in COCKATOO — it specifies the knowledge to be acquired. Another subsystem of Protégé, called Dash, can be used interactively to define a graphical user-interface for the KA tool. Although the graphical user interfaces are very appealing, we do not believe that Protégé has the same knowledge specification power as COCKATOO, since it is not supported by an underlying constraints engine.

The idea of minimising the number of questions asked of the user was inspired by the questioning techniques of MOLE [3], and the intelligent mode of the MLT Consultant [5]. However, the approach taken by COCKATOO is different from both of these systems. MOLE reduces the amount of questioning by making intelligent guesses about the values of undetermined variables, and subsequently requesting the user's feedback. MLT Consultant uses an information theoretic measure to determine which questions are asked. In contrast, COCKATOO uses local propagation techniques to identify redundant questions.

A so-called *Adaptive Form* [4] is a graphical user-interface for acquiring structured data that modifies its appearance depending on the user's inputs. For example, a form for entering personal details would only show a field for entering the spouse's name if the user had entered *married* in the *marital status* field. Although this kind of behaviour is very similar to the reactive knowledge acquisition of COCKATOO, Adaptive Forms are driven by (context-free and regular-expression) grammars alone, and do not support more complex constraints. The system uses *look-ahead parameters* to decide which unbound fields to display at any given time. We

also note that Frank and Szekely extended their grammar notation to include 'labels', which have the same function as the questions of COCKATOO. They do not provide the equivalent of comments, name spaces, or post-processing. The Amulet system [6] does use a constraint solver to manage its user-interface, but employs it to control the positioning and interactions of graphical objects, rather than to support knowledge acquisition.

Limitations

COCKATOO currently does not allow the user to backtrack from a given user input, and try something else. Once an input has been entered, the user is committed to that value and cannot change it later. This is a serious limitation because not only does it not allow for typographical errors; it also prevents the user from experimenting with the COCKATOO grammar in a 'what-if' mode. Of course, COCKATOO can always be aborted and the acquisition restarted from the beginning, but a more flexible backtracking capability is a desirable feature that should be addressed. Ideally, at each stage of the acquisition process the user should be given the option to go back to the previous stage, retract the previous input, and input a new value. The problem is that retracting an input is not simple, because the constraint-based assertions associated with that input may already have caused propagation to other constraint variables. To retract an input, one must be able to recover the states of *all* constraint variables before the assertion was made. One option to achieve this functionality is to record the states of all constraint variables before each user input; another option is to record the changes that occur after each user input, so that they may be reversed. Ideally, the constraints package would provide a retraction facility of its own [16].

There are two types of propagation that should be supported by a KA tool for MUSKRAT; namely *inter-KB propagation* and *intra-KB propagation*. In the former case, knowledge is propagated from one knowledge base to a different knowledge base. In the latter case, knowledge is propagated from one part of a knowledge base to a different part of the *same* knowledge base.

For inter-KB propagation, we require a mechanism by which the knowledge contained in an *existing* knowledge base is made available to COCKATOO for acquiring a different, but related, knowledge base. We have not yet addressed this problem. However, COCKATOO already provides a mechanism for intra-KB propagation through the functions `find-clause` and `acquired-valuev`. The function `find-clause` can be used from within a grammar to search for a known clause, thus offering a facility for grammar introspection. The function `acquired-valuev` is used to make assertions about the value that a clause (eventually) acquires. These two functions can therefore be used together to specify at acquisition time the knowledge that some other clause should acquire. This is the behaviour of intra-KB propagation.

We feel that the usability of COCKATOO would be significantly improved by the addition of graphical user-interfaces at two levels. Firstly, a graphical user-interface 'front-end' to COCKATOO would provide the opportunity for enhanced end-user support; for example, through the use of distinct graphical input forms (with appropriate widgets, such as text boxes and drop-down menus). Secondly, a graphical user-interface could provide useful support for the acquisition of grammars, so that the knowledge engineer would no longer have to input COCKATOO grammars in their 'LISPified' syntax. Such a 'meta-tool' would output a COCKATOO grammar (perhaps as the result of post-processing). It would be interesting to investigate whether COCKATOO is flexible enough to act as both the meta-tool and the domain expert's tool.

COCKATOO will be used by a class of undergraduate students in the autumn of 2001, and we expect it to be used subsequently by the Advanced Knowledge Technologies (AKT) project.

ACKNOWLEDGEMENTS

REFERENCES
[1] Diaper, D., (1989), "Knowledge Elicitation: Principles, Techniques and Applications", Ellis Horwood, Chichester, England, UK.

[2] Eriksson, H., Puerta, A. R., Gennari, J. H., Rothenfluh, T. E., Samson, W. T., Musen, M., (1994), "Custom-Tailored Development Tools for Knowledge-Based Systems", Technical Report KSL-94-67, Section on Medical Informatics, Knowledge Systems Laboratory, Stanford University, California, USA.

[3] Eshelman, L., (1988), "MOLE: A Knowledge-Acquisition Tool for Cover-and-Differentiate Systems", in Marcus, S., (Ed.), "Automating Knowledge Acquisition for Expert Systems", Kluwer Academic Publishers, pp. 37-80.

[4] Frank, M. R., Szekely, P., (1998), "Adaptive Forms: An Interaction Technique for Entering Structured Data", Knowledge-Based Systems, Vol. 11, pp. 37-45.

[5] Kodratoff, Y., Sleeman, D., Uszynski, M., Causse, K., Craw, S., (1992), "Building a Machine Learning Toolbox", in Enhancing the Knowledge Engineering Process, Steels, L., Lepape, B., (Eds.), North-Holland, Elsevier Science Publishers, pp. 81-108.

[6] Myers, B. A., Mcdaniel, R. G., Miller, R. C., Ferrency, A. S., Faulring, A., Kyle, B. D., Mickish, A., Klimovitski, A., And Doane, P., (1997), "The Amulet Environ-

ment: New Models for Effective User Interface Software Development", IEEE Transactions on Software Engineering, Vol. 23, No. 6, pp. 347-365.

[7] O'Hara, K., Shadbolt, N., Van Heijst, (1998), "Generalised Directive Models: Integrating Model Development and Knowledge Acquisition", International Journal of Human-Computer Studies, Vol. 49, No. 4, pp. 497-522.

[8] Preece, A., Flett, A., Sleeman, D., Curry, D., Meany, N., Perry, P., (2001), "Better Knowledge Management through Knowledge Engineering", IEEE Intelligent Systems, Vol. 16, No. 1, pp. 36-43.

[9] Reichgelt, H., Shadbolt, N., (1992), "ProtoKEW: A knowledge-based system for knowledge acquisition", in Artificial Intelligence, Sleeman, D, and Bernsen, NO (Eds.), Research Directions in Cognitive Science: European Perspectives, volume 6, Lawrence Erlbaum, Hove, UK.

[10] Shadbolt, N. R., (2000), Knowledge Elicitation Techniques In Knowledge Engineering & Management: The CommonKADS Methodology. G. Schreiber, H. Akkermans, A. Anjewierden, R. de Hoog, N. Shadbolt, W. van der Velde & B. Wielinga. Pub: MIT Press.

[11] Siskind, J. M., McAllester, D. A., (1993), "Nondeterministic LISP as a Substrate for Constraint Logic Programming", in proceedings of AAAI-93.

[12] Siskind, J. M., McAllester, D. A., (1994), "SCREAMER: A Portable Efficient Implementation of Nondeterministic Common LISP", Technical Report IRCS-93-03, Uni. of Pennsylvania Inst. for Research in Cognitive Science.

[13] Van Heijst, G., Terpstra, P., Wielinga, B., Shadbolt, N., (1992), "Using generalised directive models in knowledge acquisition", in Proceedings of EKAW-92, Springer Verlag.

[14] White, S., Sleeman, D., (1998), "Constraint Handling in Common LISP", Department of Computing Science Technical Report AUCS/TR9805, University of Aberdeen, Aberdeen, UK.

[15] White, S., Sleeman, D., (1999), "A Constraint-Based Approach to the Description of Competence", in Fensel, D., Studer, R., (Eds.), Proceedings of the Eleventh European Workshop on Knowledge Acquisition, Modelling, and Management (EKAW-99), LNCS, Springer Verlag, pp. 291-308.

[16] White, S., (2000), "Enhancing Knowledge Acquisition with Constraint Technology", PhD Thesis, University of Aberdeen, Scotland, UK.

From Thesaurus to Ontology

B. J. Wielinga   A. Th. Schreiber   J. Wielemaker   J. A. C. Sandberg

University of Amsterdam, Social Science Informatics

Roetersstraat 15, NL 1018 WB Amsterdam, The Netherlands

{wielinga,schreiber,jan,sandberg}@swi.psy.uva.nl

Abstract

Thesauri such as the Art and Architecture Thesaurus (AAT) provide structured vocabularies for describing art objects. However, if we want to create a knowledge-rich description of an (image of an) art object, such as required by the "semantic web", thesauri turn out to provide only part of the knowledge needed. In this paper we look at problems related to capturing background knowledge for art resources. We describe a case study in which we attempt to construct an ontology for a subset of art-object descriptions, namely antique furniture, using AAT as well as metadata standards as input. We discuss the representation requirements for such an ontology as well as representational problems for our sample ontology with respect to the emerging web standards for knowledge representation (RDF, RDFS, OIL).

Keywords

Ontology construction, thesaurus, web standards, image indexing

INTRODUCTION

In this paper we address the problem of capturing knowledge needed for indexing and retrieving image information using highly structured semantic descriptions. Such structured descriptions can be much richer than the traditional "set of terms approach". In fact they come nearer to a description in natural language, often considered to be the ideal way of describing and indexing pictorial material. In order to circumvent the problems of ambiguity in natural language descriptions and queries, structured descriptions should be limited to a fixed set of predefined structures and a closed vocabulary. In this paper we assume that the structured descriptions are created by a human annotator using specialized tools. Two related problems arise in this approach: (1) how can a human be supported during the annotation process, and (2) where does the vocabulary or ontology for filling in the structured descriptions come from? The solution to these problems that we will pursue in this paper is to extend an existing thesaurus with additional knowledge such

that is becomes an ontology suitable to support rich structured descriptions. The paper is structured as follows. First we will discuss various alternative approaches to image indexing and retrieval and the requirements that they pose on the vocabulary. Then we will discuss the properties of a particular thesaurus, the Art and Architecture Thesaurus (AAT) in the light of these requirements. We then discuss the construction of an ontology for antique furniture using AAT and existing metadata standards. With respect to knowledge representation we have tried to adhere to the new web standards as much as possible and we discuss problems arising in the pursuing this objective.

IMAGE RETRIEVAL

There are several paradigms for image retrieval currently in use:

- Content-based image retrieval (CBIR)
- Text-based image retrieval
- Field-based image retrieval
- Structure-based image retrieval
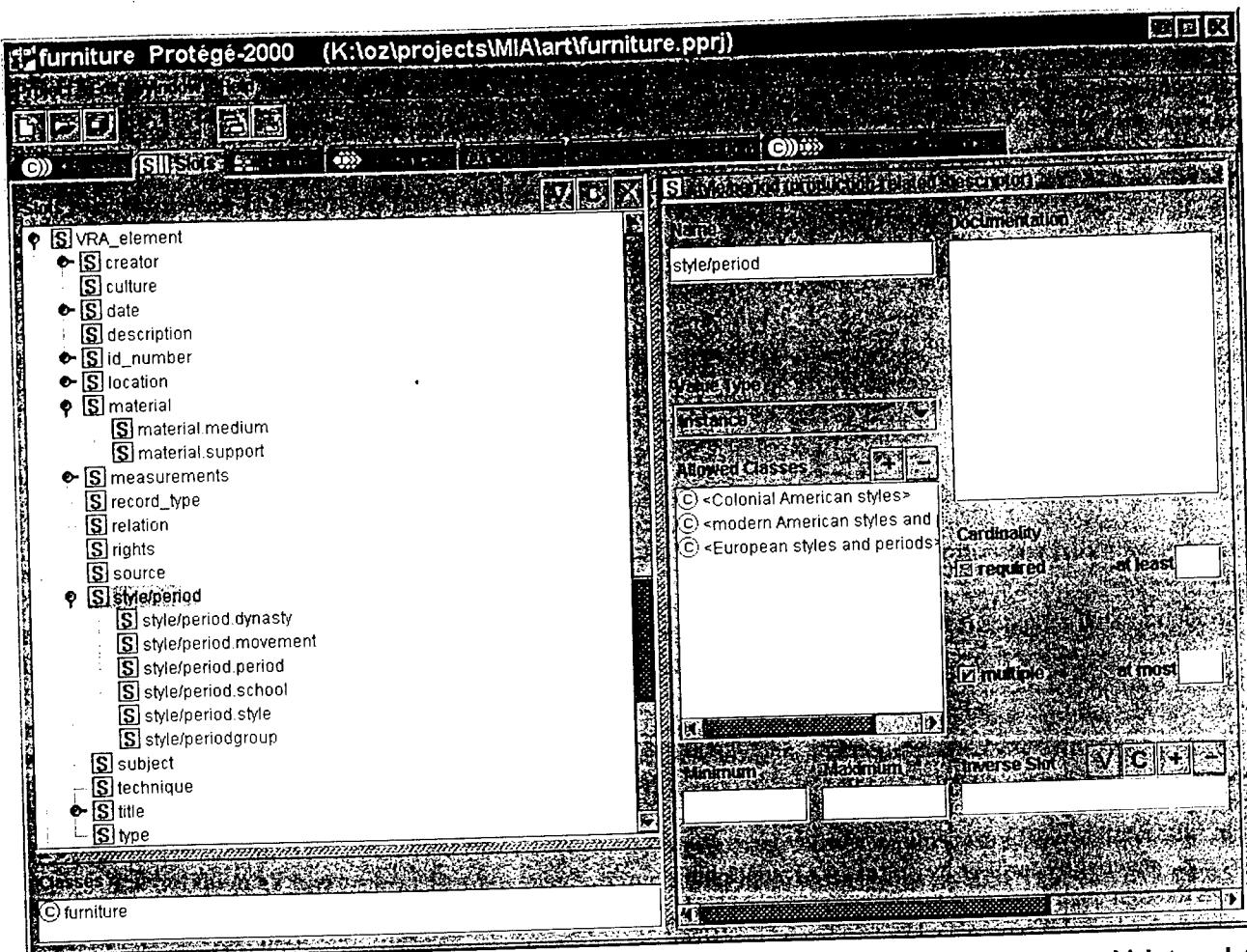
We will discuss each of these approaches in turn.

The content-based image retrieval paradigm indexes images on their intrinsic and primary features, which are computed by various image analysis algorithms. These features include color structure, shape properties, textures etc. This paradigm will not be discussed here since the link with the more semantically oriented other methods is very difficult to make given the current state of the art in mage processing.

There are a number of different forms that text-based image retrieval can take:

- Keyword search with free vocabulary
- Keyword search with a closed vocabulary
- Thesaurus-based search, where not only the vocabulary is closed but also hierarchical (broader and narrower terms) and other relations can be taken into account in the search process.

The general characteristic of this method is that the query is composed of a (possibly Boolean structured) set of terms. The index usually consists of an unordered set of terms. The indexing and retrieval process can both be supported by tools to browse and select terms from the vocabulary. Such browsers are available for large thesauri such as AAT, LCSH and ICONCLASS.

Figure 1: VRA element set defined as Protégé slots. The element qualifiers are defined as subslots, which translate to RDFS subproperties. For a particular visual object multiple instances of a data element can be defined. For example, one can define multiple styles for a piece of furniture.

The field-based approach describes or retrieves an item not by a set of keywords, but by a set of attribute-value pairs. Typically, a metadata schema is defined that describes the elements (fields) and some indication is given what values can be assigned to a particular field. The most widely used schema is the Dublin Core metadata template (DC) [7] for describing documents in general. For specialized domains such as the description of art objects in museums, qualified versions of DC have been created, such as the Visual Resource Association (VRA) Core Categories. VRA version 3.0 [3] defines 17 data elements for describing visual resources. Some data elements have qualifiers which can (optionally) be used to specify more detailed semantics of the data value. For example, VRA defines a data element style/period with qualifiers such as style, period and school. The data elements are linked to one or more corresponding DC elements. For example, style/period is linked to the DC elements coverage and subject. For a particular visual object multiple instances of a data element can be defined. For example, one can define multiple styles for a piece of furniture.

Fig. 1 shows a representation of the set of VRA elements. This representation was developed with the help of the Protégé-2000 tool [6]. The data elements are represented as Protégé slots; the qualifiers as subslots, allowing one to specialize the value set of the element for the qualifier. We come back to this in more detail in the discussion about the ontology.

Many of the field-based initiatives recommend the use of closed vocabularies such as AAT [10], but do not associate particular parts of a thesaurus with a field. As a consequence the only support that a human indexer has is the thesaurus browser. To improve the support for indexing a mapping is required from the fields to particular parts of the thesaurus, such that the indexer is only presented with terms that are relevant for a particular field. As we will argue in subsequent sections of the paper, this is not always easy to do.

Where the field-based approach essentially uses a flat structure of attribute-value pairs, the structure-based approach allows more complex descriptions involving relations. For ex-

ample, a description of a piece of furniture can include a description of its components, e.g. a drawer of a chest. The components are again objects that can be described using a number of attributes such as material, size, shape. Components can even have components themselves, e.g. drawers can have handles. The structure-based approach introduces a large degree of complexity in the indexing process. Relational descriptions can vary widely between different categories of objects. Furniture can have components, but paintings in general do not have components, they can be described by a complex subject matter structure. A solution to the problem of complexity of the indexing process is to use contextual information to constrain the relations and terms presented to the indexer. We first discuss what the knowledge requirements are with respect to existing thesauri in order to create knowledge-rich art-object descriptions.

ANALYSIS OF EXISTING THESAURI

A first requirement for a thesaurus to be useful in the field- and structure-based approach is that it provides a hierarchical structure that has an unambiguous interpretation. Some hierarchically organized thesauri, such as ICONCLASS [12], mix the sub/super class relation with a part-of relation [1]. AAT uses a strict sub/super class relation in a single inheritance hierarchy. The single inheritance limits the amount of information about a term that can be derived from its position in the hierarchy, as terms can be classified in multiple ways, e.g., material by form or by origin. AAT attacks this problem through qualification of certain terms. For example, the concept **landscape** is represented by two terms: **landscape (representation)** and **landscape (environment)**. This solution has some drawbacks: the distinction between the two qualified terms may not be clear to a user and it is difficult to decide where subclasses of the concept should be placed.

A second requirement is that fields in a description can be linked to particular parts of the thesaurus. For example, the field material should be linked to the part of a thesaurus that contains a hierarchy of material types. In some cases this is straightforward. AAT for example has a hierarchy **Materials**, which clearly defines the terms that can be used as value for the material field. However, there are many cases where values to be assigned to a field are scattered over several parts of the thesaurus. In AAT certain types of porcelain (e.g. five-colored porcelain or Wucai) are situated under <**Chinese ceramics styles**>, while **ironstone** (a semi-porcelain) is located in the **Object Genres** hierarchy. This is not only a problem when the user is presented with a hierarchy or list of values from which a selection has to be made, but also a problem for search processes that use inheritance. Searching for a **ceramics** object requires knowledge about the various parts of the thesaurus hierarchy.

A third requirement follows from the complexity of the indexing space. A human indexer who uses the structure-based approach, will be confronted with large sets of possible val-

ues to choose from. For example, the **Materials** hierarchy in AAT contains several hundreds of terms. A solution to this problem is to constrain the value-sets for a particular field, based on a partial description of the image or object. For example, when it is known that an object is a piece of furniture, the possible materials, styles and periods of that object are highly constrained. In some cases various fields can be inferred from information available in other fields. If an object is described as a **Ming vase**, the material is **porcelain**, the region of origin is **China** and the period is between 1368 and 1644.

EXTENDING THE AAT

As the basis for building an ontology for indexing images, we have used the Art and Architecture thesaurus. The AAT is the most elaborate and most standardized body of knowledge concerning the classification of art objects. It contains about 28.000 main terms and 120.000 terms in all, including synonyms and related terms. Besides it offers scope notes: textual definitions of AAT concepts for a major part of main terms. The AAT concepts are represented in 33 hierarchies. A particular concept occurs only once in the full AAT hierarchy, following the ISO 5964 standard (Guidelines for the Establishment and Development of Multilingual Thesauri). AAT uses intermediate concepts ("guide terms") to group concepts lower in the hierarchy. For example, <**ceramic and ceramic products**> is such an intermediate concept.

In an early attempt to use the AAT thesaurus as an ontology [13] we treated the main terms as concept names in the knowledge base. Although this is possible since each main term in AAT is unique, it causes problems when a concept can also be identified by its synonyms, as is the case in WordNet synsets [9]. Searching AAT for the term wood returns as first concept **woods** (area with trees) rather than **wood** (material). It was decided to represent each concept in the knowledge base by a unique identifier, derived from the AAT record number.

The full AAT hierarchy was converted into a hierarchy of concepts, where each concept has a label slot corresponding with the main term in AAT and a synonyms slot where alternate terms are represented. The knowledge base is represented in RDFS [2]. We constructed an RDFS browser to inspect and browse the hierarchy. A snapshot of the browser is shown in Fig. 2.

A second step was to augment a number of concepts with additional slots and fillers. For example, concepts representing a style or period were augmented with slots time period from, time period to, general style and region. The values for these slots were partly derived using explicit tables of periods, and partly by using the intermediate concepts in AAT. For example, the British furniture style **George IV** (1820-1830) is augmented with **Regency** as a more general style indication. A third step was to add knowledge about the relation between possible values of fields and nodes in the knowledge base.
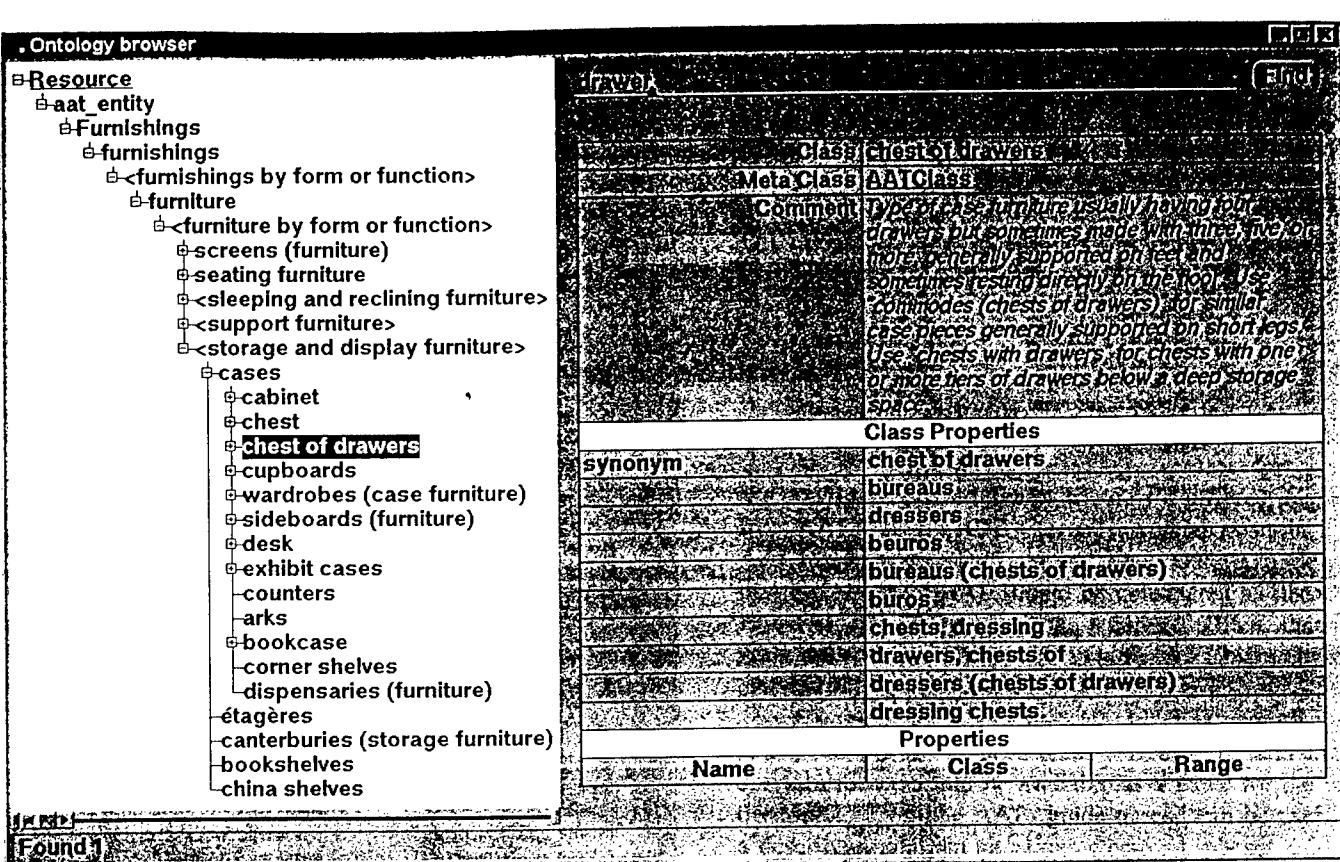
Figure 2: Part of the AAT hierarchy. The snap shot is of our RDFS browser in which an RDFS version of AAT has been loaded

The nature of this knowledge is discussed below.

AN ONTOLOGY FOR FURNITURE

We developed an ontology for a subset of art objects, namely antique Western furniture. This ontology was developed in three steps:

1. Construction of a description template for antique furniture: what kind of information does one want to record for a particular furniture item?

2. Linking the furniture properties to specific subsets of AAT that can be used as values for furniture properties.

3. Describing additional domain knowledge, in particular about constraints between furniture-property values.

Furniture description template

Fig. 3 shows the template we developed for describing a piece of antique furniture. A piece of furniture can be described through 25 "descriptors".[1] Of these 25 descriptors, 17 are derived from the VRA Core Categories [3] (see Fig. 1). The other descriptors are based on the results of the European GRASP project [13]. This project developed an

ontology for describing and retrieving stolen art objects. The following "GRASP" slots were added to the VRA elements: functional context (e.g., religious), intended location, form, color, color cardinality (e.g., monochrome), color type (e.g., primary colors), marking, and component. This last descriptor allows for describing subparts of a piece of furniture (e.g., the feet or drawers of a chest). AAT provides a special hierarchy of terms for this, namely <furniture components>. Qualifiers of the data elements were defined as subslots.

We used Protégé-2000 [6] as ontology editor with RDFS as the underlying representation language. The furniture concept is represented as a Protégé class and the descriptors as template slots of this class. Protégé slots are translated into RDFS properties; the qualifiers are translated into subproperties.

This simple representation leads to a long unstructured list of furniture descriptors. In addition, we also wanted to represent natural groups of descriptors. We distinguished four descriptor groups:

1. *Production-related descriptors*: e.g., creator ("maker"), style/period, technique.

2. *Physical descriptors*: e.g., measurements, color, material, etc.

---

[1] The term descriptor is sometimes used to indicate an attribute value, but we use it here in the "attribute" sense.

Project Edit Window Help

Classes S   C

Relationship Superclass V C X

- ○ THING ▲
  - ○ⒸSYSTEM-CLASS ▲
    - ●Ⓒ CLASS ▲
    - ●Ⓒ SLOT ▲
      - ●Ⓒ STANDARD-SLOT D
        - ●Ⓒ art_descriptor
          - Ⓒ functional_descriptor
          - Ⓒ general_descriptor
          - Ⓒ natural_hierarchy
          - Ⓒ physical_descriptor
          - Ⓒ production-related_descriptor
          - Ⓒ subpart
    - ●Ⓒ FACET ▲
    - ●Ⓒ CONSTRAINT ▲
  - ●Ⓒ AAT_category
  - Ⓒ ICONCLASS_category
  - Ⓒ ULAN_name
  - Ⓒ furniture

Superclasses
- ○ THING ▲

○ furniture (art_class)

Name
furniture

Template Slots

Name	Type	Cardinality	Other Facets
S color	Instance	multiple	classes={c50124640,c50131648}
S component	Instance	multiple	classes={furniture}
S creator	Instance	multiple	classes={ULAN_name}
S culture	Instance	multiple	classes={c50107946,c50111149}
S date	String	multiple	
S description	String	multiple	
S form	Instance	multiple	classes={c50136953}
S functional_context	Symbol	multiple	allowed-values={lady's,recreational,profa...
S id_number	String	multiple	
S intended_location	Symbol	multiple	allowed-values={church,wall,roof,garden,...
S location	String	multiple	
S marking	String	multiple	
S material	Instance	multiple	classes={c50010358,c50011915}
S measurements	String	multiple	
S pattern	Instance	multiple	classes={c50168686}
S record_type	Symbol	required single	allowed-values={work,image}
S relation	Any	multiple	
S rights	String	multiple	
S source	String	multiple	
S style/period	Instance	multiple	classes={c50018032,c50018115,c50111...
S subject	Instance	multiple	classes={ICONCLASS_category}
S technique	Instance	multiple	classes={c50155056}
S texture	Instance	multiple	classes={c50056362}
S title	String	multiple	
S type	Instance	multiple	classes={c50048628}

Figure 3: Furniture description template. This template containis the 17 VRA data elements plus 8 additional elements

3. *Functional descriptors*: related to the intended usage of the furniture item, e.g., intended location functional context.

4. *Administrative descriptors*: e.g., collection ID, rights, current location.

It is tempting to represent these descriptor groups as an aggregation: a furniture description has four subparts, one for each descriptor group. However, one requirement we had with respect to the use of RDFS/RDF was that a general RDF-aware browser should be able to interpret as much as possible the resulting furniture-item description. From this point of view the representation of a furniture template as consisting of subparts with their own set of descriptors is cumbersome. It would mean that in the RDF representation there is only an indirect link from the furniture instance to the descriptor triple:[2]

```
<rdf:Description about="furniture36">
  <physical_description rdf:resource="phdesc53"/>
</rdf:Description>
```

[2] As we will see further on, the specification of material in the example using a class is problematic.

```
<physicalDescription rdf:about="phdesc53">
  <material rdf:resource="&aat;mahogany"/>
</physicalDescription>
```
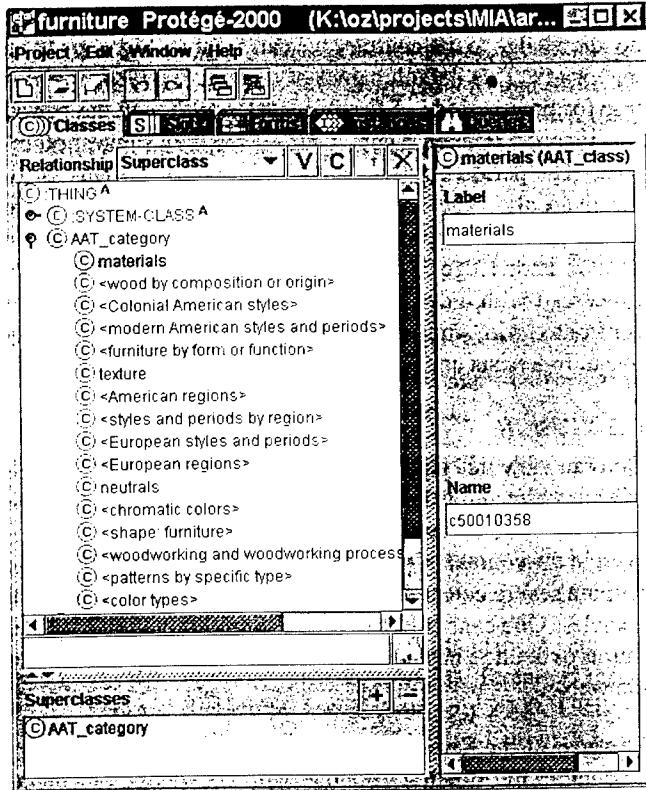
We therefore refrained from using a part-of organization of descriptors. Instead, we defined a metaclass **art descriptor** with the descriptor groups as subclasses. Subsequently, the furniture slots were defined as instances of the appropriate art-descriptor subclass. For example, the property technique is an instance of a **production-related descriptor**. The descriptor metaclasses are listed in Fig. 3 (see the "class" tab at the left).

One of the reasons we prefer Protégé as RDFS editor is that it supports, as RDF/RDFS does, treating instances as classes and vice versa. Not allowing this is in fact a weakness of many description-logic languages, which adhere to a strict separation. Martin [8] considers class/instance flexibility as a central requirement for adequate conceptual modelling.

The VRA element type plays a special role. This descriptor is used to represent the natural category to which the furniture item belongs, e.g. a case. For furniture we used the

198

AAT hierarchy under the "guide term" <furniture by form or function> as the value set for the type element. Part of the furniture hierarchy can be found in Fig. 2. Additional domain knowledge is typically centered around these categories. For indexing purposes the furniture category is crucial because the categorization can be used during retrieval for query generalization (e.g., case → <storage and display furniture>) or query specialization (e.g., case → chest-of-drawers).


Figure 4: AAT categories used for furniture descriptions. All AAT terms in the hierarchy below the category can act as a value for a particular furniture descriptor

Linking to AAT

For nine slots in the furniture template, parts of the AAT hierarchy could be identified as slot value sets. Fig. 4 shows the AAT categories we used. In some cases, multiple parts of AAT act as alternative value sets for a single slot. For example, the AAT categories **neutrals** and **<chromatic colors>** provide the controlled vocabulary for the color slot. Both are subclasses of the AAT category **colors**, to which also a hierarchy of **color types** belongs which do not represent legal values color slot. Fig. 1 shows another example: the slot style/period can be filled with a term from three alternative AAT hierarchies (e.g., **<European styles and periods>**).

What we frequently wanted to do is to specify a class in the AAT hierarchy where all subclasses in this subpart of the hierarchy are possible slot values for a descriptor. Representing this kind of value types is not straightforward with the current (web) representation methods. Protégé allows the specifica-

tion of a "class" as the value type for a slot and asks for one or more superclasses for the allowed class values. However, this information is lost in the translation to RDFS. Property ranges are defined in RDFS through the class of the RDF instance, which in this case is just class **class**. Using the OIL language [5] instead of RDFS would not have helped us here. OIL allows classes as slot-value types, but only when explicitly enumerated in a disjunction:

```
class-def furniture
    slot-constraint color
        has-value (black OR grey OR white OR ...)
```

We could have solved this problem by a different mapping from AAT to RDFS/RDF. Currently, we map all terms in the AAT hierarchy to RDFS classes. One could take the view that leaf terms in AAT should be considered as instances. However, this is not a realistic solution. Often, there are many subtle term specializations in the AAT hierarchy. For example, in the color hierarchy there is a term "pink", which also acts as a superclass for a whole range of pink colors (e.g.., variants of "purplish pink"). Both **pink** and and its specializations should be available as a value for the color descriptor. Even the AAT "guide terms" can be useful as a descriptor value, in situations where an indexer does not know to which subcategory an item belongs.

We finally decided to represent descriptor values as instances of RDFS classes representing AAT concepts. For example, we defined the value of the descriptor color as an *instance of* the AAT categories **neutrals** or **chromatic colors**. This means that an RDF annotation of a piece of furniture cannot have a property "color" with value "pink". Instead, the property value should be some instance of "pink". With "pink" represented by the AAT record **aat:c50124707**, the RDF for the annotation becomes:

```
<rdf:Description about="furniture34">
  <color>
    <aat:c50124707/>
  </color>
</rdf:Description>
```

This expression is the RDF serialization of two relations. The first defines that the property "color" of **furniture34** has the value **pink22** and the second defines that **pink22** is an instance of **aat:c50124707**, a class labeled "pink". In these relations **pink22** is an *anonymous* resource generated by the RDF parser.

From a philosophical point of view something can be said in favor of this representation: "pink" can be considered to be an idealization (in the Platonic sense) of a color, of which the particular color of a piece of furniture is only an approximation. Still, the representation feels somewhat awkward.
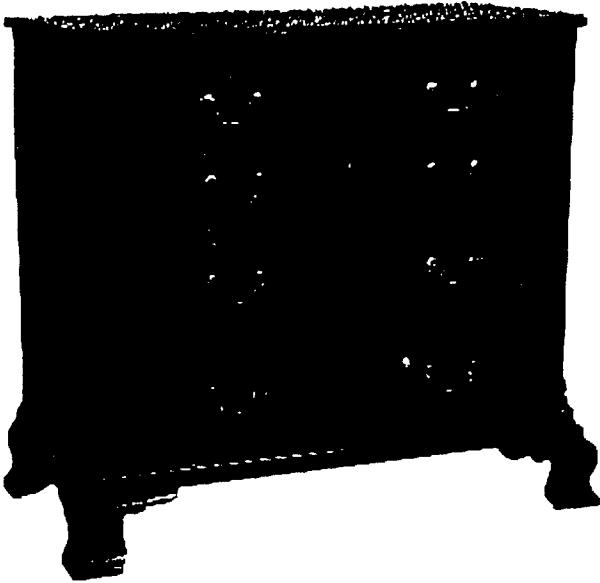
Figure 5: Sample furniture piece: an 18th century chest-of-drawers, Late Georgian style, made of mahogany. "Chest of drawers" is a main term in AAT; the AAT description can be found in the right-hand part of Fig. 2

Adding domain knowledge

In addition to the furniture descriptors and their value sets, there is also a considerable amount of domain knowledge about relationships between descriptor values. To illustrate this we look at an example piece of antique furniture (Fig. 5, taken from [4, p. 28]). The figure shows an 18th-century chest-of-drawers in Late Georgian style (1760–1811), made primarily of mahogany.

Several types of art-historic background knowledge can be distinguished here:

- Knowledge about the relationship between a style period ("Late Georgian") and a time period. Sometimes, the period of a style is dependent on the "culture", e.g. the British Queen Anne style is shorter (1702–1714) than its American pendant (1702–1727).
- Knowledge about the relationship between style periods and furniture characteristics. For example, Late Georgian chests-of-drawers were typically made of mahogany.

This kind of domain knowledge can be extremely useful for supporting both the image indexing and retrieval. During indexing domain knowledge can be used to suggest descriptor values, which puts less burden on the task of the annotator. During retrieval, domain knowledge can be used to make semantic matches, e.g. to retrieve images of Late Georgian chests when a person is looking for "chest mahogany".

However, there are a number of problems in representing this domain knowledge. Firstly, there is no way in RDFS to extend a set of class/property definitions with this kind of inter-property constraints. The same holds for the OIL language.

The OIL slot constraints only apply to a single slot and cannot be used to specify constraints between slots.

Protégé has a constraint language based on KIF and therefore expressing these constraints in Protégé is possible. However, we are then confronted with a second problem. The domain knowledge does not consist of absolute statements about the state of affairs in antique furniture, but provides us mainly with elaborate default knowledge. For example, a Late Georgian chest-of-drawers can be made from oak, but if we have no knowledge to the contrary we can assume it is made from mahogany. This default nature of domain knowledge is also true for time periods of furniture styles, although the period specification (Queen Anne: 1702–1714) may suggest otherwise. The period borders are treated by art historians as indicative only. The semantics of a first-order language are therefore hardly appropriate for expressing the art-historic domain knowledge in this domain. In an earlier case study concerned with indexing photographs of apes [11] we were confronted with similar problems (e.g., urang-utans typically live in Indonesia and have an orange color).

DISCUSSION

One can view this paper as a case study in "real-life" knowledge representation. Many of the issues raised have been discussed and solved in knowledge representation theory. However, in the context of web standards and existing knowledge corpora severe constraints are placed on the representational vehicles. One cannot just redfine the representation of a thesaurus or define a new knowledge-representation standard for the web.

The goal of the Semantic Web initiative is to annotate large amounts of information resources with knowledge-rich metadata. In this paper we have argued that such annotations, in particular of non-textual material such as images, should be based on a rich metadata structure in connection with an ontology. Building ontologies for large domains, such as medicine or arts, is a costly affair. However, in many domains thesauri have been built that can be a basis for the construction of an ontology. A thesaurus should satisfy a number of criteria: it should have a strict sub/superclass hierarchical structure, it should be based on unique concepts rather than on natural-language terms and it should be representable in a format that is compliant with emerging web standards. In the ontology construction process additional knowledge should be added to the basic hierarchical structure of concepts derived from the thesaurus. This knowledge can come from different sources: the location of a concept in the hierarchy, additional sources such as Wordnet, or special purpose documents. Through this process we have created a knowledge base derived from the Art and Architecture Thesaurus (AAT) represented in RDFS. In a case study we have used this ontology as basis for an annotation tool for describing (images of) art objects, in particular antique furniture. The basis of the tool is a metadata structure which is a highly qualified

and extended Dublin Core structure. Each of the descriptor elements could be linked to one or more parts of the AAT, thus providing constraints on the values that can be assigned to the elements. An ever better support for annotation and retrieval can be given when additional constraints are added to the ontology, which essentially consist of complex relations between partial descriptions of objects or images. While the basic metadata knowledge can be represented within the semantic framework of RDFS, the constraint relations require additional representational contructs not available in RDFS and other semantic Web oriented languages, such as OIL. For the time being we have designed a format for representing these constraints that can be used in our own tools, but which is meaningless to the average RDFS application.

Acknowledgments

REFERENCES

1. S. Bechofer and C. Goble. Thesaurus construction through knowledge representation. *Data & Knowledge Engineering*, 37:25-45, 2001.

2. D. Brickley and R. V. Guha. Resource description framework (RDF) schema specification 1.0. Candidate recommendation, W3C Consortium, 27 March 2000. See: http://www.w3.org.

3. Visual Resources Association Standards Committee. VRA core categories, version 3.0. Technical report, Visual Resources Association, July 2000. URL: www.gsd.harvard.edu/ staffaw3/vra/vracore3.htm.

4. R. Davidson. *Miller's Antique Checklist: Furniture*. Reed, London, 1991.

5. D. Fensel, I. Horrocks, F. van Harmelen, S. Decker, M. Erdmann, and M. Klein. OIL in a nutshell. In *Knowledge Engineering and Knowledge Management: 12th International Conference EKAW2000, Juan-les-Pins*, volume 1937 of *Lecture Notes in Artificial Intelligence*, pages 1-16, Berlin/Heidelberg, 2000. Springer-Verlag.

6. N. Fridman Noy, R. W. Fergerson, and M. A. Musen. The knowledge model of Protégé-2000: combining interoperability and flexibility. In *Knowledge Engineering and Knowledge Management: 12th International Conference EKAW2000, Juan-les-Pins*, volume 1937 of *Lecture Notes in Artificial Intelligence*, pages 17-32, Berlin/Heidelberg, 2000. Springer-Verlag. Also as: Technical Report Stanford University, School of Medicine, SMI-2000-0830.

7. Dublin Core Metadata Initiative. *Dublin Core Metadata Element Set Version 1.1: Reference Description*, July 1999. Url: http://dublincore.org/documents/1999/07/02/dces/.

8. J. Martin. *Object-Oriented Methods – A Foundation. UML edition*. Prentice Hall, Upper Saddle River, NJ, 1997.

9. G. Miller. WordNet: A lexical database for english. *Comm. ACM*, 38(11), November 1995.

10. T. Peterson. *Introduction to the Art and Architecture Thesaurus*. Oxford University Press, 1994. See also: http://shiva.pub.getty.edu.

11. A. Th. Schreiber, B. Dubbeldam, J. Wielemaker, and B. J. Wielinga. Ontology-based photo annotation. *IEEE Intelligent Systems*, May/June, 2001.

12. H. van der Waal. ICONCLASS: An inconographic classification system. Technical report, Royal Dutch Academy of Sciences (KNAW), 1985.

13. B. J. Wielinga, J. A. C. Sandberg, and A. Th. Schreiber. Methods and techniques for knowledge management: What has knowledge engineering to offer? *Expert Systems With Applications*, 13(1):73-84, 1997.

Web User Clustering from Access Log
Using Belief Function

Yunjuan Xie
Computer Science Department
Louisiana Tech University
Ruston, LA 71272 USA
yxi001@coes.latech.edu

Vir V. Phoha
Computer Science Department
Louisiana Tech University
Ruston, LA 71272 USA
phoha@coes.latech.edu

Abstract
In this work, we present a novel approach to clustering Web site users into different groups and generating common user profiles. These profiles can be used to make recommendations, personalize Web sites, and for other uses such as targeting users for advertising. By using the concept of mass distribution in Dempster-Shafer's theory, the belief function similarity measure in our algorithm adds to the clustering task the ability to capture the uncertainty among Web user's navigation behavior. Our algorithm is relatively simple to use and gives comparable results to other approaches reported in the literature of web mining.

Keywords
Web mining, clustering, Dempster-Shafer, access log, personalization, common user profile

1   INTRODUCTION
The World Wide Web has become increasingly important as a medium for commerce as well as for dissemination of information. In E-commerce, companies want to analyze the user's preferences to place advertisements, to decide their market strategy, and to provide customized guide to Web customers. In today's information based society, there is an urge for Web surfers to find the needed information from the overwhelming resources on the Internet.

Web access log contains a lot of information that allows us to observe user's interest with the site. Properly exploited, this information can assist us to make improvements to the Web site, create a more effective Web site organization and to help users navigate through enormous Web documents. Therefore, data mining, which is referred to as knowledge discovery in database (KDD), has been naturally introduced to the World Wide Web.

When applied to the World Wide Web, data mining is called Web mining. In [1], Cooley, Mobasher and Strvastava give the taxonomy of Web mining. According to them, there are two major approaches to mining the World Wide Web. The first is Web content mining, which automatically searches the information resources in the Web pages. The other is Web usage mining, which focuses on the discovery of user access patterns from Web usage data. We will focus on Web usage mining.

In this paper, we present a novel algorithm for Web user clustering based on access logs. We propose a distance measure for clustering based on belief function in Dempster-Shafer's theory [5, 12]. By using the Dempster's rule of combining evidence to find and group pages that are frequently visited into different classes, we add to the clustering task the ability to capture the uncertainty among user behavior.

Web usage data collected in access log is at a very fine granularity. It usually includes every HTTP request from all users. Each request contains at least the IP address, requested pages, time requested, response code, and size of the item requested. Therefore, while the access log has the advantage of being extremely detailed, it also has some drawbacks. When we apply statistical and probability methods to it, we tend to get results that are too refined than it should be because the analysis might focus on micro trends rather than macro trends. However, based on our observation, user's browsing behavior on the Web is highly uncertain. Users might browse the same page for different purposes, spend various amounts of time on the same page or make different number of visits on it, or even get to the page from different sources each time. Therefore, micro trends tend to be erroneous and not of much use.

In this paper, we use Dempster-Shafer theory of evidence to model the uncertainty inherent in Web user access patterns. We do not try to assign probabilities for single pages based on the statistics from the Web usage data. Instead, probabilities are assigned to groups of pages based on their co-occurrence in sessions. These groups are refined when evidence accumulates using Dempster's rule of combination. The refined groups of pages are common user profiles we want. This theory is appropriate for clustering analysis because it provides an aggregation operator, Dempster's rule for combining evidence, which allows the expression of the uncertainty with respect to aggregated components. Fur-

thermore, the set operation in Dempster's rule is particularly suitable for clustering pages into groups.

We first introduce the related work in clustering Web usage data in section 2. In section 3, we give a brief description of Dempster-shafer's theory. In section 4, we explain the selection of session as the unit of mining. And in section 5, we propose an algorithm using Dempster-shafer's theory for usage data clustering analysis. After that in section 6 we give an example of clustering a sample Web site and show preliminary experimental results using the clustering analysis algorithm. We conclude our work in section 7.

2   RELATED WORK

Data mining, which is referred to as knowledge discovery in database, has become an important research area as a consequence of the maturity of very large databases. It uses techniques from areas such as machine learning, statistics, neural networks, and genetic algorithms to extract implicit information from very large amounts of data. The goals of data mining are prediction, identification, classification, and optimization. The knowledge discovered by data mining includes association rules, sequential patterns, clusters, and classification. Garofalakis [6] gives a review of popular data mining techniques and the algorithms for discovering the Web. Cooley et al [1] proposes a taxonomy of Web mining and identified further research issues in this field. Yu [15] examines new developments in data mining and its application to personalization in E-commerce.

Various data mining techniques have been successfully applied to Web access logs to extract useful information [4, 8, 9, 10, 11, 13, 14]. Among them, clustering allows us to group together clients or data items that have similar characteristics. The information discovered by this technique is one of the most important types that has a wide range of applications from real-time personalization to link prediction. It can facilitate the development of future marketing strategies, such as automated return mail, present advertisements to clients falling within a certain cluster, or dynamically changing a particular site for a client on a return visit based on past classification of that client. The key problem lies in how we effectively discover clusters of Web pages or users with common interest.

Clustering analysis to mine the Web is quite different from traditional clustering due to the inherent difference between Web usage data clustering and classic clustering. Therefore, there is a need to develop specialized techniques for clustering analysis based on Web usage data. Some approaches to clustering analysis have been developed for mining the Web access logs.

Perkowitz and Etzioni [10] discuss adaptive Web sites that learn from user access patterns. The PageGather [10] algorithm uses the page co-occurrence frequencies to find clusters of related but unlinked pages. It creates a graph whose nodes are pages and whose edge weights are page co-occurrence frequencies. Clusters are found by finding the cliques or connected components in this graph. Based on the algorithm, new index pages are created for easier navigation.

Mobasher, Cooley and Srivastava [8] propose a technique for capturing common user profiles based on association-rule discovery and usage-based clustering. This technique directly computes overlapping clusters of URL references based on their co-occurrence patterns across user transactions. A *hypergraph* is built whose hyperedges are frequent itemsets that are found by the *a priori* algorithm. The weight of a hyperedge is calculated by averaging all the confidences of association rules in this frequent itemset. Clusters are obtained by applying the hypergraph partitioning algorithm to this hypergraph.

Nasraoui et al [9] defines a similarity measure between sessions using a modified cosine angle similarity measure that takes the hierarchical structure of URL into consideration. Then sessions are clustered by a Relational Fuzzy C-Maximal Density Estimator (RFC-MDE) algorithm based on pair-wise dissimilarities between sessions.

All these approaches find session clusters from all user sessions. These approaches tend to find the frequent user access pattern of all users. Our approach differs from these clustering algorithms in that it finds user clusters. It separates users into different groups and finds a common access pattern for each group of users. To our knowledge, user clustering has not been studied in the Web usage mining field.

3   BACKGROUND

3.1   Dempster-Shafer's Theory

Dempster-Shafer's theory [6, 12] of combining evidence has attracted considerable attention as a promising method for dealing with some problems arising in combining of evidence and data fusion. It starts by assuming a Universe of Discourse U, also called Frame of Discernment, which is a set of mutually exclusive alternatives. The frame of discernment can consist of the possible values of an attribute.

It gives to each subset A of U a **basic probability assignment** (*bpa*) $m(A)$, which represents the strength of some evidence. For the empty set, m is 0; the sum of m over all subsets of U is 1. That is:

$$m(\phi) = 0 \quad \text{and} \quad \sum_{A_i \subseteq U} m(A_i) = 1.$$

The basic probability assignment m is referred to as *mass distribution* to distinguish it from the probability distribution. Note that it applies directly to the evidence (subsets of the frame of discernment U), not to the elements of U as in traditional probability theory.

A belief Bel(S) summarizes all our reasons to believe S:

$$Bel(S) = \sum_{A_i \subseteq S} m(A_i). \tag{1}$$

203

3.2 Dempster's Rule of Combination

Dempster-Shafer's theory provides a means for combining beliefs from distinct sources, known as Dempster's rule of combination. Suppose m_1 and m_2 are two *bpa*'s of the same U from independent bodies of evidence, A_i and B_j. The combined *bpa* can be computed as:

$$m_1 \oplus m_2(C) = \frac{\sum_{A_i \cap B_j = C} m_1(A_i)m_2(B_j)}{1 - \sum_{A_i \cap B_j = \phi} m_1(A_i)m_2(B_j)}, \quad for$$

all non-empty C, \hfill (2)

where $1 - \sum_{A_i \cap B_j = \phi} m_1(A_i)m_2(B_j)$ is a normalization factor, making the sum of $m_1 \oplus m_2(C)$ between 0 and 1.

In words, the Dempster's combination rule computes a measure of agreement between two bodies of evidence concerning different propositions discerned from a common frame of discernment. The rule focuses only on those propositions that both bodies of evidence support.

If $1 - \sum_{A_i \cap B_j = \phi} m_1(A_i)m_2(B_j) = 0$, we say that m_1 and m_2 are incompatible, and $m_1 \oplus m_2$ is undefined.

4 SELECTION OF SESSION AS THE UNIT OF MINING

Session, by definition, is the set of pages visited by a user in one single visit. It is the basic unit in Web mining. For Web data mining tasks, session is too coarse grained. Users may perform multiple tasks in one single session. Based on this observation, researchers developed specialized algorithms to refine single user session into smaller units. Among them, Mobasher *et al* [2] separates the pages for content purpose from those for navigation purpose based on the time spent on the page and divides user sessions into semantically meaningful units, called transactions. The authors define two types of transactions, auxiliary-content transactions and content-only transactions. Chen, Park and Yu [4] propose the maximal forward reference identification. The former group of authors assumes that users travel through auxiliary pages to get to content pages. And the latter assumes that all the backward references are made for ease of traveling but not for browsing. Both of them obtain reasonable results based on these assumptions.

User navigation behavior is highly uncertain. Assumptions about user access patterns should be made with utmost care. Poor assumptions make the goal to find the common user traversal pattern even more difficult. Based on our observation, the user may perform one or multiple *tasks* in one single session. For example, a user may go through both the financial part and the sports part of a news site in one session. This is a two-task session. To perform one task, the user needs to access a group of pages. So the pages needed

in one task tend to appear together. However, in contrast to trying to explicitly divide sessions into tasks based on some kind of assumption that does not accurately describe all the user's Web activities, in our approach, we directly overlap clusters of URLs based on their co-occurrence patterns in one session. As the process moves on, some tasks are separated from sessions. The clusters obtained this way tend to group related pages together across tasks to show a co-occurrence pattern of a particular type of users, even though these tasks are themselves not deemed to be similar. This allows us to obtain clusters that potentially capture overlapping interests of the same type of users.

Given a large access log, our goal is to cluster Web site users into groups and find groups of pages that tend to co-occur in visits by a certain type of user. Standard clustering algorithms partition Web pages into a set of mutually exclusive clusters. Whereas traditional clustering is concerned with placing each page in exactly one cluster, ours may place a single page in multiple overlapping clusters. Instead of attempting to partition the entire log file into disjoint clusters, the algorithm finds a small number of, possibly overlapping, clusters. It can discover multiple interests of the same type of user and group users into different types.

5 OUR APPROACH: CLUSTERING AND COMMON USER PROFILE ANALYSIS USING DEMPSTER-SHAFER'S THEORY

5.1 Extracting Content Pages from Access Log

A critical step in effective Web mining is the data preprocess. It includes access log cleaning, session identification, and transformation of access log data to an appropriate format, according to the need of the mining analysis. In [3], the authors give a detailed summary of data preparation work for mining the World Wide Web. Generally, data preparation needs to meet the requirements of the particular mining task. For our clustering analysis, data preprocessing contains three steps: access log cleaning, session identification, and low support page filtering.

5.1.1 Access Log Cleaning

Redundant references (images, sound files, multiple frames, and dynamic pages that have the same template) are removed in this step, leaving only one entry per page request. We eliminate the irrelevant items by checking the suffix of the URL requests. All log entries with filename suffixes such as *gif, jpeg, jpg,* and *map* are removed.

5.1.2 Session Identification

Session identification identifies a set of user sessions by a maximal elapsed time. If the time between page requests exceeds a certain limit, we assume that the user is starting a new session. Here, like many commercial products, we use 30 minutes as a default timeout.

5.1.3 Low Support Page Filtering

For the purposes of clustering analysis, only the content pages are of interest. The pages used just to facilitate the navigation are referred to as auxiliary pages. Filtering the session files to remove these auxiliary pages is necessary to remove noise from important data. Whether a page should be classified as an auxiliary page or a content page for that user is based on the time the user spends on that page. It is expected that the variance of the time spent on auxiliary pages is small. However, the length of time spent on content pages is expected to vary widely from user to user. Generally, the time spent on a content page is much longer than on an auxiliary page. In [2], the authors find that the distribution of time and hits on Web page requests contains a large exponential component. By evaluating the percentage of auxiliary pages in a particular Web site, they calculate a cutoff time between content pages and auxiliary pages based on the exponential distribution. In contrast, we sort all of the lengths from the log and *then* find the cutoff time between auxiliary pages and content pages. All auxiliary pages in sessions are filtered out. Now what remains in the sessions are pure content pages. (If the same page appears more than one time, the time spent on it is summed up).

However there are also content pages with very low hit rates in the session. These pages represent only the personal interest of individual users. They are also to be filtered out since what we are interested in is the interests of a group of users. These pages should be filtered out based on the average number of hits of content pages.

5.2 Basic Probability Assignment for Each User

Basic probability assignment (bpa) is assigned to each user. After data preprocessing, we find that some sessions from the same user can overlap because a user may perform the same task in different sessions. A probability is assigned to each unique session after data preprocessing; it is the fraction of this unique session to the total number of user sessions. This probability measures how likely the user will perform the tasks identified in the unique session. The total probability has measure one. It gives a big picture of what the user usually does, as well as how often she does it in the site. This assignment is reasonable since it captures the uncertainty among visits to single pages. In our observation, session by itself is a semantically meaningful unit. It represents one or several tasks users tend to perform in one visit. Users usually need to browse a group of pages, rather than a single page, to accomplish one task. Therefore, assigning a probability to a group of pages seems to fit perfectly the semantic meaning of session.

5.3 Common User Profile Clustering Algorithm

5.3.1 Belief Function as Similarity Measure

Clustering, by definition, is to partition data points into clusters, so that the data points within one cluster are more similar to each other than data points in different clusters.

Therefore, some similarity measure should be adopted in every clustering algorithm. For our Web user clustering algorithm, we propose to use belief function as the similarity measure.

Suppose $m(A_i)$, $m(B_j)$ are two **bpa**s for two users, A and B. We also use A, B to represent the set of unique content pages in the user's profile, respectively. We define $bel(A)$ as the total belief that user A's profile can represent user B's profile:

$$bel(A) = \sum_{B_i \subseteq A} m(B_i).$$

In some cases, if B is contained in A, $bel(A) = 1$. However, the reverse is not true. So we define the *similarity* between A and B as:

$$sim(A, B) = \min(\sum_{A_i \subseteq B} m(A_i), \sum_{B_i \subseteq A} m(B_i)). \quad (3)$$

It measures the similarity between two user profiles.

5.3.2 Greedy Clustering Using Belief Function (GCB)

We present a GCB algorithm for our clustering task using the similarity measure we defined above. The greedy technique has been widely used in many algorithms as an efficient and effective way to approach a goal. In this process representatives of the clusters are picked iteratively, so that the current representative is well separated from those that have been chosen so far. An outline of the algorithm follows:

Input: K: number of cluster; S: a simple set of users
Output: M: the set of cluster representatives
begin
    M = {ø}
    //select a random user m₁ into the common profile set
    M = {m₁}
    For each user profile x ∈ S − M, calculate the distance
between x and m₁
    Dist(x) = -ln(sim(x, m₁))
    For i = 2 to K
      begin
        //choose representative mᵢ to be far from previous
representatives
        Let mᵢ ∈ S − M, such that dist(mᵢ) = max(dist(x)|x∈
S − M)
        M = M∪{mᵢ}
        //Update the similarity of each point to the closest
representative
        for each x ∈ S − M
          dist(x) = min(dist(x), -ln(sim(x, mᵢ)))
      end
    return M   *//M will contain a set of distinct cluster*
representatives
end

5.3.3 Common User Profile Creation

A user is assigned to the cluster whose representative is most similar to this user based on the similarity measure. After we assign each user to different groups, we apply Dempster's rule of combination to get the common user profiles.

$$m_1 \oplus m_2 \dots\dots \oplus m_n$$

In the definition, if $1 - \sum_{A_i \cap B_j = \phi} m_1(A_i) m_2(B_j) = 0$, m_1

and m_2 are said to be incompatible, and $m_1 \oplus m_2$ is undefined. Here in our application, we need to restrict the condition to get higher quality clusters. So we define two *bpa* as incompatible if one subset of one *bpa* has an empty intersection with any subset of the other *bpa*. In this case, these two *bpa*s should belong to two different profiles. The probability value for an empty set may get larger after iterations. Normalization is used to eliminate the empty set. The probability portion for the empty set is subtracted and the probability distribution is recalculated, so that the total measure is one. After iterations, the sets in the common profile become separated and stable. Thus we get groups of pages in each common user profile. It is expected that most of these groups will represent a single task and some of them may contain multiple tasks. Association rules can be found in the co-occurrences of multiple tasks.

6 ILLURSTRATION AND EXPERIMENTAL RESULTS

6.1 Illustration

In our work, the frame of discernment U contains all the possible (and mutually exclusive) values (URLs). An initial belief function is defined over U describing our prior information about one user. As different users continue to enter, different user evidence is then converted to corresponding belief functions over U. These belief functions are combined together through Dempster's rule, to give rise to a consensus answer to a common user profile.

In real-time recommendation, our task is to match the user's active session with the set of common user profiles obtained from the system over time. If S is the set of pages of the user's active session, then the belief that the user belongs to the group is:

$$Bel(S) = \sum_{A \subseteq S} m(A).$$

Bel(S) calculates the similarity between a user and the common user profiles. The greater the belief value, the higher the similarity. Recommendations are made based on the common user profile that matches the user with the highest value.

The following figure illustrates a sample Web site. In this figure, A, B, C, D are auxiliary pages and filtered out in data preprocessing. And F, G, H, E, J, P are content pages.
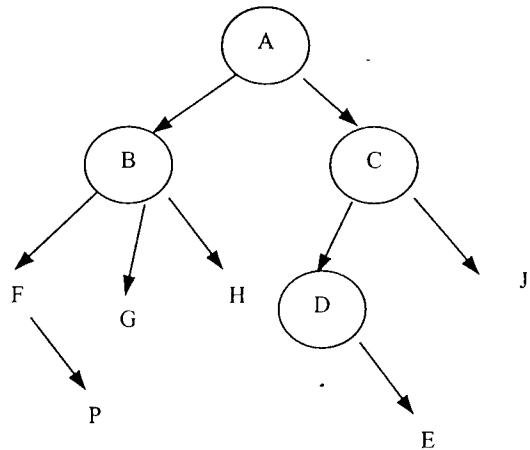

Figure 1. Sample Web site

Table 1 shows the profiles for thirteen users. Each includes a basic probability assignment (bpa) and a unique content page set. For example, for user 1, session (F, G) has a bpa of 2/5, session (F, P) has a bpa of 2/5, and session (J) has a bpa of 1/5--the total bpa will sum up to one. And the unique content page set contains the unique pages from all sessions of the user, in this case, F, G, P, and J. For each user profile, we use the following notation below:

User profile: {(session$_1$): bpa$_1$, (session$_2$): bpa$_2$, ..., (session$_n$): bpa$_n$}.

Table 1: Sample user profiles

User	Profile	Unique pages
1	{(F,G):2/5, (F, P):2/5, (J):1/5}	(F, G, P, J)
2	{(F, H):1/2, (G):1/2}	(F, H, G)
3	{(F, P): 3/3}	(F, P)
4	{(G, J):1/2, (F):1/2}	(F, G, J)
5	{(G,H):2/3, (F,J):1/3}	(G, H, F, J)
6	{(F,H):1/3, (G, H):1/3, (F,G):1/3}	(F, H, G)
7	{(G,H):1/2, (J):1/2}	(G, H, J)
8	{(F,P,G):1/3, (J):2/3}	(F, P, G, J)
9	{(F,G):3/3}	(F, G)
10	{(G):1/4, (H,J):3/4}	(G, H, J)
11	{(E, J):4/4}	(E, J)
12	{(G,H,F,P):1/2, (E,J):1/2}	(G, H, F, P, E, J)
13	{(F,P,G):1/4, (E):3/4}	(F, P, G, E)

Table 2 shows the clustering analysis results obtained on the sample user profiles in Table 1 with the number of clusters, the input parameter, equaling 5. It includes three steps:

206

1. Find the cluster representatives by applying the GCB algorithm. The results are shown in the "Representative" column.

2. Assign the rest of the users to the closest cluster based on the similarity measure we defined in section 5.3.1. The results are shown in the "Members" column.

3. Create common user profiles by applying Dempster's rule of combination to each cluster. The results are shown in the "Common user profile" column.

Table 2: Clustering result from sample user profile (K = 5)

Cluster	Representative	Members	Common user profile
1	User 1: {(F,G), (F,P), (J)}	User 7: {(F,P,G), (J)} User 12: {(G,H,F,P), (E,J)} User 13: {(F,P,G), (E)}	{(F,G), (F,P)}
2	User 2: {(F,H), (G)}	User 5: {(G,H), (F,J)} User 6: {(F,H), (G,H), (F, G)} User 9: {(F, G)}	{(F), (G)}
3	User 3: {(F,P)}	____	____
4	User 4: {(G,J), (F)}	____	____
5	User 7: {(G,H), (J)}	User 10: {(G), (H,J)} User 11: {(E, J)}	{(J)}

From Table 2, we see five users are first selected as representatives for five clusters. Cluster 1 initially has User 1 as the representative and ends up with Users 7, 12, and 13 as members after step 2. The common user profile for Cluster 1 is {(F, G), (F, P)}. Cluster 2 initially has User 2 as the representative and ends up with Users 5, 6, and 9 as members. The common user profile for Cluster 2 is {(F), (G)}. Cluster 5 initially has User 7 as the representative and ends up with User 10 as a member. The common user profile for

Cluster 5 is {(J)}. Cluster 3 and 4 do not have members in this case.

6.2 Preliminary Experimental Results

We apply the GCB algorithm to the access log of the Web site of the Boston University Computer Science department. It was collected by the Oceans Research Group at Boston University. It contains a total of 1,143,839 requests, representing a population of 762 different users. A portion of the access log file is available at http://ita.ee.lbl.gov/html/contrib/BU-Web-Client.html. After access log cleaning, we get 667 sessions from a total of 50 users. Users are assigned an identification number from 1 to 50. We choose a clustering factor of 5 because the amount of data is small.

Table 3 shows five clusters we obtained. Not all users are assigned to these clusters because some users do not belong to any of them due to the sparseness of the data.

Table 3: Clusters results from BU log (k = 5)

Cluster	Members	Common user profile
1	1, 3, 4, 13, 14, 39, 43, 46	{(/cs-www.bu.edu/), (cs101a1/Home.html), (cs101b1/Home.html) }
2	2, 6, 7, 10, 11, 27, 28, 29, 32, 49	{(/cs-www.bu.edu/), (/faculty/heddaya/CS792/schedule.html), (/faculty/kfoury/CS530/home.html) }
3	11, 8, 26	{/cs-www.bu.edu/, /faculty/kfoury/CS520/grades.text /students/grads/ianw/cs530.html}
4	33, 41, 44	{/cs-www.bu.edu/ //faculty/mcchen/cs320/Home.html }
5	36, 17, 48, 9	{/cs-www.bu.edu/, /pointers/Home.html /students/grads/Home.html }

The results show that:

- Different groups of users can be identified by the common courses they selected, such as cluster 1, 2, 3, 4.

- Some groups of users are discerned by their common interest, such as cluster 5.

- /cs-www.bu.edu/ appears in every profile because as the entry page to the site, it has both high hit rate and long viewing time.

207

7 CONCLUSION AND FUTURE WORK

The ability to mine Web usage data provides E-commerce companies with a great opportunity for personalizing their Web site appearance to customers. Most current approaches to personalization rely heavily on human participation to collect profile information about users. Such practice suffers from problems of being subjective, as well as getting out of date as the user preferences change over time. In this paper, we present an automatic Web personalization method and introduce an effective clustering technique using belief function based on Dempster-Shafer's theory.

This work still has several research issues, which we plan to address in the future. First, usage data by itself is not sufficient for recommendation. The personalization and recommendation process needs to have specific knowledge about the particular domain to do anything besides filtering based on statistical attributes of the discovered rules or patterns.

Another problem is the scalability problem. Usage data collection on the Web is incremental. Hence, there is a need for mining algorithms to be scalable. They should be able to take as input the existing data, and mined knowledge, as well as the new data, and develop a new model in an efficient manner. Our future work will address these problems.

REFERENCES

[1] Cooley, R., Mobasher, B., and Srivastava, J. Web Mining: Information and Pattern Discovery on the World Wide Web, *ICTAI'97,* 1997

[2] Cooley, R., Mobasher, B., and Srivastava, J. Grouping Web page references into transaction for mining world wide Web browsing patterns, *Proceedings of the 1997 IEEE Knowledge and Data Engineering Exchange Workshop (KDEX '97),* 1997

[3] Cooley, R., Mobasher, B., and Srivastava, J. Data preparation for mining world Wide Web browsing patterns, *Journal of Knowledge and Information Systems,* (1) 1, 1999

[4] Chen, M. S., Park, J. S., and Yu, P. S. Efficient Data Mining for Path Traversal Patterns, *IEEE Transactions on Knowledge and Data Engineering,* Vol. 10, No. 2, March/April, 1998

[5] Dempster, A. P. A Generalization of Bayesian Inference J. Roy. Stat. Soc. B, 30(1968), 205-247

[6] Garofalakis, M.N., Rastogi, R., Seshadri, S., and Shim, K. Data mining and the Web: past, present and future In *Proceedings of the second international workshop on Web information and data management,* ACM 1999

[7] Lalmas, M. Dempster-Shafer's Theory of Evidence applied to Structured Documents: modeling Uncertainty, *SIGIR97,* Philadelphia, USA, 1997.

[8] Mobasher, B., Cooley R., and Srivastava, J. Creating Adaptive Web Sites Through Usage-based Clustering of URLs, *Proceedings of the 1999 Workshop on Knowledge and Data Engineering Exchange,* 1999

[9] Nasraoui, O., Frigui, H., Joshi, A., and Krishnapuram, R. Mining Web access log using relational competitive fuzzy clustering *Proceedings of the Eight International Fuzzy Systems Association World Congress,* 1999

[10] Perkowitz, M., Etzioni, O. Adaptive Web sites: automatically synthesizing Web pages in *Proceedings of Fifteenth National Conference on Artificial Intelligence,* Madison, WI, 1998

[11] Schechter, S., Krishnan, M., and Smith, M.D. Using path profiles to predict HTTP requests. In *Proceedings of the Seventh International World Wide Web Conference, Brisbane, Australia, 1998*

[12] Shafer, G. A Mathematical Theory of Evidence, *Princeton: Princeton University Press,* 1976

[13] Shahabi, C. Zarkesh, A. M., Adibi, J., Shah, V. Knowledge Discovery from Users Web-Page Navigation, *Proceeding of Workshop on Research Issues in Data Engineering,* Birmingham, England, IEEE, 1997

[14] Spiliopoulou, M. and Faulstich, L. C. WUM: A Web Utilization Miner. In *Proceedings of EDBT Workshop WebDB98,* Valencia, Spain, LNCS 1590, Springer Verlag, 1999

[15] Yu, P. S. Data Mining and Personalization Technologies, Proceedings of the 6th International Conference on Database Systems for Advanced Applications, 1998

[16] Zadeh, L. A. A simple view of the Dempster-Shafer theory of evidence and its implication, *AI Magazine* 7,2 (1986), 85-90 1986

Author Index