

Wiley Series in Systems Engineering and Management • Andrew P. Sage, Series Editor

AGENT-DIRECTED SIMULATION AND SYSTEMS ENGINEERING

Edited by

LEVENT YILMAZ AND TUNCER ÖREN



Levent Yilmaz and Tuncer Ören

Agent-Directed Simulation and Systems Engineering



WILEY-VCH Verlag GmbH & Co. KGaA

**Agent-Directed Simulation and
Systems Engineering**

Edited by
Levent Yilmaz and Tuncer Ören

Wiley Series in Systems Engineering and Management

Series Editor: Andrew P. Sage

A complete list of the titles in this series appears at the end of this volume.

Levent Yilmaz and Tuncer Ören

Agent-Directed Simulation and Systems Engineering



WILEY-VCH Verlag GmbH & Co. KGaA

The Editors

Prof. Levent Yilmaz

Auburn University
3116 Shelby Center
Computer Science and Software Engineering
College of Engineering
Auburn, AL 36849
USA

Prof. Tuncer Ören

University of Ottawa
Faculty of Engineering
800 King Edward
Ottawa, ON K1N 6N5
Canada

Cover

Spieszdesign,
Neu-Ulm, Germany.

All books published by Wiley-VCH are carefully produced. Nevertheless, authors, editors, and publisher do not warrant the information contained in these books, including this book, to be free of errors. Readers are advised to keep in mind that statements, data, illustrations, procedural details or other items may inadvertently be inaccurate.

Library of Congress Card No.: applied for
British Library Cataloguing-in-Publication

Data: A catalogue record for this book is available from the British Library.

**Bibliographic information published
by the Deutsche Nationalbibliothek**

The Deutsche Nationalbibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data are available on the Internet at <<http://dnb.d-nb.de>>.

© 2009 WILEY-VCH Verlag GmbH & Co. KGaA, Weinheim

All rights reserved (including those of translation into other languages).
No part of this book may be reproduced in any form – by photostriking, microfilm, or any other means – nor transmitted or translated into a machine language without written permission from the publishers. Registered names, trademarks, etc. used in this book, even when not specifically marked as such, are not to be considered unprotected by law.

Printed in the Federal Republic
of Germany

Printed on acid-free paper

Typesetting le-tex publishing services
GmbH, Leipzig

Printing betz-druck GmbH, Darmstadt

Binding Litges & Dopf GmbH,
Heppenheim

Cover Design Spieszdesign, Neu-Ulm

ISBN 978-3-527-40781-1

To Funda
Levent Yilmaz

To Füsun
Tuncer Ören

Foreword

What makes agent software different from ordinary software? What can modeling and simulation contribute to agent systems? Conversely, is there a role for agents in modeling and simulation? This book is the first extended work to provide in-depth answers to these questions and others like them. Indeed, the term “agent” has become a ubiquitous “buzzword” used in an enormous variety of contexts to refer to a wide range of software attributes. So it is very timely to bring together a group of experts to offer their unique insights into various aspects of agent software as they relate primarily to modeling and simulation and to systems engineering. However, this book is more than a collection of essays on agents in their diverse applications. The editors, Tuncer Ören and Levent Yilmaz, combining depth of experience with bleeding-edge enthusiasm, provide a degree of coherence well beyond that usually seen in edited collections. In addition to writing some of the key chapters, the editors contribute new concepts and organizing principles that illuminate the current state of the art and reveal intriguing possibilities for theory and applications going forward.

The looseness of terminology in information technology is something we have grown to live with as the field has developed. Terms such as “agent” that have broad connotations are useful for such socially important activities as building communities of interest, organizing conferences, and successfully communicating with the nontechnical layperson. However, there comes a point where imprecise terminologies need to be given greater definitiveness so that critical concepts can be delineated and clarified, thereby allowing the field to move forward on a sounder foundation. To address this need, the book opens with a chapter that provides an integrative and comprehensive view of modeling and simulation, laying the basis for the rest of the book. Interspersed in the sequel are chapters that provide similarly foundational discussions of agent concepts, systems engineering, and the application of each one of these areas to the other. Whether you are a software developer, simulation practitioner, or systems engineer, you will find some eye-opening material in this presentation.

The central thesis of the book is that, while simulation in application to agents is fairly well established, the converse application of agents to the enterprise of modeling and simulation is much less appreciated and, as the editors assert, no less important. Indeed, Ören’s taxonomy in the opening chapter alone is worth the

price of admission. His framework enables one to consider the mutual synergies among modeling and simulation, system theories, systems engineering, software agents, and artificial intelligence. I can envision research professors and graduate students being stimulated to explore branches of this tree toward new research directions in proposals or dissertations.

As a contributor of a chapter to the book, I can attest that its title, “Agent-Directed Simulation and Systems Engineering”, challenged me to address its novel theme in the context of my own and fellow authors’ work. Interoperability has become a critical feature in modern systems of systems engineering. Testing for interoperability (the ability of independent systems to effectively communicate) has likewise become a necessary part of the overall system development process. Our work in this context employs foundational modeling and simulation concepts in novel ways. We use models derived from system requirements to execute in real time as agents deployed across a network to observe net-enabled collaboration of participants. In this way, agent, modeling and simulation, and systems engineering, concepts, and technologies are brought together to provide a testing solution that would be difficult to synthesize otherwise.

We used the term “agent-implemented” in our title (Chapter 13) to suggest the particular flavor of support provided by agents in this context. By all means, let the reader be challenged to find new species of agent-directed, enabled, oriented, or driven simulation to further the goals of model-based systems engineering. And conversely!

December 2008

Bernard P. Zeigler

Contents

Foreword	VII
Preface	XIX
List of Contributors	XXIII
Part One Background	1
1 Modeling and Simulation: a Comprehensive and Integrative View	3
<i>Tuncer I. Ören</i>	
1.1 Introduction	3
1.2 Simulation: Several Perspectives	4
1.2.1 Purpose of Use	4
1.2.2 Problem to Be Solved	8
1.2.3 Connectivity of Operations	9
1.2.4 M&S as a Type of Knowledge Processing	9
1.2.5 M&S from the Perspective of Philosophy of Science	13
1.3 Model-Based Activities	13
1.3.1 Model Building	15
1.3.2 Model-Base Management	15
1.3.3 Model Processing	15
1.3.4 Behavior Generation	17
1.4 Synergies of M&S: Mutual and Higher-Order Contributions	20
1.5 Advancement of M&S	20
1.6 Preeminence of M&S	24
1.6.1 Physical Tools	27
1.6.2 Knowledge-Based or Soft Tools	27
1.6.3 Knowledge Generation Tools	30
1.7 Summary and Conclusions	32
2 Autonomic Introspective Simulation Systems	37
<i>Levent Yilmaz and Bradley Mitchell</i>	
2.1 Introduction	37
2.2 Perspective and Background on Autonomic Systems	39
2.3 Decentralized Autonomic Simulation Systems: Prospects and Issues	41

2.3.1	Motivating Scenario: Adaptive Experience Management in Distributed Mission Training	41
2.3.2	An Architectural Framework for Decentralized Autonomic Simulation Systems	42
2.3.3	Challenges and Issues	44
2.4	Symbiotic Adaptive Multisimulation: An Autonomic Simulation System	47
2.4.1	Metamodels for Introspection Layer Design	48
2.4.2	Local Adaptation: First-Order Change via Particle Swarm Optimizer	50
2.4.3	The Learning Layer: Genetic Search of Potential System Configurations	51
2.4.4	SAMS Component Architecture	52
2.5	Case Study: UAV Search and Attack Scenario	55
2.5.1	Input Factors	56
2.5.2	Agent Specifications	57
2.6	Validation and Preliminary Experimentation with SAMS	64
2.6.1	Face Validity of the UAV Model	65
2.6.2	Experiments with the Parallel SAMS Application	67
2.7	Summary	70

Part Two Agents and Modeling and Simulation 73

3	Agents: Agenthood, Agent Architectures, and Agent Taxonomies	75
	<i>Andreas Tolk and Adelinde M. Uhrmacher</i>	
3.1	Introduction	75
3.2	Agenthood	76
3.2.1	Defining Agents	76
3.2.2	Situated Environment and Agent Society	78
3.3	Agent Architectures	79
3.3.1	Realizing Situatedness	79
3.3.2	Realizing Autonomy	81
3.3.3	Realizing Flexibility	82
3.3.4	Architectures and Characteristics	84
3.4	Agenthood Implications for Practical Applications	86
3.4.1	Systems Engineering, Simulation, and Agents	87
3.4.2	Modeling and Simulating Human Behavior for Systems Engineering	88
3.4.3	Simulation-Based Testing in Systems Engineering	91
3.4.4	Simulation as Support for Decision Making in Systems Engineering	93
3.4.5	Implications for Modeling and Simulation Methods	94
3.5	Agent Taxonomies	96
3.5.1	History and Application-Specific Taxonomies	96
3.5.2	Categorizing the Agent Space	99
3.6	Concluding Discussion	101

4	Agent-directed Simulation	111
	<i>Levent Yilmaz and Tuncer I. Ören</i>	
4.1	Introduction	111
4.2	Background	113
4.2.1	Software Agents	113
4.2.2	Complexity	113
4.2.3	Complex Systems of Systems	114
4.2.4	Software Agents within the Spectrum of Computational Paradigms	115
4.3	Categorizing the Use of Agents in Simulation	118
4.3.1	Agent Simulation	118
4.3.2	Agent-Based Simulation	119
4.3.3	Agent-Supported Simulation	119
4.4	Agent Simulation	120
4.4.1	A Metamodel for Agent System Models	120
4.4.2	A Taxonomy for Modeling Agent System Models	122
4.4.3	Using Agents as Model Design Metaphors: Agent-Based Modeling	123
4.4.4	Simulation of Agent Systems	127
4.5	Agent-Based Simulation	129
4.5.1	Autonomic Introspective Simulation	130
4.5.2	Agent-Coordinated Simulator for Exploratory Multisimulation	131
4.6	Agent-Supported Simulation	134
4.6.1	Agent-Mediated Interoperation of Simulations	135
4.6.2	Agent-Supported Simulation for Decision Support	139
4.7	Summary	141

Part Three Systems Engineering and Quality Assurance for Agent-Directed Simulation 145

5	Systems Engineering: Basic Concepts and Life Cycle	147
	<i>Steven M. Biemer and Andrew P. Sage</i>	
5.1	Introduction	147
5.2	Agent-Based Systems Engineering	148
5.3	Systems Engineering Definition and Attributes	148
5.3.1	Knowledge	149
5.3.2	People and Information Management	150
5.3.3	Processes	151
5.3.4	Methods and Tools	156
5.3.5	The Need for Systems Engineering	157
5.4	The System Life Cycle	157
5.4.1	Conceptual Design (Requirements Analysis)	160
5.4.2	Preliminary Design (Systems Architecting)	161
5.4.3	Detailed Design and Development	161
5.4.4	Production and Construction	163
5.4.5	Operational Use and System Support	164
5.5	Key Concepts of Systems Engineering	164
5.5.1	Integrating Perspectives into the Whole	164

5.5.2	Risk Management	165
5.5.3	Decisions and Trade Studies (the Strength of Alternatives)	166
5.5.4	Modeling and Evaluating the System	168
5.6	Summary	169
6	Quality Assurance of Simulation Studies of Complex Networked Agent Systems	<i>173</i>
	<i>Osman Balci, William F. Ormsby, and Levent Yilmaz</i>	
6.1	Introduction	173
6.2	Characteristics of Open Agent Systems	174
6.3	Issues in the Quality Assurance of Agent Simulations	175
6.4	Large-Scale Open Complex Systems – The Network-Centric System Metaphor	177
6.5	M&S Challenges for Large-Scale Open Complex Systems	179
6.6	Quality Assessment of Simulations of Large-Scale Open Systems	181
6.7	Conclusions	186
7	Failure Avoidance in Agent-directed Simulation: Beyond Conventional v&v and qa	<i>189</i>
	<i>Tuncer I. Ören and Levent Yilmaz</i>	
7.1	Introduction	189
7.1.1	The Need for a Fresh Look	189
7.1.2	Basic Terms	191
7.2	What Can Go Wrong	192
7.2.1	Increasing Importance of M&S	192
7.2.2	Contributions of Simulation to Failure Avoidance	192
7.2.3	Need for Failure Avoidance in Simulation Studies	194
7.2.4	Some Sources of Failure in M&S	196
7.3	Assessment for M&S	198
7.3.1	Types of Assessment	198
7.3.2	Criteria for Assessment	200
7.3.3	Elements of M&S to be Studied	200
7.4	Need for Multiparadigm Approach for Successful M&S Projects	200
7.4.1	V&V Paradigm for Successful M&S Projects	201
7.4.2	QA Paradigm for Successful M&S Projects	203
7.4.3	Failure Avoidance Paradigm for Successful M&S Projects	204
7.4.4	Lessons Learned and Best Practices for Successful M&S Projects	204
7.5	Failure Avoidance for Agent-Based Modeling	206
7.5.1	Failure Avoidance in Rule-Based Systems	207
7.5.2	Failure Avoidance in Autonomous Systems	208
7.5.3	Failure Avoidance in Agents with Personality, Emotions, and Cultural Background	209
7.5.4	Failure Avoidance in Inputs	210
7.6	Failure Avoidance for Systems Engineering	212
7.7	Conclusion	213

8	Toward Systems Engineering for Agent-directed Simulation	219
	Levent Yilmaz	
8.1	Introduction	219
8.2	What Is a System?	220
8.2.1	What Is Systems Engineering?	220
8.2.2	The Functions of Systems Engineering	220
8.3	Modeling and Simulation	221
8.4	The Synergy of M&S and SE	221
8.4.1	The Role of M&S in Systems	221
8.4.2	Why Does M&S Require SE?	222
8.4.3	Why Is SSE Necessary?	222
8.5	Toward Systems Engineering for Agent-Directed Simulation	222
8.5.1	The Essence of Complex Adaptive Open Systems (CAOS)	223
8.5.2	The Merits of ADS	224
8.5.3	Systems Engineering for Agent-Directed Simulation	225
8.6	Sociocognitive Framework for ADS-SE	225
8.6.1	Social-Cognitive View	226
8.6.2	The Dimensions of Representation	227
8.6.3	The Functions for Analysis	228
8.7	Case Study: Human-Centered Work Systems	228
8.7.1	Operational Level – Organizational Subsystem	229
8.7.2	Operational Level – Organizational Subsystem	230
8.7.3	Operational Level – Integration of Organization and Social Subsystems	232
8.7.4	The Technical Level	232
8.8	Conclusions	235
9	Design and Analysis of Organization Adaptation in Agent Systems	237
	Virginia Dignum, Frank Dignum, and Liz Sonenberg	
9.1	Introduction	237
9.2	Organizational Model	239
9.3	Organizational Structure	240
9.3.1	Organizational Structures in Organization Theory	240
9.3.2	Organizational Structures in Multiagent Systems	241
9.4	Organization and Environment	242
9.4.1	Environment Characteristics	242
9.4.2	Congruence	244
9.5	Organization and Autonomy	245
9.6	Reorganization	247
9.6.1	Organizational Utility	247
9.6.2	Organizational Change	248
9.7	Organizational Design	250
9.7.1	Designing Organizational Simulations	252
9.7.2	Application Scenario	253
9.8	Understanding Simulation of Reorganization	256

9.8.1	Reorganization Dimensions	257
9.8.2	Analyzing Simulation Case Studies	257
9.9	Conclusions	263
10	Programming Languages, Environments, and Tools for Agent-directed Simulation	269
	<i>Yu Zhang, Mark Lewis, and Maarten Sierhuis</i>	
10.1	Introduction	269
10.2	Architectural Style for ADS	271
10.3	Agent-Directed Simulation – An Overview	272
10.3.1	Language	273
10.3.2	Environment	275
10.3.3	Service	276
10.3.4	Application	276
10.4	A Survey of Five ADS Platforms	277
10.4.1	Ascape	277
10.4.2	NetLogo	280
10.4.3	Repast	283
10.4.4	Swarm	286
10.4.5	Mason	289
10.5	Brahms – A Multiagent Simulation for Work System Analysis and Design	291
10.5.1	Language	291
10.5.2	Environment	295
10.5.3	Service	298
10.5.4	Application	299
10.6	CASESim – A Multiagent Simulation for Cognitive Agents for Social Environment	300
10.6.1	Language	302
10.6.2	Environment	302
10.6.3	Service	306
10.6.4	Application	310
10.7	Conclusion	312
11	Simulation for Systems Engineering	317
	<i>Joachim Fuchs</i>	
11.1	Introduction	317
11.2	The Systems Engineering Process	317
11.3	Modeling and Simulation Support	318
11.4	Facilities	320
11.5	An Industrial Use Case: Space Systems	321
11.5.1	Simulators for Analysis and Design	323
11.5.2	Facility for Spacecraft Qualification and Acceptance	325
11.5.3	Facility for Ground System Qualification and Testing and Operations	325
11.6	Outlook	325
11.7	Conclusions	327

12	Agent-directed Simulation for Systems Engineering	329
	<i>Philip S. Barry, Matthew T.K. Koehler, and Brian F. Tivnan</i>	
12.1	Introduction	329
12.2	New Approaches Are Needed	331
12.2.1	Employing ADS Through the Framework of Empirical Relevance	332
12.2.2	Simulating Systems of Systems	334
12.3	Agent-Directed Simulation for the Systems Engineering of Human Complex Systems	336
12.3.1	A Call for Agents in the Study of Human Complex Systems	337
12.3.2	Noteworthy Agent-Directed Simulations in the Science of Human Complex Systems	338
12.4	A Model-Centered Science of Human Complex Systems	338
12.5	An Infrastructure for the Engineering of Human Complex Systems	339
12.5.1	Components of the Infrastructure for Complex Systems Engineering	339
12.5.2	Modeling Goodness	341
12.5.3	The Genetic Algorithm Optimization Toolkit	341
12.6	Case Studies	344
12.6.1	Case Study 1: Defending The Stadium	345
12.6.2	Case Study 2: Secondary Effects from Pandemic Influenza	350
12.7	Summary	355

Part Four Agent-Directed Simulation for Systems Engineering 361

13	Agent-implemented Experimental Frames for Net-centric Systems Test and Evaluation	363
	<i>Bernard P. Zeigler, Dane Hall, and Manuel Salas</i>	
13.1	Introduction	363
13.2	The Need for Verification Requirements	364
13.3	Experimental Frames and System Entity Structures	366
13.4	Decomposition and Design of System Architecture	371
13.5	Employing Agents in M&S-Based Design, Verification and Validation	376
13.6	Experimental Frame Concepts for Agent Implementation	378
13.7	Agent-Implemented Experimental Frames	381
13.8	DEVS/SOA: Net-Centric Execution Using Simulation Service	382
13.8.1	Automation of Agent Attachment to System Components	382
13.8.2	DEVS-Agent Communications/Coordination	384
13.8.3	DEVS-Agent Endomorphic Models	386
13.9	Summary and Conclusions	388
13.A	cAutoDEVS – A Tool for the Bifurcated Methodology	391
14	Agents and Decision Support Systems	399
	<i>Andreas Tolk, Poornima Madhavan, Jeffrey W. Tweedale, and Lakhmi C. Jain</i>	
14.1	Introduction	399
14.1.1	History	399

14.1.2	Motivating Agent-Directed Decision Support Simulation Systems	401
14.1.3	Working Definitions	403
14.2	Cognitive Foundations for Decision Support	405
14.2.1	Decision Support Systems as Social Actors	406
14.2.2	How to Present the System to the User and Improve Trust	407
14.2.3	Relevance for the Engineer	410
14.3	Technical Foundations for Decision Support	411
14.3.1	Machine-Based Understanding for Decision Support	412
14.3.2	Requirements for Systems When Being Used for Decision Support	413
14.3.3	Agent-Directed Multimodel and Multisimulation Support	417
14.3.4	Methods Applicable to Support Agent-Directed Decision Support Simulation Systems	418
14.4	Examples for Intelligent and Agent-Directed Decision Support Simulation Systems	421
14.4.1	Supporting Command and Control	421
14.4.2	Supporting Inventory Control and Integrated Logistics	423
14.5	Conclusion	426
15	Agent Simulation for Software Process Performance Analysis	433
	<i>Levent Yilmaz and Jared Phillips</i>	
15.1	Introduction	433
15.2	Related Work	435
15.2.1	Organization-Theoretic Perspective for Simulation-Based Analysis of Software Processes	435
15.2.2	Simulation Methods for Software Process Performance Analysis	436
15.3	Team-RUP: A Framework for Agent Simulation of Software Development Organizations	437
15.3.1	Organization Structure	437
15.3.2	Team-RUP Task Model	438
15.3.3	Team-RUP Team Archetypes and Cooperation Mechanisms	439
15.3.4	Reward Mechanism in Team-RUP	440
15.4	Design and Implementation of Team-RUP	441
15.4.1	Performance Metrics	443
15.4.2	Validation of the Model	444
15.5	Results and Discussion	445
15.6	Conclusions	447
16	Agent-Directed Simulation for Manufacturing System Engineering	451
	<i>Jeffrey S. Smith, Erdal Sahin, and Levent Yilmaz</i>	
16.1	Introduction	451
16.1.1	Manufacturing Systems	452
16.1.2	Agent-Based Modeling	453
16.2	Simulation Modeling and Analysis for Manufacturing Systems	454
16.2.1	Manufacturing System Design	455
16.2.2	Manufacturing Operation	458
16.3	Agent-Directed Simulation for Manufacturing Systems	463

- 16.3.1 Emergent Approaches 463
- 16.3.2 Agent-Based Manufacturing 464
- 16.3.3 The Holonic Approach: Hierarchic Open Agent Systems 466
- 16.4 Summary 468

17 Organization and Work Systems Design and Engineering: from Simulation to Implementation of Multiagent Systems 475

Maarten Sierhuis, William J. Clancey, and Chin H. Seah

- 17.1 Introduction 475
- 17.2 Work Systems Design 475
 - 17.2.1 Existing Work System Design Methods 476
 - 17.2.2 A Brief History of Work Systems Design 477
- 17.3 Modeling and Simulation of Work Systems 478
 - 17.3.1 Designing Work Systems: What Is the Purpose and What Can Go Wrong? 478
 - 17.3.2 The Difficulty of Convincing Management 479
- 17.4 Work Practice Modeling and Simulation 480
 - 17.4.1 Practice vs. Process 481
 - 17.4.2 Modeling Work Practice 481
 - 17.5 The Brahms Language 487
 - 17.5.1 Simulation or Execution with Brahms 488
 - 17.5.2 Modeling People and Organizations 489
 - 17.5.3 Modeling Artifacts and Data Objects 490
 - 17.5.4 Modeling Communication 492
 - 17.5.5 Modeling Location and Movement 493
 - 17.5.6 Java Integration 495
 - 17.6 Systems Engineering: From Simulation to Implementation 496
 - 17.6.1 A Cyclic Approach 498
 - 17.6.2 Modeling Current Operations 499
 - 17.6.3 Modeling Future Operations 501
 - 17.6.4 MAS Implementation 502
 - 17.7 A Case Study: The OCA Mirroring System 503
 - 17.7.1 Mission Control as a Socio-Technical Work System 504
 - 17.7.2 The OCA Officer's Work System 505
 - 17.7.3 Simulating the Current OCA Work System 505
 - 17.7.4 Designing the Future OCA Work System 510
 - 17.7.5 Simulating the Future OCA Work System 511
 - 17.7.6 Implementing OCAMS 511
 - 17.8 Conclusion 514

Index 517

Preface

Simulation is the enabling technology for hundreds of very important application areas requiring any type of decision support (such as prediction, evaluation, testing, planning, acquisition, and proof of concept), understanding, and education, as well as training to develop and/or enhance motor skills to gain proficiency in the use of equipment, decision making and communication skills, and operational skills by getting real-life-like experience in controlled environments.

The maturity of simulation is (1) *facilitated* by the advances of computer hardware, software engineering, artificial intelligence, software agents, and system theories; (2) *due to* the dedicated contributions of several simulationists, the requirements of advanced users, and the support of influential people who realize its importance; and (3) *achieved* through developments, improvements, and especially through several paradigm shifts. As presented by Thomas Kuhn in his seminal book *The Structure of Scientific Revolutions*, three stages are necessary for a paradigm shift to occur. In the first stage, a paradigm becomes dominant. In the second stage, limitations or problems with the dominant paradigm are observed or better anticipated. In the third stage, a new paradigm that would surpass these limitations and problems is proposed and after some delay becomes the new dominant paradigm.

This book emphasizes the benefits of a double synergy: first, the synergy of modeling and simulation with software engineering, which leads to agent-directed simulation; and then the synergy of agent-directed simulation with systems engineering.

The use of agents in computational modeling has now become pervasive. The power of agents comes partly from their ability to conceptualize problems and devise solutions in terms of interacting entities that communicate, collaborate, coordinate, and intentionally deliberate their actions and reactions. The significant benefit gained by an event-based interactive computing perspective over algorithmic computation is due to our new understanding of complex systems and the universal principles, as well as patterns underlying their mechanisms. This coherent theme about the significance of interaction spans complex systems from cellular mechanisms in systems biology, physiology, brain dynamics, organizational dynamics, ecosystems, as well as patterns of human behavior and culture that define the dynamics of sociotechnical, cognitive, and cultural systems. The call for decentralized problem solving, adaptation, and flexibility in systems engineering

is also resulting in increased use of agent technologies to engineer robustness and resilience into complex artificial systems, while increasing our ability to explore and understand information processes underlying natural systems.

The motivation behind this book, however, goes beyond recognition of these observations. Since its adoption by the simulation modeling and systems engineering communities, the use of agents, unfortunately, has been limited to development of models that use agents as design metaphors. Yet this limited treatment of agents in simulation and systems engineering misses opportunities where agent and simulation technologies are together a central theme. Thus this book aims to fill a gap in the agent, modeling and simulation, and systems engineering communities. By expanding our horizons on the use of agents in modeling and simulation to build, explore, and understand both artificial and natural systems, the book presents a comprehensive framework, called agent-directed simulation, that consists of three distinct, yet related, areas that can be grouped under two categories as follows.

1. Simulation for agents (agent simulation): simulation of agent systems in engineering, human and social dynamics, military applications, etc.
2. Agents for simulation: agent-supported simulation deals with the use of agents as a support facility to enable computer assistance in problem solving, experimentation, or enhancing cognitive capabilities; agent-based simulation focuses on the use of agents for the generation of model behavior in a simulation study.

While agent-based modeling is widely appreciated and used in model-based science and engineering, the potential use of agents in developing next-generation intelligent and adaptive simulators and their inclusion in the simulation frontend or backend interfaces are not yet as widely acknowledged. Furthermore, the growth of new advanced distributed computing standards along with the rapid rise of service orientation is providing a new context that acts as a critical driver for the development of next-generation systems. These standards revolve around pervasive computing, Web services, grid, autonomic computing, ambient intelligence, etc. The supporting role that intelligent agents can play in the design and development of such systems is becoming pervasive, and simulation plays a critical role in the analysis and design of such systems. The synergy between systems engineering, simulation modeling, and agent technologies is examined in this book to facilitate mutual advancement of each area.

To explore interrelations between systems engineering, simulation modeling, and agent technologies, the book is comprised of three parts. First, we start with a background section that includes a comprehensive overview of modeling and simulation, agent paradigm, systems engineering, and quality assurance. In the second part, we examine the use of systems engineering principles, formal methods, tools, toolkits, and environments for developing agent-directed simulation systems. The final section focuses on the role that agent-directed simulation can play in various systems engineering problems such as testing and evaluation, process performance analysis, decision support, and organization and work system engineering.

Writing this book would not have been possible without the support of many people. The efforts of the referees were instrumental in shaping the contents. Esther Dörring, Anja Tschörtner, and other staff members at Wiley were also very helpful in facilitating the process as well as in supporting the production and final camera-ready copy. Special thanks to authors, since without their intellectual work and creative contributions this book could not exist.

We hope that the introduction of agent-directed simulation and systems engineering as a comprehensive framework that expands our horizons on the mutual contributions of modeling and simulation, software agents, and systems engineering will achieve a broader impact of the associated theories, methodologies, and applications and that challenging and complex problems can be tackled more appropriately. We both wish everyone a pleasant and fruitful time reading and using this book.

December 2008

Levent Yilmaz and Tuncer Ören

List of Contributors

Osman Balci

Virginia Tech
Department of Computer Science
3160B Torgersen Hall, MC 0106
Blacksburg, VA 24061
USA

Philip S. Barry

MiTRE Corporation
McLean, VA 22101
USA

Steven M. Biemer

Johns Hopkins University
Whiting School of Engineering
3400 North Charles Street
Baltimore, MD 21218-2608
USA

William J. Clancey

NASA Ames Research Center
Intelligent Systems Division
Moffett Field, CA 94035
USA

Frank Dignum

Utrecht University
Department of Information
and Computing Sciences
Centrumgebouw Noord
Padualaan 14
De Uithof/3584 CH Utrecht
The Netherlands

Virginia Dignum

Utrecht University
Department of Information
and Computing Sciences
Centrumgebouw Noord
Padualaan 14
De Uithof/3584 CH Utrecht
The Netherlands

Joachim Fuchs

European Space Agency
2200 AG Noordwijk
The Netherlands

Dane Hall

BAE Systems
Sierra Vista, AZ 85706
USA

Lakhmi C. Jain

University of South Australia
School of Electrical and
Information Engineering
Mawson Lakes Campus
Mawson Lakes, SA 5095
Australia

Matthew T.K. Koehler

MiTRE Corporation
McLean, VA 22101
USA

Mark Lewis

Trinity University
Department of Computer Science
San Antonio, TX 78212-7400
USA

Poornima Madhavan

Old Dominion University
Department of Engineering
Management and Systems
Engineering
Kaufman Hall
Norfolk, VA 23529
USA

Bradley Mitchell

Auburn University
3116 Shelby Center
Computer Science
and Software Engineering
College of Engineering
Auburn, AL 36849
USA

William F. Ormsby

Naval Surface Warfare Center
6149 Welsh Road, Suite 203
Dahlgren, VA 22448
USA

Tuncer I. Ören

University of Ottawa
Faculty of Engineering
800 King Edward
Ottawa, ON, K1N 6N5
Canada

Jared Phillips

Auburn University
3116 Shelby Center
Computer Science
and Software Engineering
College of Engineering
Auburn, AL 36849
USA

Andrew P. Sage

George Mason University
Systems Engineering &
Operations Research
MS4A6
4400 University Drive
Fairfax, VA 22030
USA

Erdal Sahin

Auburn University
3116 Shelby Center
Computer Science
and Software Engineering
College of Engineering
Auburn, AL 36849
USA

Manuel Salas

Modular Mining Systems, Inc.
3289 Hemisphere Loop
Tucson, AZ 85706-5028
USA

Maarten Sierhuis

NASA Ames Research Center
Carnegie Mellon Silicon Valley
Moffett Field, CA 94035-1000
USA

Liz Sonenberg

The University of Melbourne
Department of Information Systems
Victoria 3010
Melbourne
Australia

Chin H. Seah

NASA Ames Research Center
SGT, Inc.
Moffett Field, CA 94035-1000
USA

Jeffrey Smith

Auburn University
3116 Shelby Center
Computer Science
and Software Engineering
College of Engineering
Auburn, AL 36849
USA

Brian F. Tivnan

MiTRE Corporation
McLean, VA 22101
USA

Andreas Tolk

Old Dominion University
Department of Engineering
Management and Systems
Engineering
Kaufman Hall
Norfolk, VA 23529
USA

Jeffrey W. Tweedale

University of South Australia
School of Electrical and
Information Engineering
Mawson Lakes Campus
Mawson Lakes, SA 5095
Australia

Adelinde M. Uhrmacher

University of Rostock
Institute of Computer Science
Albert-Einstein-Str. 21
18059 Rostock
Germany

Levent Yilmaz

Auburn University
3116 Shelby Center
Computer Science
and Software Engineering
College of Engineering
Auburn, AL 36849
USA

Bernard P. Zeigler

University of Arizona
Department of Electrical
and Computer Engineering
1230 E. Speedway Blvd.
Tucson, AZ 85721
USA

Yu Zhang

Trinity University
Department of Computer Science
San Antonio, TX 78212-7400
USA

Part One Background

1

Modeling and Simulation: a Comprehensive and Integrative View

Tuncer I. Ören

1.1

Introduction

Compared to other well-established disciplines such as those in the humanities and the social, natural, formal, and applied sciences, modeling and simulation (M&S) is a relatively new discipline. For example, an important professional society, the Society for Modeling and Simulation International, was only formed in 1952 (SCS, 2008). However, the development of the field has been phenomenal. Currently, there are almost 100 associations and organizations in several aspects of M&S (M&S-AO, 2008), and M&S is successfully used in a range of application areas (Ören, 2009).

Daniel Bell, who promoted the concepts of postindustrial society and information age, also predicted the importance of simulation. In a seminal article, he posited that abstract theories, models, simulations, decision theory, and systems analysis would be the methodologies in postindustrial societies (Bell, 1976). He also said that “the key political problems in a post-industrial society are essentially elements of science policy”.

After many success stories in challenging application areas, possibilities are still growing on what one can achieve by proper use of M&S. At the dawn of the 21st century, as also acknowledged by the US Senate, M&S is indeed a critical area for the well-being of many advanced countries. The USA adopted House Resolution 487 in 2007 (HR487, 2008). Other countries or unions such as the European Union may benefit from the USA’s good example. Taking advantage of the full benefit of model-based simulation as well as complementary model-based activities wait to be tapped and offer opportunities for advanced developments in associated theories, methodologies, and technologies. In other words, decision makers who can perceive the big picture of M&S will be influential in taking advantage of the opportunities offered by M&S. This article aims to contribute to the development of a body of knowledge in M&S. Simulation is based on the concept of similarity, which has a very broad connotation. A list of over 80 terms other than simulation with similar connotations is given in the appendix.

1.2

Simulation: Several Perspectives

An early list of definitions of simulation was compiled by Pritsker (1979). Currently, many definitions of simulation can be found on the Web (simDefs-Web, 2008), and several hundred more definitions can be found by googling “definition of simulation”. Several groups prefer to use narrow definitions of M&S. For example, version 1.0 of the NATO Modelling and Simulation Master Plan has the following definition of simulation: “The execution over time of models representing the attributes of one or more entities or processes. Human-in-the-Loop simulations, also known as simulators, are a special class of simulations” (NATO-MP-v1.0, 2008). The Canadian Synthetic Environment Coordination Office uses the following definition: “A simulation is the implementation of a model over time” (simDef-SECO, 2008). Some of the definitions are very specific and even circular; for example, the glossary of an administration of the US Department of Health and Human Services defines simulation as follows: “A method of quantifying costs or benefits in which the process is analyzed and simulated to obtain costs” (US-DHaHS, 2008). Other definitions are even incorrect and misleading. For example, in one online dictionary, computer simulation is defined as an “Alternative term for computer modeling” (BD, 2008).

Simulation is used for hundreds of application areas and can be perceived from different perspectives. Most definitions of simulation are representative of this aspect-oriented perception of simulation (simDefs-Web, 2008). In this chapter, the intention is to offer generic definitions that can also be top-down decomposable to explore the rich paradigms that M&S is associated with. Having a comprehensive and integrative view of M&S is essential to benefiting from its full potential. As seen in Table 1.1, M&S can be perceived from the following perspectives: purpose of use, problem to be solved, connectivity of operations, types of knowledge processing, and philosophy of science.

1.2.1

Purpose of Use

Table 1.2 highlights three purposes of M&S. As a process, the term simulation has three meanings, two technical and one nontechnical.

As a technical term, two main types of uses of M&S can be identified. M&S is used (1) to perform experiments (for decision support, understanding, and education) and (2) to provide experience (for training and entertainment) through controlled conditions. The technical meanings cover any type of simulation regardless of whether or not the simulation is computerized and whether it is carried out on pure software or on any type of hardware/software. Furthermore, both of the technical meanings allow top-down decomposition of the entities and activities involved and thus enable their systematic and hierarchical elaborations. As a nontechnical term, simulation has been used in English since 1340 and means “imitation” or “fake”. The explanation in the Online Etymology Dictionary is: “a false show, false

Table 1.1 Perception of M&S from different perspectives.

Perception with respect to	Perceptions of simulation
Purpose of use	Perform experiments for: Decision support, Understanding, Education Provide experience for: Training, Entertainment Imitation (fake)
Problem to be solved	Black box perception (M&S is an infrastructure to support real-world activities)
Connectivity of operations	Standalone simulation Integrated simulation (symbiotic simulation)
Types of knowledge processing	Computational activity Systemic activity Model-based activity Knowledge generation activity Knowledge processing activity
Philosophy of science	Simulation supports and enriches modern scientific thinking [Francis Bacon (<i>Novum Organon</i> , 1620)]

Table 1.2 Three purposes of use of M&S.

Purpose of use of simulation	Type of simulation
Perform experiments for:	Simulation
• Decision support	
• Understanding	
• Education	
Provide experience (under controlled conditions) for:	
• Training (for gaining/enhancing competence):	
– motor skills	Virtual simulation
– decision and/or communication skills	Constructive simulation
– operational skills	Live simulation
• Entertainment	Simulation game
Imitation	Representation Fake

profession', from O. Fr. [Old French] *simulation*, from L. [Latin] *simulationem* (nom. *simulatio*) 'an imitating, feigning', noun of action from *simulare* 'imitate', from stem of *similis* 'like'" (OED-sim, 2008). The definition, from the same dictionary, for the related term *simulacrum*, which has been in use in English since 1599, is: "from L. *simulacrum* 'likeness, image, form, representation, portrait', dissimilated from *simulaclum*, from *simulare* 'to make like'. The word was borrowed earlier as *sem-*

ulacre (c. 1375), via O. Fr. *simulacre*" (OED-simulacrum, 2008). In this chapter, we focus on the technical meanings of the term simulation.

1.2.1.1 Experimentation

In areas other than training and entertainment, simulation is goal-directed experimentation with dynamic models. These areas include decision support, understanding, and education. Experimentation has been one of the key concepts in scientific thinking ever since Francis Bacon (1561–1626) advocated it in 1620 in his *Novum Organum (New Instrument)*. Bacon's work was a categorical departure from and reaction to the *Organon (The Instrument)*, which was the title given to Aristotle's (384–322 BC) logical works, which itself had an unparalleled influence on the history of Western thought (Ören, 2002). Hence, the technical definition related to experiments also ties simulation to the origins of modern scientific thinking. However a programmer's view of simulation would be biased to the execution of the simulation program and would hinder this important point. The advantage of performing the experiments on a model rather than on the real system is also well established.

Use of Simulation for Decision Support As outlined in Table 1.3, the use of simulation for decision support includes its use for prediction (of behavior and/or performance), evaluation of alternatives, sensitivity analysis, evaluation (of behavior and/or performance) of engineering design, virtual prototyping, testing, planning, acquisition, and proof of concept.

Use of Simulation for Understanding In analyses of mostly natural problems, simulation is a very powerful technique used to understand them. Several models can be tested until the behaviors of the model and the real system match under the same or very similar conditions. Understanding has several connotations. The meanings of understanding in machine understanding are applicable for

Table 1.3 Three purposes of use of M&S.

Prediction of behavior and/or performance of the system of interest within the constraints inherent in the simulation model (e.g., its granularity) and the experimental conditions

Evaluation of alternative models, parameters, experimental and/or operating conditions on model behavior or performance

Sensitivity analysis of behavior or performance of the system of interest based on granularities of different models, parameters, experimental and/or operating conditions

Evaluation of behavior and/or performance of engineering designs

Virtual prototyping

Testing

Planning

Acquisition (or simulation-based acquisition)

Proof of concept

simulation-based understanding. In Ören et al. (2007), nearly 60 types of machine understanding are presented in an ontology-based dictionary.

Use of Simulation for Education Simulation provides an excellent laboratory in teaching any type of dynamic system. A bibliography related to the classroom use of simulation has been prepared by O'Haver (2008). The site <http://www.site.uottawa.ca/~oren/MSBOK/sim4Ed.htm> (sim4Ed, 2008), though dated, contains links to several reference books, articles, and theses as well as many demonstration software resources and individual software in teaching areas such as acoustics; electricity, electronics, magnetism; energy, heat, thermodynamics; fluid dynamics; mathematics; mechanics; nuclear physics; optics; oscillations and waves; plasma physics; quantum physics; radiation; relativity; sound; superconductivity; and theoretical physics. Some portals and additional links can also be found.

1.2.1.2 Experience (for Training and Entertainment)

As outlined in Table 1.2, in training, simulation is used to gain/enhance competence through experience under controlled conditions. Three major types of simulation correspond to three types of training.

1. *Virtual simulation* (i. e., use of simulators or virtual simulators) is used to enhance motor skills to gain proficiency of use of equipment such as an airplane, a tank, or a car. In virtual simulation, real people use virtual equipment in virtual environments; hence the term “virtual simulation”.
2. *Constructive simulation* (or gaming simulation such as war gaming, peace gaming, international relations gaming, business gaming, etc.) is used to enhance decision making and/or communication skills. In constructive simulation, simulated people use simulated equipment in a virtual environment and real people get experience by interacting with the simulation system.
3. *Live simulation* is used to gain/enhance operational skills by getting real-life-like experience in a controlled environment. Live simulation is used in such diverse areas as military exercises as well as for the training of health specialists. In live simulation, real people use imitation (or virtual or dummy) equipment in the real world.

In entertainment (i. e., simulation games and some types of animation of dynamic systems), simulation provides experience under controlled conditions. “Getting experience under controlled conditions” is the common aspect in using M&S for training (i. e., gaining/enhancing competence) as well as for entertainment purposes. The term “serious game” is used to distinguish simulation games used in areas other than entertainment.

1.2.1.3 Imitation

As explained in Section 1.2.1, since the 14th century, the term “simulation” has been used to denote an imitation, a representation, something similar; when the dissimilarity was deceitful, then the term “simulation” was used to mean fake”. For

example, “simulated leather” means “imitation leather”. Sometimes, “to simulate” is used in the sense of “to pretend”. The French postmodern theorist Jean Baudrillard (1929–2007) is well known for elaborating on the interaction of reality and symbols in postmodern society, particularly in his *Simulacra and Simulation* (*Simulacres et Simulation* in French) (Baudrillard, 1985). Simulacrum, the singular form of simulacra, means (1) an image or representation; (2) an unreal or vague semblance. However, Baudrillard uses the term simulacra to mean “the copy without an original”. Understanding “hyperreality” may be helpful to clarify some concepts. “Hyperreality is closely related to the concept of the simulacrum: a copy or image without reference to an original. In postmodernism, hyperreality is the result of the technological mediation of experience, where what passes for reality is a network of images and signs without an external referent, such that what is represented is representation itself” (SEP-postmodernism, 2008).

For Baudrillard, modern societies are organized around the production and consumption of commodities, while postmodern societies are organized around simulation and the play of images and signs, denoting a situation in which codes, models, and signs are the organizing forms of a new social order where simulation rules. In the society of simulation, identities are constructed by the appropriation of images, and codes and models determine how individuals perceive themselves and relate to other people. Economics, politics, social life, and culture are all governed by the mode of simulation, whereby codes and models determine how goods are consumed and used, politics unfold, culture is produced and consumed, and everyday life is lived. In addition, Baudrillard’s postmodern universe is one of hyperreality in which entertainment, information, and communication technologies provide experiences more intense and involving than the scenes of banal everyday life, as well as the codes and models that structure everyday life. The realm of the hyperreal (e.g., media simulations of reality, Disneyland and amusement parks, malls and consumer fantasylands, TV sports, and other excursions into ideal worlds) is more real than real, whereby the models, images, and codes of the hyperreal come to control thought and behavior. Yet determination itself is aleatory in a nonlinear world where it is impossible to chart causal mechanisms in a situation in which individuals are confronted with an overwhelming flux of images, codes, and models, any of which may shape an individual’s thought or behavior (SEP-Baudrillard, 2008).

1.2.2

Problem to Be Solved

With respect to the problem to be solved, a perception of M&S is that it is as an infrastructure to support real-world activities. This is the black box perception by practitioners, who would like to emphasize that simulation is a tool to achieve other goals. This view allows concentrating on the original problems they face; for

example, for NASA the goal is successful space missions and not simulation; likewise for the military, similarly and justifiably, the goal is not simulation. From this perspective, simulation is perceived as not being the “real thing”. This attitude is well documented in STRICOM’s motto: “All but war is simulation” (STRICOM, 2008). This view can lead to successful applications of simulation in familiar areas. However, a broader view of M&S can lead to better appreciation of the full scope of possibilities that simulation offers. Hence, the limitations of this black box perception of simulation are: (1) not even wanting to know what one misses and (2) to be obliged to have “patches” in the conception of M&S when the need arises instead of benefiting from a comprehensive and integrative view.

1.2.3

Connectivity of Operations

As seen in Table 1.4, two important categories of simulation can be distinguished with respect to the connectivity of operations of simulation and the system of interest. They are: standalone simulation and integrated (or symbiotic) simulation. In *standalone simulation*, operations of the simulation and the system of interest are independent, that is, are not connected. The majority of simulations belong to this category. In *integrated (or symbiotic) simulation*, operations of the simulation and the system of interest are interwoven. In integrated simulation, simulation enriches or supports real-system operation. To enrich operations of the real system, the system of interest and the simulation program operate simultaneously to assure online diagnosis or augmented reality (enhanced reality) operation. To support operations of the real system, the system of interest and the simulation program operate alternately to provide predictive displays. Predictive displays are based on parallel experiments while the system is running.

1.2.4

M&S as a Type of Knowledge Processing

Simulation can be seen as a type of knowledge processing at different levels of abstraction (Table 1.1). Hence, simulation can be perceived as a computational activity, systemic activity, model-based activity, knowledge generation activity, or knowledge processing activity.

1.2.4.1 M&S as a Computational Activity

The role of the computer in simulation spans from the generation of model behavior to simulation-based problem-solving environments. Some definitions of simulation, for example, the execution over time of models representing the attributes of one or more entities or processes, concentrate on the lowest level of computerization activity. This computational view of execution of a simulation program may hinder the high-level possibilities of simulation-based computer-aided problem-solving environments such as computer-aided problem specification, model speci-

Table 1.4 Types of M&S with respect to the connectivity of operations of simulation and the system of interest.

Type of connectivity	Type of simulation
Operations of the simulation and the system of interest are: Not connected	Standalone simulation
Interwoven – Integrated simulation	<p><i>To enrich</i> real system's operation</p> <p>(The system of interest and the simulation program operate simultaneously)</p> <ul style="list-style-type: none"> • online diagnostics (or simulation-based diagnostics) • simulation-based augmented/enhanced reality operation (for training to gain/enhance motor skills and related decision making skills) <p><i>To support</i> real system operations</p> <p>(The system of interest and the simulation program operate alternately to provide predictive displays)</p> <ul style="list-style-type: none"> • parallel experiments while system is running

fication (model synthesis, model composition), model transformation, experimental frame specification (design and execution of experiments, as well as analyses of the results of the experiments), program generation, and symbolic processing of problem specifications to assure built-in quality. The concept of high-level computer assistance in M&S has been promoted since the early 1980s (Ören, 1982).

1.2.4.2 M&S as a Systemic Activity and System Theory-Based Simulation

From a systemic point of view, simulation can be used to find the values of output, input, or state variables of a system, provided that the values of the two other types of variables are known (Table 1.5). The state represents the structure of the system, that is, state variables, state transition function, and output functions.

In an *analysis* problem, the system is given. A model can be constructed to represent the system of interest. Hence the state variables are known. In a simulation run, the model is driven with a behavior generator under the experimental conditions to generate model behavior (output). In a *design* problem, input-output pairs are given as part of the design requirements; the problem is finding the state variables that may satisfy the input-output pairs. For a given design, hence a model to represent it, the state is given. Hence simulation runs can be performed with the given inputs until input-output pairs are satisfied. Then the state variables used cor-

Table 1.5 Three types of system problems.

System problems	Three types of system knowledge		
	Inputs	States	Outputs
Analysis	(given)	(given)	find
Design	(given)	find	(given)
Control	find	(given)	(given)

respond to the desired design. In a *control* problem, the system, and hence the state, is given along with the desired system behavior (i. e., system output). The problem is to find the necessary input to generate the desired output. Hence, with the given state, simulation runs can be performed until the desired output is obtained. Then the inputs used correspond to the necessary control.

As a very important and fundamental contribution to M&S, system sciences provide the basis for modeling formalisms as well as for symbolic processing of models for a wide variety of dynamic systems. These include automata, cellular automata, Lindenmayer systems (or L-systems), Petri nets, system dynamics, bond graphs, goal-directed systems, variable-structure systems, and evolutionary systems (Ören et al., 1984). Advances in discrete event systems specification (DEVS) by Zeigler (1984) and many valuable variant theories based on DEVS provide a robust theoretical background for the simulation of complex systems modeled as discrete event systems. The first model-specification language based on a system theory (Wymore, 1967) for continuous systems described by differential equations was developed in the early 1970s (Ören, 1971, 1984a). The craftsmanship of a carpenter who can build a summer cottage cannot scale up to build a skyscraper, which requires the appropriate engineering knowledge based on theoretical knowledge. Similarly, to build simulation systems for large and complex problems, system-theoretic-robust approaches are necessary, especially if one would like to avoid problems at later phases of projects.

1.2.4.3 M&S as a Model-Based Activity

The perception of simulation as a model-based activity has several advantages; among other possibilities, it allows for the construction of simulation-based computer-aided problem-solving environments (Zeigler et al., 1979; Ören, 1984b; Ören et al., 1984; Elzas et al., 1986, 1989). Currently, several disciplines have adopted model-based paradigms. They include systems engineering (Wymore, 1993), software engineering (mc-swEng, 2008), and model-driven enterprise information systems (md-EIS, 2008). In M&S, in addition to the generation of model behavior, the following can be considered:

- computer-aided modeling (model composability);
- model-base management (or management of model repositories, including their interfaces) (for reusability);

- parameter-based management (for example, in nuclear fuel waste management simulation systems, several thousand constants and parameters, some of which are represented as probability distribution functions, have to be managed);
- symbolic processing of models (see the section on model-based activities).

Of course, in M&S, model-based activities can be best achieved by using an appropriate mathematical system theory.

1.2.4.4 M&S as a Knowledge-Generation Activity

From an epistemological point of view, simulation is a knowledge generation activity; more specifically, simulation is a goal-directed knowledge generation activity with dynamic models within dynamic environments. This view allows advanced methodologists and technologists to integrate simulation with several other knowledge generation techniques (Ören, 1990). At this abstract level, the definition of simulation can be interpreted as model-based experiential knowledge generation. This abstraction facilitates the synergy of simulation with other knowledge generation (and processing) techniques. Knowledge can be generated with or without models by (experience-based or not) techniques that include instrumentation and experimentation in the real world; any type of simulation; computation, optimization, statistical inferencing, reasoning, hypothesis processing, as well as en-

Table 1.6 Types of simulation-based augmented reality.

	Type of environment (and equipment)	
	Real environment	Simulated environment
Real people (operator)	<p><i>Live simulation</i> Real people operate real and/or simulated equipment in real environment</p>	<p><i>Virtual simulation</i> Real people operate simulated equipment within virtual (simulated) environments</p> <ul style="list-style-type: none"> • Simulator • Virtual simulator (All software simulator)
	<p><i>Automated vehicles</i> Automatic control system (virtual operator) operates real equipment in real environment for example, vehicle without driver, aircraft without pilot; auto pilot</p>	<p><i>Constructive simulation</i> Simulated people use simulated equipment within simulated environments to provide experience to real people</p>
Virtual (simulated) people	<p><i>Virtual (simulated) training vehicle</i> Virtual people operate simulated equipment in real environment for example, virtual (simulated or AI) aircraft in a dogfight training; virtual tank, etc.</p>	

riched (augmented) reality. Table 1.6 outlines types of simulation-based augmented reality.

In live simulation, simulation as a knowledge generation activity is integrated with the operations of the real system where the real system acts as yet another source for knowledge generation.

1.2.4.5 M&S as a Knowledge-Processing Activity

This view allows advanced methodologists and technologists to integrate simulation with several model-based activities and several other knowledge processing techniques (Ören, 1990) to generate integrated simulation-based problem-solving environments. In this case, one can combine modeling, model processing, and other knowledge processing engines to have advanced simulation environments. These include integrated use of M&S with optimization, artificial intelligence, and software agents. In the last case, agents can also be used for additional purposes to assure the quality and reliability of operations. This view also allows for the combination of simulation systems with sensors and effectors. Sensors are energy transducers that transform energy into other types of energy or into information about the input energy that can be inputted to a simulation system. (Energy transducers and their relation to knowledge transducers were introduced by Ören (1990). Inversely, information inputted to an effector is transformed into energy to perform an action that comes about as a result of a deliberative system or a simulation system representing it.

1.2.5

M&S from the Perspective of Philosophy of Science

Simulation supports and enriches modern scientific thinking as promoted by Francis Bacon in 1620 in his *"Novum Organum"* by extending the possibilities for making experiments:

- Simulation allows experimentation (with dynamic models) when experimentation with a real system is not possible or feasible.
- Simulation enriches experimentation by allowing experimentation (with dynamic models) done under conditions not feasible with real systems.

1.3

Model-Based Activities

Model-based activities consist of model building, model-base management, and model processing (Table 1.7).

In software engineering, one of the sources of several types of failures has been the practice of the maintenance of the code, instead of maintenance of the specification. In M&S, since the early days program generators have existed to generate code from high-level specifications (Mathewson, 1974). In model-based simulation, the

Table 1.7 Types of model-based activities.

-
1. **Model building**
 - modeling
 - model synthesis
 - model composition (and dynamic model composition)
 2. **Model-base management** (and management of model repositories)
 - model search
 - semantic model search
 - model integrity
 3. **Model processing**
 - model analysis
 - model characterization (descriptive model analysis)
 - model evaluation (evaluative model analysis)
 - model transformation
 - behavior generation (generation of behavior of model)
-

Table 1.8 Some advantages of model-based simulation.

-
1. **Efficiency in Computerization**
 - Modelbases (or model repositories) may contain model specifications that can easily be converted into programs. Hence, programming aspect can and should be fully automated.
 - This aspect also eliminates programming errors and contributes to the reliability of the computerization of models.
 2. **Reliability**
 - Models can easily be read and understood by specialists in the field assuring model reliability.
 - Model specifications can be checked by specialized software as well as manually for consistency, completeness, and correctness. This aspect is definitely superior to traditional V&V techniques that work on code only and can be the basis for built-in reliability in M&S studies.
 3. **Reusability and Composability**
 - Model specifications can easily be modified for model reusability as well as model composition.
 - Some of the model composability techniques can be dynamically applicable for systems that not only have dynamic behavior but also can and should be modified dynamically as the simulation evolves.
 4. **Interoperability**
 - It is highly desirable to check interoperability of model specifications rather than the codes of models. Executability of code does not necessarily signify its semantic interoperability.
-

model specification can be transformed into a computer code by a program generator. Some of the advantages of model-based simulation are efficiency, reliability, reusability, and interoperability; they are summarized in Table 1.8.

1.3.1

Model Building

Modeling has long been automated in M&S. The applications started with filling in questionnaires (Oldfather et al., 1966) and the use of high-level simulation languages. For a review of the early developments see Nance (1983). Graphical modeling is another aspect that is used successfully. However, other possibilities exist for the use of dynamically tailorable templates for advanced modeling (Ören, 1991).

1.3.2

Model-Base Management

Model-base management facilitates the management of model specifications as opposed to the management of computerized expressions of models; hence, it facilitates the efficiency, reliability, reusability, and interoperability of models. Some additional issues are semantic search of models and especially model integrity.

1.3.3

Model Processing

As shown in Table 1.7, model processing consists of model analysis, model transformation, and behavior generation. From a pragmatic point of view, they can be applicable in model-based simulation where models are expressed in terms of appropriate mathematical systems theories that provide solid methodologies to specify models as well as to process them. The next generation of powerful model-based simulation environments can be realized by an integrative use of several model-based activities.

1.3.3.1 **Model Analysis**

There are two types of model analysis: descriptive and evaluative analyses. Descriptive model analysis is model characterization. Evaluative model analysis is model evaluation. As seen in Table 1.9, model characterization consists of model comprehensibility and model usability.

Table 1.10 summarizes model evaluation, or the evaluative analysis of models. Tables 1.11–1.14 outline model evaluation (i.e., evaluative model analysis) with respect to modeling formalisms, another model, real systems, and the goal of the study.

Validity, as a special type of model evaluation, has several types, as shown in Table 1.15. In a glossary of validation, one can list the definitions of each one of the terms given in Table 1.15. However, an ontology-based dictionary for these terms would have two advantages: (1) It can also provide the logical relationships of the terms since it will be based on a classification of the concepts and (2) the terms can be linked from a regular alphabetical list. For examples of ontology-based dictionaries, see Ören (2006) and Ören et al. (2007).

Table 1.9 Types of descriptive model analysis.

Model characterization (descriptive model analysis) for model comprehensibility
• model documentation
• static model documentation
• dynamic model documentation
• model ventilation (to examine its assumptions, deficiencies, limitations, etc.)
model usability
• model referability
• model-base management
• model integrity
• model composability
• model modifiability

Table 1.10 Types of evaluative model analysis (model evaluation).

Model evaluation (evaluative model analysis) with respect to:

- modeling formalisms
 - another model (model comparison)
 - real system
 - goal of study
-

Table 1.11 Types of model evaluation with respect to modeling formalisms.

**Model evaluation (evaluative model analysis) with respect to:
modeling formalisms**

- consistency of model representation
 - static structure of
 - component models
 - total system (coupled model, model of system of systems)
 - dynamic structure
 - state transitions, output function(s)
 - structural change
 - dynamic coupling
 - model robustness
-

Similarly, verification, as a special type of model evaluation, has several types, as shown in Table 1.16. In this case also, an ontology-based dictionary can display not only the definitions of each of the terms but also their logical relationships.

1.3.3.2 Model Transformation

Model transformation, as shown in Table 1.17, consists of model copying, model reduction, model pruning, model simplification, model elaboration, model isomorphisms, model homomorphisms, and model endomorphisms.

Table 1.12 Types of model evaluation with respect to another model (comparison).

Model evaluation (evaluative model analysis) with respect to: another model (model comparison)
<ul style="list-style-type: none"> • structural model comparison • model verification (comparison of a computerized model and corresponding conceptual model) • checking <ul style="list-style-type: none"> • model homomorphism • model isomorphism • model equivalencing for: <ul style="list-style-type: none"> • any two models • a simplified and original model • an elaborated and original model • behavioral model comparison (comparison of behaviors of several models within a given scenario)

Table 1.13 Types of model evaluation with respect to a real system.

Model evaluation (Evaluative model analysis) with respect to: real system
<ul style="list-style-type: none"> • model qualification <ul style="list-style-type: none"> • model realism (model veracity, model verisimilitude) • adequacy of model structure <ul style="list-style-type: none"> static structure (relevant variables, interface of models) dynamic structure • adequacy of model constants and parameters <ul style="list-style-type: none"> model identification model fitting model calibration • model correctness analysis <ul style="list-style-type: none"> • dimensional analysis • model validity <ul style="list-style-type: none"> • (see Table 1.15 for types of validity)

Table 1.14 Types of model evaluation with respect to the goal of study.

Model evaluation (Evaluative model analysis) with respect to: goal of study
<ul style="list-style-type: none"> • model relevance <ul style="list-style-type: none"> • domain of intended application(s) (appropriate use of a model) • range of applicability of a model • acceptability of a model with respect to its technical system specification

1.3.4

Behavior Generation

In a simulation process, the model is driven by a behavior generator under experimental conditions to generate the model behavior. Depending on the emphasis,

Table 1.15 Types of validity.

Absolute validity	Model validity
Conceptual validity	Multistage validity
Convergent validity	Operational validity
Cross validity	Parameter validity
Cross-model validity	Partial validity
Data validity	Predictive validity
Dynamic validity	Predictive model validity
Empirical validity	Replicative validity
Event validity	Statistical validity
Experimental validity	Strict validity
External validity	Structural validity
Face validity	Structural model validity
Full validity	Submodel validity
Gradual validity	Technical validity
Historical validity	Theoretical validity
Historical-data validity	Time-series validity
Hypothesis validity	Validity
Internal validity	Variable validity
Logical validity	

Table 1.16 Types of verification.

Black box verification
Code verification
Correctness verification
Data verification
Design verification
Formal verification
Functional verification
Independent verification and validation
Logical verification
Model verification
Model-based verification
Program verification
Verification
Verification of conceptual model

the behavior generator can also be called the simulation engine or the simulator, as is the case in the terminology used by Zeigler (1984). So long as there is no ambiguity, the choice of the terminology may be immaterial. However, since the term “simulator” is widely used for other devices such as airplane simulators or in

Table 1.17 Types of model transformation.**Model transformation**

- Model copying
- Model reduction
- Model pruning
- Model simplification
 - Structural model simplification
 - Behavioral model simplification
- Model elaboration
- Model isomorphism
- Model homomorphism
- Model endomorphism

terms such as “virtual simulators”, the more descriptive term “behavior generator” is used in this article. There are three types of model behavior: point behavior, trajectory behavior, and structural behavior. Generations of trajectory and structural behaviors correspond to trajectory and structural simulations, respectively. As seen in Table 1.18, behavior can be generated by numerical or nonnumerical techniques. Numerical techniques, especially for continuous models, are treated by Cellier and Kofman (2006). Nonnumerical techniques are used in artificial intelligence applications to M&S.

- *Point behavior*: This is the behavior of static models, that is, models whose behavior does not depend on time. Experimentation with static models is not simulation. Computation, optimization, and search are examples where the result (behavior) is point behavior. Point behavior can be scalar or n -dimensional vector.
- *Trajectory behavior*: Most simulations belong to this category where trajectories of descriptive variables are generated. At the end of a simulation study, trajectory behavior can be reduced to a performance index. Some possibilities are simulators, several types of simulation, and intermittent simulation such as optimizing simulation and gaming simulation.
- *Structural behavior*: In some studies, the evolution of the structure of a system can be generated by simulation techniques. Such systems include, for example, growth systems for which L-systems (or Lindenmeyer systems) provide a solid background. Some other applications include, for example, the spread of oil spills, forest fires, epidemics, and rumor.
- *Mixed behavior*: Some possibilities are mixed trajectory and structural behavior.

Table 1.18 Types of model behavior and behavior generation.**Types of model behavior**

- *point behavior*
 - computation
 - optimization
 - search
- *trajectory behavior*
 - simulators
 - simulation
 - intermittent simulation
 - optimizing simulation
 - gaming simulation
- *structural behavior*
 - growth systems
 - Lindenmeyer systems (L-systems)
- *mixed behavior*
 - mixed trajectory and structural behavior

Behavior generation by

- numerical techniques
- nonnumerical techniques
- mixed numerical and symbolic techniques

1.4**Synergies of M&S: Mutual and Higher-Order Contributions**

Mutual contributions of M&S, systems theories, and systems engineering to software engineering, artificial intelligence, and software agents are very important in advanced information technologies. Figure 1.1 depicts these mutual contributions. There are two types of mutual contributions. Tables 1.19–1.23 outline the direct contributions of M&S, systems theories, systems engineering, software engineering, artificial intelligence, and software agents to each other. Figure 1.1 can also be useful in conceiving higher-order contributions among these disciplines. For example, M&S contributes to software agents, which in turn can contribute to systems engineering, which can contribute to M&S. Similarly, several higher-order important contributions can be identified. These positive feedbacks make the mutual contributions even more important.

1.5**Advancement of M&S**

Due to its tremendous potential, discussions to advance the state of the art of M&S continue (Yilmaz et al., 2008). A framework for systematic discussion of normative views about the future of M&S is outlined in Table 1.25 (Ören (2008b)). The last sentence of another article states (Ören, 2002): “Progress in any area is not possible by keeping the status quo no matter how advanced it can be”. Indeed M&S is

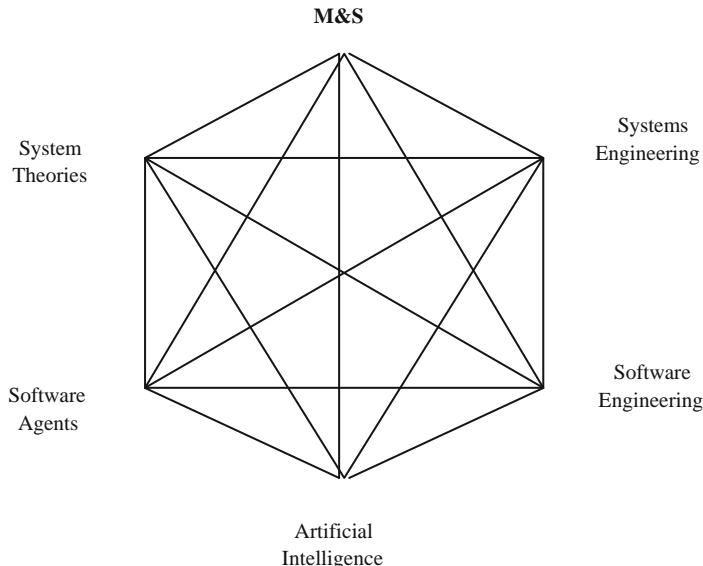


Fig. 1.1 Mutual contributions of M&S, systems theories, systems engineering, software engineering, artificial intelligence, and software agents.

Table 1.19 Contributions of M&S to systems theories, systems engineering, software engineering, artificial intelligence, and software agents.

Contributions of Modeling and Simulation to

- Systems Theories
- Systems Engineering
 - Bases for system design and analyses
 - Simulation-based experimentation to test system designs
- Software Engineering
 - Model-based software engineering
 - Simulation-based tests of software components and systems
- Artificial Intelligence
 - Modeling intelligent systems
 - Simulation-based testing of intelligent systems
- Software Agents
 - Agent simulation (Modeling and simulation of agents)
 - Simulation-based testing of agent systems

already very advanced and is also an important, sometimes vital, enabling technology for many areas. However, we ought to continue advancing it for several reasons: (1) to consolidate and disseminate pertinent knowledge about M&S; (2) to assure professional standards; (3) to advance M&S science, methodology, and technology to continue solving problems in hundreds of traditional application areas and

Table 1.20 Contributions of systems theories to M&S, systems engineering, software engineering, artificial intelligence, and software agents.

Contributions of Systems Theories to

- Modeling and Simulation
 - Bases for modeling & simulation of several categories of systems
 - Bases for symbolic processing of simulation models
 - Systems Engineering
 - Bases for system design
 - Software Engineering
 - Robust bases for model-based software engineering
 - Artificial Intelligence
 - Robust modeling bases for advanced intelligent systems such as goal-directed systems and goal-generating systems
 - Bases for symbolic processing of system models
 - Software Agents
 - Bases for modeling (design) of advanced agents
-

Table 1.21 Contributions of systems engineering to M&S, systems theories, software engineering, artificial intelligence, and software agents.

Contributions of Systems Engineering to

- Modeling and Simulation
 - Simulation systems engineering
(Systems engineering for simulation)
 - Distributed simulation systems engineering
 - Agent-directed simulation systems engineering
(Systems engineering for agent-directed simulation)
 - Agent-based simulation systems engineering
 - Systems Theories
 - Software Engineering
 - Software systems engineering
 - Artificial Intelligence
 - Software Agents
 - Systems engineering for large-scale/complex agent systems
-

challenging new areas. The framework offered here (which can be elaborated on and refined) may be useful (1) in consolidating and refining the views expressed by advanced users with high expectations as well as by theoreticians, methodologists, and technologists and (2) by systematically monitoring the progress. Some activities would require R&D; additional activities would include other types of professional and dedicated activities. My (over 100) publications, presentations, and other activities on advanced methodologies and normative views for advancement of M&S are listed in Ören (2008a).

In the sequel, Table 1.26 summarizes the activities for the consolidation of M&S knowledge, Table 1.27 outlines the dissemination of M&S knowledge, Table 1.28

Table 1.22 Mutual contributions of software engineering to M&S, systems theories, systems engineering, artificial intelligence, and software agents.

Contributions of Software Engineering to

- Modeling and Simulation
 - Computerized simulation
 - Computer-aided modeling and experimentation
 - Computerized tools, tool sets, and environments for M&S
 - Software engineering for agent-directed simulation
 - Systems Theories
 - CAST – Computer-aided system theories
 - Systems Engineering
 - Computerization of systems engineering functions
 - Artificial Intelligence
 - Computerization of AI techniques
 - Software Agents
 - Computerization of agents
-

Table 1.23 Contributions of artificial intelligence to M&S, systems theories, systems engineering, software engineering, and software agents.

Contributions of Artificial Intelligence to

- Modeling and Simulation
 - AI for M&S:
 - AI-supported M&S
 - AI-based M&S
 - Systems Theories
 - Systems Engineering
 - AI support for systems engineering
 - Cognitive systems engineering
 - Software Engineering
 - Intelligent software
 - Intelligent software engineering tools, tool set, and environments
 - Software Agents
 - Intelligent agents
-

concerns the requirements for professionalism, Table 1.29 elaborates on the science, engineering, and technology of M&S, Table 1.30 is on the trustworthiness, reliability, and quality, and, finally, and Table 1.31 is on more challenging applications.

Table 1.24 Contributions of software agents to M&S, systems theories, systems engineering, software engineering, and artificial intelligence.

Contributions of Software Agents to

- Modeling and Simulation
 - Agents for M&S:
 - Agent-supported M&S
 - Agent-based M&S
- Systems Theories
 - Agent-based systems theories
 - Necessities for specific system theories to model and symbolically process agents, such as moral agents (systems), understanding agents (systems)
- Systems Engineering
 - Agents for systems engineering
(Agent-based systems engineering)
- Software Engineering
 - Agents for software engineering
(Agent-based software engineering)
- Artificial Intelligence

Table 1.25 Categories of advancement areas for M&S.

Consolidation of M&S Knowledge

Dissemination of M&S Knowledge

Requirements of Professionalism

Science, Engineering, and Technology of M&S

Trustworthiness, Reliability, and Quality

Challenging M&S Applications

Table 1.26 Consolidation of M&S knowledge.

Comprehensive view (big picture) of all aspects of M&S

M&S Body of Knowledge (M&SBOK):

- Systematization of the index and preparation of a guideline

M&S Dictionaries

- As inventory of M&S concepts
- As systematic inventory of M&S concepts
(ontology-based M&S dictionary)

Curriculum development and international standardization for:

- Degree programs (graduate, undergraduate)
- Service programs for other disciplines
- Professional development courses

1.6

Preeminence of M&S

Tool making is an essential characteristic of humans or *Homo faber* “man the toolmaker”, as posited by Henri Bergson in *L'Evolution Creatrice* (*Creative Evolution*)

Table 1.27 Dissemination of M&S knowledge.

-
- (National, Regional, International) e-clearinghouse(s) of:
- Resource libraries
 - Funding agencies, funding sources
 - Documents of funded research
 - Dissertations/theses (as sources of specialists and specialized knowledge)
 - Centralized dissemination of professional information (events, job market)
 - e-encyclopedia, e-books, M&S portals
-

Table 1.28 Requirements of professionalism.

-
- **Ethics:** widespread adoption and practice of code of professional ethics (voluntary and/or required for individuals and/or M&S companies)
 - **Certification:** (voluntary and/or required for individuals and M&S companies)
 - **Assesment of maturity levels** of individuals and/or companies
 - **Recognition** of the M&S discipline and profession
 - US house **resolution 487** (widespread dissemination: nationally, internationally)
 - **Simulation Systems Engineering** needs to be promoted
 - Consider **analogies** with history of dentistry and professional engineering and current status where (uncertified) nonsimulationists doing simulation studies.
 - No need to be too humble (consider the use of the term “model” in art and in M&S as well as simulation-based engineering) (in art “model” designates real-world object!)
-

Table 1.29 Science, engineering, and technology of M&S.

Proper system-theory-based modeling and symbolic model processing for simulation of complex systems

Proper simulation paradigms and practices

- Model-based simulation
- Mixed formalism simulation
- Multisimulation
- Concurrent simulation
- Holonic agent simulation for
 - Simulation of **coopetition** (cooperative competition)
- Specification languages/environments for **interoperability**
- Computer-aided problem solving environments with M&S abilities

Advanced modeling formalisms and technologies

- With abstraction and descriptive power
- **Theory-based** modeling formalisms for:
 - Variable structure models
 - Multimodels
 - Multiaspect, multistage, multiperspective, multiresolution, multiparadigm, and evolutionary modeling

Advanced experimentation/scenario generation

- Automation of design & execution of experiments as well as analyses of results
-

Table 1.30 Trustworthiness, reliability, and quality.

-
- **Built-in reliability assurance** prior to traditional validation and verification
 - Proper computer-aided and computer-processable **documentation** of simulation studies (including assumptions)
 - **Taming**, monitoring, and assuring software agents in order for agents to behave in a trustworthy way
-

Table 1.31 Challenging M&S applications.

-
- Reduce **time of simulation** (from conception to generation of alternatives for decision makers)
 - Formulate **new success metrics**
 - Applications
 - Use of simulation for machine **learning**
 - Use of **switchable understanding** in simulation (to avoid dogmatic thinking and to assure emotional intelligence)
 - **Proactive** system simulation
 - **Introspective** system simulation
 - Simulation of **emergent phenomena**
 - **Conflict management** simulation
 - **Security** training simulation
 - **Personality, emotions, and cultural backgrounds** in simulation
-

tion) (Bergson, 1998). In this section, the preeminent position of M&S in the spectrum of the evolution of tools is presented. The term “tool” has several meanings. For example, the following are selected from the American Heritage Dictionary (AHD-tool, 2008): “1. A device, such as a saw, used to perform or facilitate manual or mechanical work. 2. A machine, such as a lathe, used to cut and shape machine parts or other objects. 3. Something regarded as necessary to the carrying out of one’s occupation or profession: *Words are the tools of our trade*. 4. Something used in the performance of an operation; an instrument: “*Modern democracies have the fiscal and monetary tools . . . to end chronic slumps and galloping inflations*” (Paul A. Samuelson). 5. *Computer Science* An application program, often one that creates, manipulates, modifies, or analyzes other programs.

Differences of meaning between tool, instrument, implement, utensil, and appliance are as follows.

Tool applies broadly to a device that facilitates work; specifically it denotes a small manually operated device: a box full of tools for bike repair. *Instrument* refers especially to a relatively small precision tool used by trained professionals: a sterilized scalpel and other instruments. *Implement* is the preferred term for tools used in agriculture and certain building trades: rakes, hoes, and other implements. *Utensil* often refers to an implement used in a household, especially in the kitchen: cooking utensils hung by the stove. *Appliance* most frequently denotes a power-driven de-

vice that performs a specific function: a store selling toasters and other appliances (AHD-tool, 2008).

To be able to see the preeminent place of M&S in the spectrum of the evolution of tools, let's consider three categories of tools: (1) physical tools, (2) knowledge-based or soft tools, and (3) knowledge generation tools. For each category, we will consider three levels, namely, manual tools, power tools, and cybernetic tools. Passing from one level to the next requires a critical additional feature. M&S, being model-based experiential knowledge generation, has a special place among knowledge generation tools.

1.6.1

Physical Tools

The *first level* of physical tools consists of manual tools including different types of stone and bone tools, metallic tools, mechanical tools, and electronic tools. A very large group of tools and instruments, from a hammer to a doctor's scalpel, or manual typewriters, and scopes including Galileo's telescope, or microscopes fall in the category of manual physical tools.

The *second level* of physical tools or power tools have access to added energy; hence they have the ability to perform work. They include simple power tools, machine tools, machines (including automotive vehicles), and integrated machines (including transfer machines used in building cars).

The *third level* of physical tools or cybernetic tools also have a knowledge processing ability. There are two types of knowledge processing machines: machines for knowledge processing and machines with a knowledge processing ability.

- *Machines for knowledge processing* include (digital) computers. However, there are two other types of machines for knowledge processing depending on whether they are fixed wired tools or machines such as an abacus, astrolabe, and bar-linkage computers. Variable-wired tools/machines for knowledge processing include early punched-card machines (or unit records), as well as analog and hybrid computers.
- *Machines with a knowledge processing* ability (or computer-embedded systems/machines – CES) benefit from the knowledge processing ability to perform their main goal. Most contemporary digital tools/devices/machines fall in this category. For example, digital cameras, smart buildings, smart roads, and so on. For more detail see Ören (1990).

1.6.2

Knowledge-Based or Soft Tools

Nonphysical tools are knowledge-based or soft tools. They include not only software tools but also other knowledge-based tools.

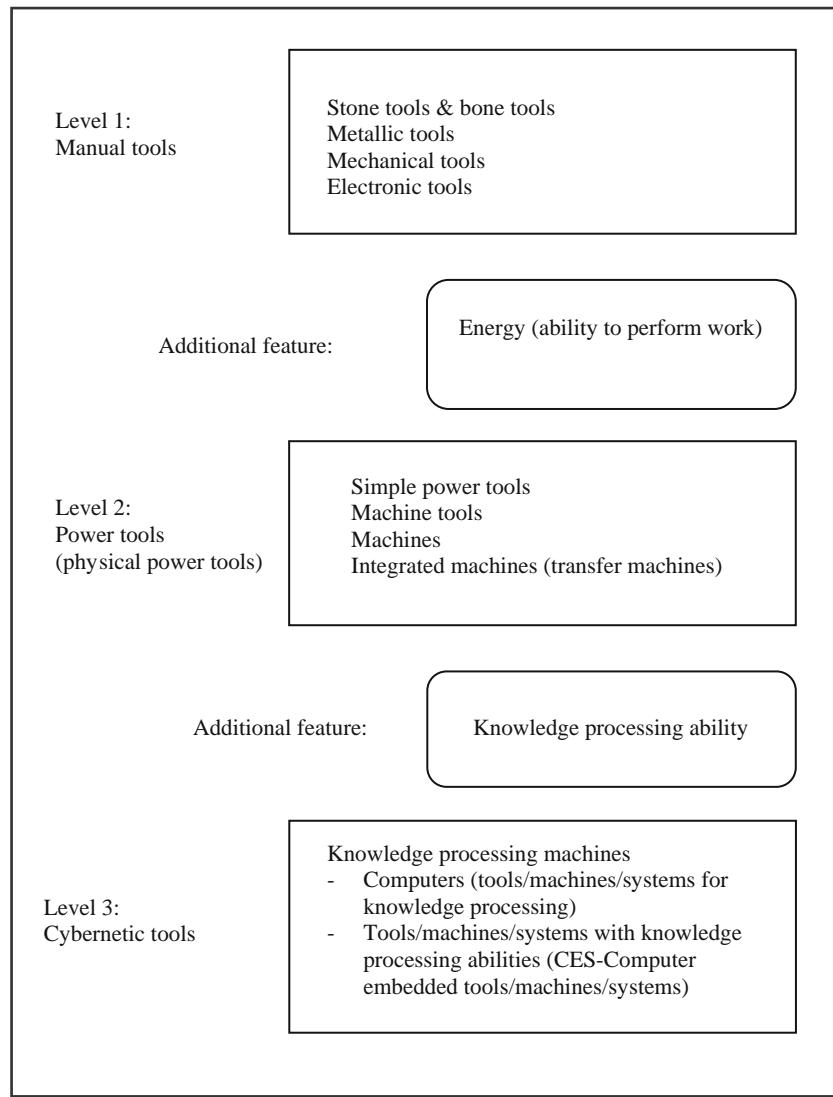


Fig. 1.2 Levels of physical tools.

The *first level of knowledge-based tools* includes software (hand-coded software programs and nonautomated documentation) and any other knowledge-based tools as exemplified in the definitions given in Section 1.6.

Computerization is the added critical feature to realize *knowledge-based power tools* or *knowledge-based soft tools*. As far as software is concerned, tools, toolkits, and environments of computer-aided software engineering fall in this category. For other knowledge-based tools, computerization is the key feature.

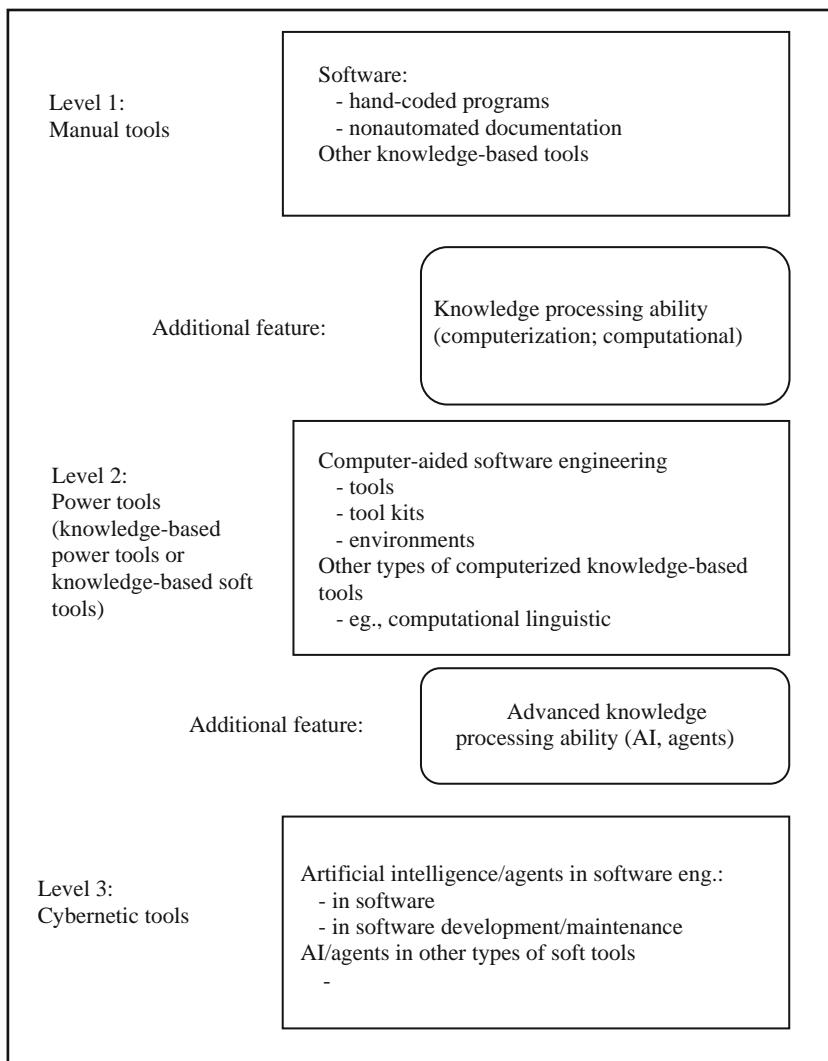


Fig. 1.3 Levels of knowledge-based and soft tools.

Advanced knowledge processing is needed to pass to cybernetics tools. For software engineering, there are two possibilities: AI in software engineering and software agents in software engineering. For other knowledge-based tools, similarly, contributions of AI or software agents are essential.

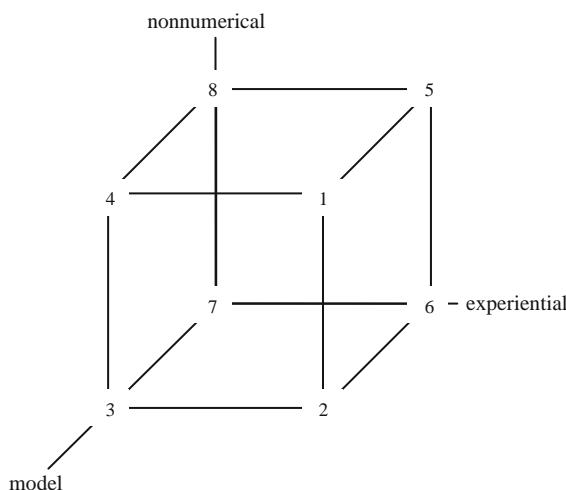
Table 1.32 Eight types of knowledge generation.

	Model or real system	Experimentation	Knowledge	Technique			
	Model-based	Real system-based	Experiential	Nonexperiential	generation with	Numerical	Non-numerical
1	✓		✓		✓		✓
2	✓		✓		✓	✓	
3	✓			✓	✓	✓	
4	✓			✓	✓		✓
5		✓	✓		✓		✓
6	✓		✓		✓	✓	
7	✓			✓	✓	✓	
8	✓			✓	✓		✓

1.6.3

Knowledge Generation Tools

As outlined earlier, M&S can also be perceived as a knowledge generation tool, more specifically as a model-based experiential knowledge generation tool. Hence, M&S has close affinity with other knowledge generation techniques. This third category of tools includes any knowledge generation tools. However, since the aim, is to point out the important and dominant position of M&S, we concentrate on the relative position of M&S. For the sake of completeness, an outline of the typology of knowledge generation is given in Figure 1.4 and Table 1.32.

**Fig. 1.4** Three dimensions of the typology of knowledge generation.

Three factors determine the essential characteristics of knowledge generation: (1) knowledge generation is based on a real system or a model, (2) knowledge generation is either experiential or it is not, and (3) the technique is numerical or not. These three dimensions and the eight possibilities are shown in Figure 1.4. The eight types of knowledge generation are as follows (the first two correspond to M&S). Table 1.32 outlines eight possibilities for knowledge generation.

1. model-based experiential knowledge generation with nonnumerical techniques;
2. model-based experiential knowledge generation with numerical techniques;
3. model-based nonexperiential knowledge generation with numerical techniques;
4. model-based nonexperiential knowledge generation with nonnumerical techniques;
5. real-system-based experiential knowledge generation with nonnumerical techniques;
6. real-system-based experiential knowledge generation with numerical techniques;
7. real-system-based nonexperiential knowledge generation with numerical techniques;
8. real-system-based nonexperiential knowledge generation with nonnumerical techniques.

At the first level, M&S tools are manual. They include simulation with scale models, sandbox simulations, and any manual simulation. To pass to the second level, computerization is needed. Computerized simulation (or, as it is also called, computer simulation even when the system being simulated is not a computer system) offers a gamut of possibilities from the generation of model behavior to simulation-based comprehensive problem-solving environments. Several tools, toolkits, and environments exist for computer-aided M&S, including problem specification, modeling, model-base management, design and execution of experiments, and analysis and visualization of results.

An advanced knowledge processing ability (i. e., artificial intelligence or software agents) is necessary to achieve the third level (or cybernetic) of M&S tools. AI-directed and agent-directed simulations are realized with the synergy of M&S with AI and agents, respectively.

- AI-directed simulation consists of simulation of intelligent entities, AI-supported simulation, and AI-based simulation (Ören, 1994, 1995).
- As is clarified in this volume, agent-directed simulation consists of agent simulation, agent-supported simulation, and agent-based simulation.

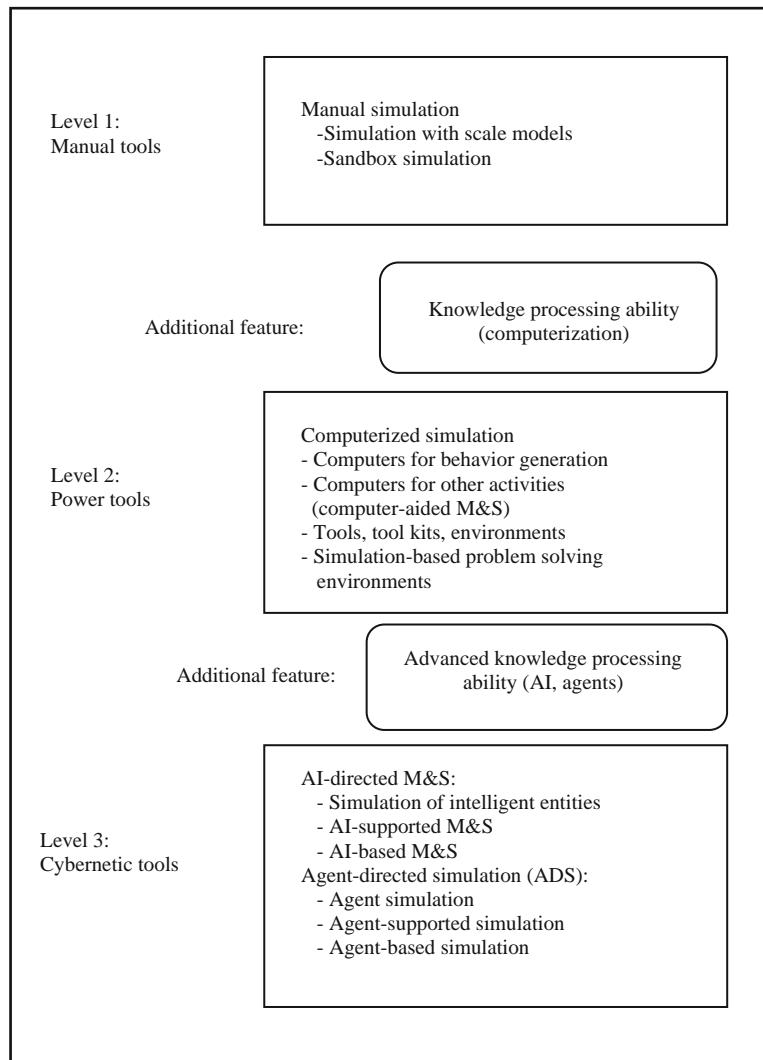


Fig. 1.5 Levels of M&S tools (model-based experiential knowledge-generation tools).

1.7

Summary and Conclusions

A comprehensive and integrative view of modeling and simulation (M&S) is offered to those interested in seeing the consolidated big picture and benefiting from all aspects of M&S. To this end, M&S is considered from five perspectives as follows: (1) purpose of use, that is, experiments (for decision support, understanding, and education); experience (for three types of training and entertainment); and imitation; (2) problem to be solved; (3) connectivity of operations of simulation and

a real system; (4) M&S as a type of knowledge processing (i. e., M&S as a computational activity, systemic activity, model-based activity, knowledge generation activity, and knowledge processing activity); (5) from the perspective of philosophy of science. Afterwards, model-based activities are clarified under the following headings: (1) model building, (2) model-base management, and (3) model processing. The latter is covered under descriptive and evaluative model analyses, model transformation, and model generation. Important synergies of M&S with system theories, systems engineering, software engineering, artificial intelligence, and software agents are outlined under mutual contributions as well as higher-order contributions. The M&S discipline is already in its maturing phase (Ören, 2005). The preeminence of M&S is explained by outlining the special place of M&S in the spectrum of the evolution of tools. The similarity concept, which is the foundation of simulation, offers a rich paradigm. An appendix of over 80 terms besides simulation with similarity connotations is also given. These terms show the richness of the concept of similarity and can be useful in languages other than English in selecting/suggesting equivalent terms.

Appendix

Tab. 1.A.1: Concepts other than simulation with similarity connotation.

analog	dissimilatory	imitative
analogic	dissimilitude	isomorph
analogue	dissimulate (v)	isomorphic
analogous	dissimulated	isomorphism
analogy	dissimulating	isomorphous
assimilate (v)	dissimulation	like
assimilated	dissimulator	likeness
assimilation	emulate (v)	metaphor
autosimulation	emulated	metasimulation
autosimulative	emulating	nonsimulatable
bisimilar	emulation	pataphor
bisimulate (v)	emulative	replica
bisimulation	emulator	resemblance
comparable	endomorph	self-similar
comparative	endomorphic	similar
compare (v)	endomorphism	similarity
comparison	endomorphous	similitude
congruent	homology	simulacra
congruence	homomorph	simulacre
congruity	homomorphic	simulacrum

congruous	homomorphism	simuland
copy (v)	homomorphous	simulatable
cosimulation	homothetic	simulate (v)
cosimulative	homothetism	simulated
differentiation	homothety	simulating
dissimilar	hyperreality	simulationist
dissimilarity	imitate (v)	simulative
dissimilation	imitate (v)	simulator
dissimulator	imitation	symbol

References

- AHD-tool (2008) *The American Heritage Dictionary*, <http://www.bartleby.com/61/62/T0266200.html> (accessed 4 December 2008).
- Baudrillard, J. (1985) *Simulacres et Simulation* (original in French). Translation in English: *Simulacra and Simulation*, The University of Michigan Press.
- BD (2008) *Business Dictionary*, <http://www.businessdictionary.com/definition/computer-simulation.html>. (accessed 3 December 2008).
- Bell, D. (1976) Welcome to Post-Industrial Society. *Physics Today*.
- Bergson, H. (1998) *L'Evolution Creatrice*. (Creative Evolution, translated by Arthur Mitchell), Dover, New York.
- Cellier, F.E. and Kofman, E. (2006) *Continuous System Simulation*, Springer, Norwell, MA.
- Elzas, M.S., Ören, T.I., Zeigler, B.P. (1986) *Modelling and Simulation Methodology in the Artificial Intelligence Era*, North-Holland, Amsterdam, p. 423.
- Elzas, M.S., Ören, T.I., Zeigler, B.P. (1989) *Modelling and Simulation Methodology: Knowledge Systems Paradigms*, North-Holland, Amsterdam, p. 487.
- HR 487 (2008) *U.S. Senate House Resolution 487*, <http://thomas.loc.gov/cgi-bin/query/Dc110:1/temp/c110wpVyNH> (accessed 3 December 2008).
- Mathewson, C.S. (1974) Simulation program generators. *Simulation*, 23 (6), 181–189.
- M&S-AO (2008) *Modeling and Simulation Associations and Organizations*, <http://www.site.uottawa.ca/~oren/links-MS-AG.htm> (accessed 3 December 2008).
- mc-swEng (2008) *Model-centric Software Engineering*, <http://3m4mda.telin.nl/> (accessed 4 December 2008).
- md-EIS (2008) *Model-driven Enterprise Information Systems*, <http://www.iceis.org/workshops/mdeis/mdeis2007-cfp.html> (accessed 4 December 2008).
- Nance, R.E. (1983) A Tutorial View of Simulation Program Development. *Technical Report CS83025-E*. Virginia Polytechnic Institute and State University, Blacksburg, VA. An invited presentation for the 1983 Winter Simulation Conference, Washington, DC.
- NATO-MP-v1.0 (2008) *NATO Modelling and Simulation Master Plan, version 1.0*, AC/323 (SGMS)D/2, <ftp://ftp.rta.nato.int/Documents/MSG/NMSMasterPlan/NMSMasterPlan.pdf> (accessed 3 December 2008).
- OED-sim (2008) *Online Etymology Dictionary*, <http://www.etymonline.com/index.php?term=simulation> (accessed 3 December 2008).
- OED-simulacrum (2008) *Online Etymology Dictionary*, <http://www.etymonline.com/index.php?search=simulacrum> (accessed 3 December 2008).
- O'Haver, T. (2008) *Simulations and Computer Models in the Classroom*, <http://terpconnect.umd.edu/~toh/simulations.html> (accessed 3 December 2008).
- Oldfather, M.P., Ginsberg, S.A. and Markowitz, M.H. (1966) Programming by Questionnaire: How to Construct a Program Generator. *RAND Report RM-5129-PR*.

- Ören, T.I. (1971) *GEST: A Combined Digital Simulation Language for Large-Scale Systems*. Proceedings of the Tokyo 1971 AICA (Association Internationale pour le Calcul Analogique) Symposium on Simulation of Complex Systems, September 3–7, 1971, Tokyo, Japan, pp. B-1/1–B-1/4.
- Ören, T.I. (1982) Computer-Aided Modelling Systems, in *Progress in Modelling and Simulation*, (ed. F.E. Cellier), Academic Press, London, England, pp. 189–203.
- Ören, T.I., Zeigler, B.P. and Elzas, M.S. (1984) *Simulation and Model-Based Methodologies: An Integrative View*, Springer-Verlag, Berlin, Heidelberg, New York, Tokyo.
- Ören, T.I. (1984a) GEST – A Modelling and Simulation Language Based on System Theoretic Concepts, in *Simulation and Model-Based Methodologies: An Integrative View* (eds T.I. Ören, B.P. Zeigler, M.S. Elzas), Springer-Verlag, Heidelberg, Germany, pp. 281–335.
- Ören, T.I. (1984b) Model-Based Activities: A Paradigm Shift, in *Simulation and Model-Based Methodologies: An Integrative View*, (eds T.I. Ören, B.P. Zeigler, M.S. Elzas), Springer-Verlag, Heidelberg, Germany, pp. 3–40.
- Ören, T.I. (1990) A Paradigm for Artificial Intelligence in Software Engineering, in *Advances in Artificial Intelligence in Software Engineering*, vol. 1 (ed. T.I. Ören), JAI Press, Greenwich, Connecticut, pp. 1–55.
- Ören, T.I. (1991) Knowledge-Based Simulation: Methodology and Application, in *Knowledge-Based Simulation: Methodology and Application*, (eds P.A. Fishwick and R.B. Modjeski), Springer-Verlag, Berlin, Heidelberg, New York, Tokyo, pp. 53–76.
- Ören, T.I. (1994) Artificial intelligence and simulation. *Annals of Operations Research*, 53, 287–319.
- Ören, T.I. (1995) *Artificial Intelligence and Simulation: A Typology*. Proceedings of the 3rd Conference on Computer Simulation, (ed. S. Raczyński), Nov. 15–17, Mexico City, pp. 1–5.
- Ören, T.I. (2002) *Future of Modelling and Simulation: Some Development Areas*. Proceedings of the 2002 Summer Computer Simulation Conference, pp. 3–8.
- Ören, T.I. (2005) *Maturing Phase of the Modeling and Simulation Discipline*. Proceedings of: ASC – Asian Simulation Conference 2005, (The Sixth International Conference on System Simulation and Scientific Computing (ICSC' 2005)), October 24–27, Beijing, P.R. China, International Academic Publishers – World Publishing Corporation, Beijing, P.R. China, pp. 72–85.
- Ören, T.I. (2008) *Ontology-Based M&S Dictionaries: An Example for V&V*, <http://www.site.uottawa.ca/~oren/MSBOK/terms-ob-VV.htm> (accessed 4 December 2008).
- Ören, T.I., Ghasem-Aghaee, N. and Yilmaz, L. (2007) *An Ontology-Based Dictionary of Understanding as a Basis for Software Agents with Understanding Abilities*. Proceedings of the Spring Simulation Multiconference (SpringSim'07), March 25–29, Norfolk, VA.
- Ören, T.I. (2008a) *List of Publications, Presentations and Other Activities on Normative Views for Advancements of M&S*, <http://www.site.uottawa.ca/~oren/pubsList/MS-advanced.pdf> (accessed 4 December 2008a).
- Ören, T.I. (2008b) A Framework for Desirable Activities and Research in M&S. Part of the Position Statements for the Panel Discussion (ed. L. Yilmaz) *What Makes Good Research in Modeling and Simulation: Sustaining the Growth and Vitality of the M&S Discipline*. Proceedings of the 2008 Winter Simulation Conference, pp. 677–689, Miami, FL, 2008.
- Ören, T.I. (2009) Uses of Simulation, in *Principles of Modeling and Simulation: A Multi-disciplinary Approach*, (eds J.A. Sokolowski and C.M. Banks). John Wiley and Sons, Inc. New Jersey, 153–179.
- SCS (2008) *Society for Modeling and Simulation International*, <http://www.scs.org/AboutSCS.cfm> (accessed 3 December 2008).
- SEP-Baudrillard (2008) *Stanford Encyclopedia of Philosophy*, <http://plato.stanford.edu/entries/baudrillard> (accessed 3 December 2008).
- SEP-postmodernism (2008) *Stanford Encyclopedia of Philosophy*, <http://plato.stanford.edu/entries/postmodernism> (accessed 3 December 2008).
- Pritsker, A.A.B. (1979) Compilation of Definitions of Simulation. *Simulation*, 33 (2), 61–63.

- sim4Ed (2008) *Simulation in Education and Training*, <http://www.site.uottawa.ca/~oren/MSBOK/sim4Ed.htm> (accessed 3 December 2008).
- simDef-SECO (2008) *DND/CF Modelling & Simulation/Synthetic Environments Lexicon*, <http://www.drdc-rddc.gc.ca/seco/documents/MSLexiconApr02e.html> (accessed 3 December 2008).
- SimDefsWeb (2008) *Simulation Definitions on Web*. <http://www.google.ca/search?source=definition+simulation> (accessed 3 December 2008).
- SimDefsWeb (2008) *Simulation, Training, and Instrumentation*, <http://www.globalsecurity.org/military/agency/army/stricom.htm> (accessed 3 December 2008).
- US-DHaHS (2008) *US Department of Health and Human Services – Administration for Children & Families*, <http://www.acf.hhs.gov/programs/cb/systems/>
- sacwis/cbaguide/appendixb.htm (accessed 3 December 2008).
- Wymore, W.A. (1967) *A Mathematical Theory of Systems Engineering: The Elements*. Krieger, Huntington, NY.
- Wymore, W.A. (1993) *Model-Based Systems Engineering*. CRC Press, Boca Raton.
- Yilmaz, L., Davis, P., Fishwick, P., Hu, X., Miller, J.A., Hybinette, M., Ören, T.I., Reynolds, P., Sarjoughian, H. and Tolk, A. (2008) *What Makes Good Research in Modeling and Simulation: Sustaining the Growth and Vitality of the M&S Discipline*. Proceedings of the 2008 Winter Simulation Conference, Miami, FL, pp. 677–689.
- Zeigler, B.P., Elzas, M.S., Klir, J.G. and Ören, T.I. (1979) *Methodology in System Modelling & Simulation*, North-Holland, Amsterdam, p. 537.
- Zeigler, B.P. (1984) *Multifaceted Modelling and Discrete Event Simulation*, Academic Press.

2

Autonomic Introspective Simulation Systems

Levent Yilmaz and Bradley Mitchell

2.1

Introduction

Autonomic computing is a potential strategy and philosophy in systems design and management that aims to cope with increasing complexity in the presence of constant change. At the core of the autonomic computing paradigm is the notion of open and intelligent self-managing systems that facilitate adapting a system to evolving volatile and unstable environmental conditions. The term autonomic computing was originally envisioned to characterize IT infrastructures that imitate autonomic nervous systems so as to alleviate or, if necessary, delegate tedious management tasks from IT practitioners to the system itself. Autonomic systems are based on architectures and mechanisms that facilitate self-reconfiguration and adaptation through learning, anticipation, and robust designs to be able to adjust and fine-tune system parameters to emerging situations in the environment. Such systems are envisioned to exhibit the following major characteristics:

- Be aware of not only the status of its own components, but also the resources it uses and the context in which it is embedded;
- Be able to sense, perceive, and understand the environmental conditions to notice change so that its services can be tailored and fine-tuned to respond in ways that improve its functions;
- Be able to plan and facilitate change by altering its own configuration and status via dynamic update and composition so that selected or emergent goals can be satisfied.

The fundamental areas for which these capabilities are envisioned include the following: (1) *self-configuration* capability that enables the system to respond to unforeseen situations in the environment by dynamically updating its configuration through adding or removing components, as well as reorganizing the structure of the system without disrupting the service; (2) *self-healing* or *self-protection* capabilities that render the system capable of avoiding or recovering from failures and intrusions by diagnosing or anticipating possible failure states; (3) *self-optimization*

mechanisms that monitor the environment and improve the operation of the system by continuously fine-tuning its parameters so as to keep the system's performance at optimal levels despite the changes in the operating conditions induced by volatile environments.

In simulation modeling various methodologies (e.g., online simulation, symbiotic simulation) are proposed to establish a synergistic interaction between simulations and physical systems (environment of the simulation) so that they can mutually benefit each other for the purpose of prediction and control. Yet these existing methodologies do not leverage the full-scale benefits of autonomic mechanisms that enable self-configuration and dynamic composability. In this chapter, we examine the role of agent-based and agent-supported simulation in developing next-generation autonomic simulation systems that exhibit self-organization and adaptation characteristics as the behavior of the system and the simulation unfolds.

Proper simulation-based decision support methodologies that facilitate making decisions in field settings could improve modeling course of actions (COAs), simulating them faster than real time, and then performing COA analysis to improve the robustness and resilience of decisions. Exploring the effectiveness of alternative COAs requires dynamic updating, branching, and simultaneous execution of simulations, potentially at different levels of resolution. Dynamic updating of simulation models is a key requirement for using simulation as a tool to improve systems in which information becomes available only once the system is in progress (Yilmaz, 2004). In these types of systems, the initial conditions provide little or no insight into how the system may develop over time. The emergent behavior that arises dynamically is a primary source of information in these systems. Exploiting this information to enable robust decision making in a timely manner requires the ability to observe the system in real time and adapt useful characteristics for the system with as little computational effort as possible.

As a motivating example, let us consider the following. Realistic training scenarios are typically characterized by significant uncertainty. Proper symbiotic simulation-based training support methodologies that are consistent with the way experts use their experience to make decisions in field settings could improve the credibility of training games. Exploring the effectiveness of alternative contingency scenario models at the tactical and operational levels requires dynamic updating, branching, and simultaneous execution of agent-based simulations, potentially at different levels of resolution. The evolution of the environment in which the decision makers operate could require identifying and bringing (or evolving) one or more new families of models that are consistent with the observed state of the context. Experimenting with evolutionary and/or contingency models in real time on demand would be critical for decision support in unstructured problems with the characteristics of (1) deep uncertainty, (2) dynamic environments, and (3) shifting, ill-defined, and competing goals. The major challenges pertaining to decision making in such asymmetric and irregular environments include the following (Yilmaz et al., 2007):

- For most realistic problems, the nature of the problem changes as the simulation unfolds. Initial parameters, as well as models, can be irrelevant under emergent conditions. Relevant models need to be identified and instantiated to continue exploration. Manual exploration is not cost effective and realistic within a large problem state space.
- Our knowledge about the problem being studied may not be captured by any single model or experiment. Instead, the available knowledge is viewed as being contained in the collection of all possible modeling experiments that are plausible given what is known and what is learned.
- Dealing with uncertainty is paramount to analyzing complex evolving phenomena. Adaptivity in simulations and scenarios is necessary to deal with emergent conditions for evolving systems in a flexible manner.

The rest of the chapter is organized as follows. In Section 2.2 we overview requirements and design principles for developing autonomic systems. Section 2.3 focuses on the prospects, issues, and challenges involved in designing a specific type of autonomic system, called a decentralized autonomic simulation system. An architectural framework is presented in Section 2.4 to suggest a generic and reference design strategy for such systems. Then in Section 2.5, a concrete example that realizes the proposed architecture is discussed. The design and application of the SAMS system are used as a case study to substantiate the principled design strategy proposed in Section 2.4. Finally, in Section 2.6, we conclude by discussing the benefits of autonomic simulation systems along with potential avenues for further research.

2.2

Perspective and Background on Autonomic Systems

Autonomic computing was raised as a challenge and strategic initiative by Paul Horn, vice president of research at IBM. Since then, significant advancements in embedding autonomic capabilities in IT products have occurred. Among these advancements are the development of chips that can sense their environment so as to change their circuit configuration to improve processor performance and intelligent active networking systems that can dynamically update traffic routing policies. Yet, in a complex system-of-systems setting, proper orchestration of services through collaboration and coordination mechanisms is needed to properly adjust the operation of the overall system without disrupting service.

Figure 2.1 depicts the operational cycle of an autonomic system. The fundamental building blocks for autonomic systems are those that enable sensing the environment and monitoring and analyzing observed data; in addition, they allow planners and execution engines that generate actions through effectors embedded in the environment to manage the resources. The sensor components are used to gather data from the resources in the environment, while the monitoring component is registered to be notified as the sensors observe changes in the environment.

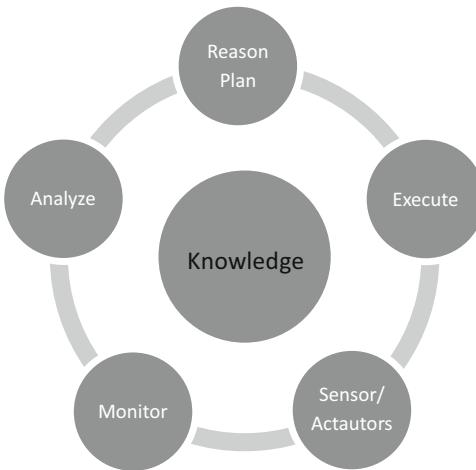


Fig. 2.1 Autonomic control loop.

The analysis of the monitored data involves perceiving, interpreting, and understanding the status of the environment, as well as the operational components of the system to react to changes. The knowledge source provides the necessary information about the managed resources and the data and policies required to manage them. If the analysis of the monitored data and knowledge embedded in the system cannot identify a proper reaction to unforeseen environmental conditions, the reasoning and planning component takes control to generate a new plan and identify a sequence of actions to act on the system configuration. The execution component translates the actions of the plan into executable commands to update the system state or manipulate the resources in the environment through the effectors of the system.

In updating the status of the system configuration and/or behavior, the autonomic control loop acts as a self-organization mechanism. Recently, particular methodologies using the concepts of self-organization have been proposed in different areas, such as software engineering (Wooldridge, 2002) electrical engineering, and collaborative support. However, there is as yet no general framework for constructing self-organizing systems. The term self-organization has been used in different areas with different meanings, as in cybernetics, thermodynamics, biology, mathematics, computing, and information theory.

A system can be considered a self-organizing complex system if its components dynamically interact to achieve a global goal or function. In decentralized self-organizing systems, the function is not imposed by a central component, but rather attained through autonomous interactions that produce feedback to regulate the system. A common characteristic of such systems is the unpredictability and uncertainty in the environment. Self-organizing systems use two major mechanisms to cope with uncertainty and volatility in the environment:

- *Adaptation:* The system uses learning techniques such as self-reinforcement learning or evolution (e.g., genetic algorithms, evolutionary computing) to adapt or update its behavior to changes in the environment.
- *Anticipation:* An anticipatory system is a system whose next state depends on its current state as well as the current image(s) of its future state(s). Anticipation involves perception of the future state(s) of the system and/or environment to proactively update the structure and behavior of the system.

A successful self-organizing system needs to be robust so that it can operate despite the perturbations in the environment. Robustness can be achieved via redundancy. Adaptation enables the system to change its behavior to increase its fitness with the environment. Robustness allows the system to be resilient so that it can adapt. Proactiveness, on the other hand, helps the system to adapt by anticipating changes.

2.3

Decentralized Autonomic Simulation Systems: Prospects and Issues

Autonomic simulation systems are specific types of autonomic systems that are useful in a variety of applications such as continuous system prediction and optimization, adaptive experience management in simulation games for training and learning, and real-time decision support under uncertainty. Systems characterized by nonlinear interactions among diverse agents often exhibit emergent behavior that may be very different from what the initial conditions of these systems would suggest. Traditional simulation techniques that rely on accurate knowledge of these conditions typically fail in these cases. Autonomic simulation systems have the potential to enable robust decision making in real-time for these problems. The insights derived from the autonomic simulation can be used to improve the performance of the system under study. Likewise, as the system develops, observations of emerging conditions can be used to improve exploration of the model ensemble. In essence, a useful coevolution between the physical system and autonomic simulation occurs.

2.3.1

Motivating Scenario: Adaptive Experience Management in Distributed Mission Training

Consider the following example. An inspection team under the command of the team of participants is at a weapons storage site in a fictional city. The team members are represented by software agents, as well as other human users engaged in the game. According to the original scenario, the inspection team discovers that weapons from the site are missing and that a hostile crowd is forming around them. As the inspection team radios for help, the members of the command staff must prepare and launch a rescue operation. Evidence begins to mount that the

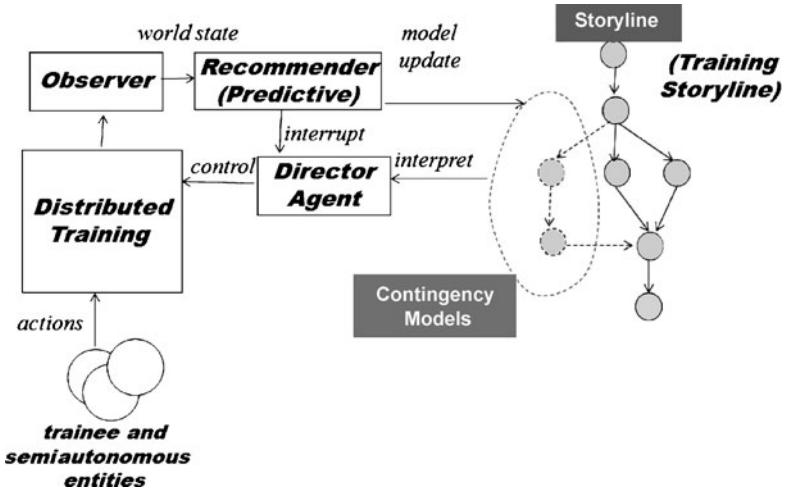


Fig. 2.2 Adaptive distributed mission training.

weapons were stolen by paramilitary troops who are motivating the hostile crowd. As additional paramilitary troops stream into the town, the command staff must overcome a series of obstacles in order to rescue the inspection team without incident or injury. The commander decides to call for air fire. Recognizing that this decision will lead to civilian casualty, which is in conflict with the learning objectives of the game, the simulation gaming environment updates the scenario (e.g., modifies the atmosphere data to bring fog into the area) so that the user can be brought back into a new scenario that is consistent with the objectives of the learning study. The challenge that needs to be addressed is the following:

- *How can we provide to trainers as much freedom as possible, while assuring that the training goals are achieved by exerting control on the scenario flow?*

As shown in Figure 2.2, an autonomic simulation gaming system monitors the actions of users and agents to perceive and understand the state of the gaming environment to make predictions about current and/or future states of the game. An online scenario recommender generates new plans to adapt the game so that learning objectives can be attained. New contingency models and scenarios are then identified to update the current scenario. The updated scenario is then interpreted by the director agent to control the distributed training environment. The control mechanism involves notifying the semiautonomous agents to play the role of team members in accordance with the updated scenario.

2.3.2

An Architectural Framework for Decentralized Autonomic Simulation Systems

As depicted in the above scenario, an autonomic simulation system should not only be able to reconfigure the simulation through dynamic composition, but also

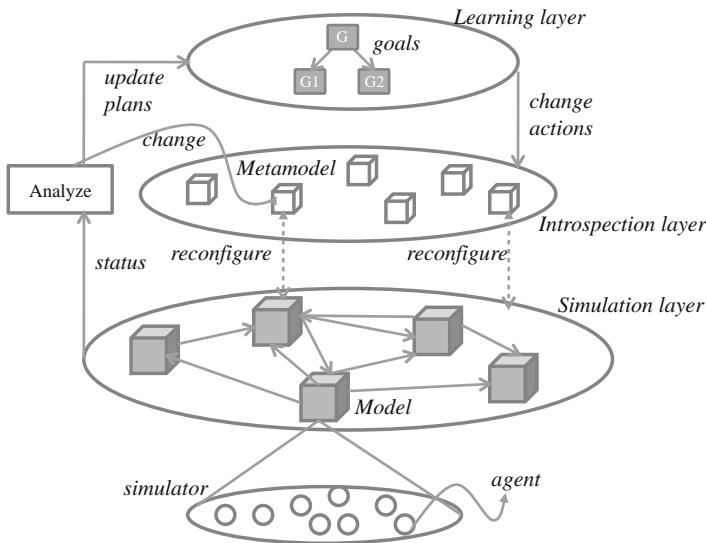


Fig. 2.3 Three-Layer architecture for self-management.

be able to manage the objectives of the system to generate new plans, if necessary. To facilitate a tractable and effective solution to seamless change and goal management, the autonomic control mechanism needs to be decoupled from the simulation system so that the autonomic control can evolve independently of the simulation, while providing self-awareness capabilities.

We choose to use a high-level component-based architectural strategy as a framework for the design and development of autonomic simulation systems to (1) enable the application of the proposed concepts to a wide variety of domains, (2) provide an appropriate level of abstraction, and (3) improve the potential for scalability of such systems. The premise of the proposed architecture is inspired by multilayer hybrid agent architectures and the recognition of the need for abstract models (metamodels) of the simulation models used in the simulation system. In what follows, we attempt to interpret layered agent architectures for autonomic simulation systems. Figure 2.3 depicts the three layers and their interaction.

The bottom layer is the *simulation* that includes a set of interconnected models that accomplish the application objective of the simulation. The simulation layer has capabilities to enable the creation, destruction, and reconfiguration of the models and generate status information that can be analyzed to facilitate online recommendations for changes. In decentralized autonomic simulation systems the model behavior is generated via interactions of semiautonomous agents that are able to perceive and act in accordance with their observation of their local environment. The capability of agents to observe the environment and interpret their perception with respect to their adaptive behavioral rules refers to the first-order change in autonomic simulation systems. Besides the models and their simulators, the simulation layer includes components for event and status reporting to

support modification – addition, deletion, and interconnection of models. When a situation is met that the current configuration of the simulation is not designed to deal with, the introspection and adaptation layers take control to identify proper strategies to update the simulation.

The introspection layer includes metamodels that encapsulate the schema of models along with behavior that is designed to manipulate and update the schema on demand based on the change requests generated by the analysis component. The analysis component is a situation and context awareness module that utilizes the status information to perceive and understand the emergent status of the simulation system. The second-order change in this layered autonomic simulation architecture pertains to updating the metamodels, which in turn updates (reconfigures) the associated models to dynamically steer the behavior to attain the objectives of the simulation application.

The learning layer is the deliberation part of the architecture and is activated to modify goals and associated plans by creating new metamodels through a learning process (e.g., adaptation through evolution). Specifically, the third-order change initiated by the learning layer decided if a schema survives or dies because of the survival or death of its corresponding metamodel. A schema can change through random or purposeful mutation and/or combination with other schemata. Hence, model schemata can undergo three types of change: first-order change, where action is taken in order to adapt the observation to the existing schema; second-order change, where there is purposeful change in the schema in order to better fit observations; and third-order change, where a new schema is produced. Schema change generally has the effect of making the agent more robust (it can perform in light of increasing variation or variety), more reliable (it can perform more predictably), or grow in requisite variety (it can adapt to a wider range of conditions). The fitness of the agent is a complex aggregate of many factors, both local and global. The general health or fitness of the agent determines what the probability of change will be. Optimization of local fitness allows differentiation and novelty/diversity; global optimization enhances the CAS coherence as a system and induces long-term memory. In general the probability of second-order schema change is a nonlinear function of the fitness value.

2.3.3

Challenges and Issues

In the previous section we outlined a three-layer architecture as a generic reference model and a framework to discuss challenges and issues pertaining to the development of autonomic simulation systems.

2.3.3.1 Simulation Layer

A simulation that is constructed in terms of models and their connections has a configuration or management state defined in terms of a status that specifies the active models, their interconnections, and set of modes depicting the state of models. The challenges at this level of the architecture is primarily the safe operation of

the system during change. To facilitate the introspection and learning layers to update the configuration of the simulation layer, proper design principles are needed. The following are generic requirements for the simulation layer.

Requirement 1: The simulation layer should be decoupled from the knowledge regarding how its models are composed, created, and represented. This requirement is critical to facilitate configuration of the Multi-resolution, Multi-stage Multimodeling (MRMSM) with one of multiple families of models representing an entity at different levels of resolution for a specific stage of the problem. Furthermore, this requirement imposes the constraint that a family of models is designed to work together, and that the simulation layer should be independent of how these model families are created and composed. This requirement enables isolating the implementation of models from the simulation layer.

Requirement 2: During the simulation, due to the need for first-order change discussed above, the entities in a model should be allowed to alter their behavior when their internal states are changed. Simulation behavior depends on the state and/or stage of the problem, and it must change its behavior at runtime depending on that state/problem phase. Therefore, the simulation layer should enable localization of state-specific behavior and partition behavior for different stages. The simulation layer should enable the definition of a family of related protocols, encapsulate each one, and make them interchangeable to facilitate varying the protocols independently of the entities that are configured with them.

Requirement 3: The constraints regarding when and under what conditions (1) the consistency of the elements of families of models in the simulation must be enforced and (2) a shift in the stage of the problem must be independent of the simulation layer. Corresponding to the time path of the change of a problem should be a time path of the appropriate model families. But the question is what should be the sequence of this shift pattern of models of family of? Or should there be trigger mechanisms indicating when a shift should occur? This requirement suggests that the constraints should be separated from the MRMSM model and not be intertwined with the entities to facilitate reuse of the model families in a different context with different constraints.

Requirement 4: At the time of the simulation update, the new set of models and their dependent components must be dynamically loaded and linked to the runtime environment of the simulation. This requires new model and simulator decoupling strategies that avoid persistent connections to facilitate extensibility. Also, the intricate details of a complex model instantiation process should be as independent of the simulation as possible to enable flexible updating.

Requirement 5: The consistency of entities undergoing (re)placement needs to be preserved. The event scheduling and simulation protocol needs to be restricted or regulated to facilitate interleaving of entity replacement activities with the simulation events.

Requirement 6: The state of the simulation layer must be constructed, or at least continue from a specific state after an update operation. This requires externalization through abstraction, state saving, transmission, and reconstruction after the update operation.

Requirement 7: The simulation should have a mechanism for changing the structure and behavior of the models dynamically. In other words, a model should support its modification a priori. This requirement suggests that simulation needs to provide facilities that establish a self-representation of the model, offer means by which this representation can be updated, and assure that the manipulations to the self-representation influence the behavior of the model. This requirement pertains to the connection between the simulation and introspection layers.

2.3.3.2 Introspection Layer

This layer is responsible for managing second-order change and executing the updates in response either to status information reported by the simulation layer or the goal changes imposed by the learning layer. As part of the introspection layer, the situation and context awareness component (e.g., the analyzer) provides perception, understanding, and anticipation capabilities to recognize the status of the present and/or future states of the simulation layer to enact the change management policy of the autonomic system.

Endsley (1995) defines situation awareness as the perception of elements in a particular environment within time and space, the comprehension of their meaning, and the projection of their status in the near future. Situation awareness, as depicted here, provides a set of mechanisms that enable attention to cues in the environment and expectancies regarding future states. In realistic settings, establishing an ongoing awareness and understanding of important situation components pose the major tasks of the autonomic system. Awareness suggests three main functional areas that revolve around a mental model of the problem domain. More specifically, a well-defined mental model provides

1. knowledge about the concepts, attributes, associations, and constraints that pertain to the application domain,
2. a mechanism that facilitates integration of domain elements to form an understanding of the situation, and
3. a mechanism to project to a future state of the environment given the current state, selected action, and the knowledge about the dynamics of the environment.

Situation awareness is an important cognitive skill that is essential for performance in any field involving complexity, dynamism, uncertainty, and risk. The failure to perceive a situation correctly may lead to faulty understanding. Ultimately, this misunderstanding may degrade a system's ability to predict future states and engage in effective change management.

Besides the awareness function, the interoperation layer needs a metamodel protocol (MMP) that supports implementation of functions that operate on one or more metamodels. Among these functions are those that manipulate parameters of models that are uncertain and hypothesized often by probability distribution functions. Emergent knowledge about the environment improves the accuracy of such parameters, the characteristics of which are specified within the metamodels of corresponding models.

2.3.3.3 Learning Layer

This layer of the autonomic simulation system is an online planning and adaptation subsystem that is enacted when existing schemata are not effective and fit with the evolving conditions in the simulation layer and/or its environment (e.g., the case of symbiotic simulation). In general, this layer is responsible for generating new goals and associated plans, the actions of which are carried out by the introspection layer to reconfigure the simulation. New goals and plans may require the addition and deletion of models, as well as the reconfiguration of their interconnections. Implementing the learning layer with online model recommenders is difficult because the interpretation of emergent conditions requires mining the state of the simulation to recognize situations within the domain theory (schema) of an application. The parameter instances of a simulation may not be readily available and need to be tuned on the basis of the observed events and high-level situations. Learning takes place as recommendations are made. Adaptive models that assume certain discernible patterns in the recommendations may be used to discover invariant situations and associated relevant models so as to reinforce matching problems to proven solutions based on previous experience.

Effective sampling of a model space of potentially infinite size requires a robust and efficient search algorithm. Such an algorithm should be capable of broad exploration while uncertainty is pervasive within the model space. This requirement implies the need for some stochastic element to deal with the combinatorial explosion of potential configurations (Dreo et al., 2006). Conversely, if uncertainty is decreasing, an appropriate search algorithm should also be capable of rapid exploitation of increasing information. This requirement rules out a simple random walk of the search space. Evolutionary computation (EC) provides techniques for implementing search algorithms with effective exploration and exploitation properties (Dreo et al., 2006). These algorithms model the processes of Darwinian evolution as a means of accomplishing a robust stochastic search (Goldberg, 1989).

2.4

Symbiotic Adaptive Multisimulation: An Autonomic Simulation System

Symbiotic simulation (S2) (Fujimoto et al., 2002) involves the use of simulation systems that are synchronized with the physical systems to enable mutually beneficial adaptation. In S2, simulation outputs are examined and used to determine how the physical system may be optimized. Similarly, measurements from the physical system are used to validate the simulation. When uncertainty in the physical system is present, multiple what-if simulation experiments can be helpful in adjusting the physical system. However, since the number of what-if experiments that may be performed is limited by both computational and real-time constraints, the ability to conduct an efficient search of the model space is essential.

Symbiotic Adaptive Multisimulation (SAMS) is intended as an S2 technique appropriate for physical systems characterized by distributed, dynamic, and uncertain

conditions. It is heavily inspired by the fields of multisimulation (Yilmaz, 2007), exploratory analysis (Davis and Bigelow, 2000), and exploratory modeling (Bankes, 1993), which involve the use of an ensemble of plausible models to provide insight into the absence of a single authoritative model. The salient feature of SAMS is the use of evolutionary computation in terms of a genetic algorithm (GA) to evolve the model ensemble in response to changes in the physical system. With this feature, it is conjectured that an effective search of an uncertain model space would be possible, thus permitting synchronization with the physical system.

2.4.1

Metamodels for Introspection Layer Design

In order to efficiently search a potentially infinite number of plausible models, SAMS uses a hybrid exploration technique. As shown in Figure 2.4, uncontrollable inputs and controllable inputs are handled with an input analysis module and an output analysis module, respectively. Measurements of the physical system's behavior are used to hypothesize distributions for uncontrollable inputs. As more details of the physical system's environment become known, the fidelity of these distributions to the actual values of the physical system should improve. Controllable input factors representing the configuration of the physical system are evolved using a GA. A set of controllable inputs that completely describes a potential system configuration can be thought of as an individual. The controllable factors are thus considered to be the decision variables of a given problem while the uncontrollable factors determine the shape of a dynamic fitness landscape. An evolutionary algorithm enables the adaptation of individuals through a process of natural selection, with better performing individuals being more likely to survive and ultimately be used as the configuration settings for agents within the physical system itself.

2.4.1.1 Partial Model Ensembles

Once a distribution has been hypothesized for each uncontrollable factor in the physical system, and the associated parameters for those distributions have been estimated, each factor is sampled multiple times. These samples are stored in a 2-dimensional array referred to as a partial model ensemble (PME). Variates for each sampled distribution are associated on a per-sample basis and placed into the same row of an array. Each column of the array stores variates from a single input factor, and each row of the array is considered to be a partial plausible model of the physical system. Later, these partial models are combined with potential system configurations to create fully specified plausible models for simulation. To hypothesize distributions for uncontrollable inputs and estimate their parameters, a means of observing the physical system is required. As real-time symbiotic simulation is performed, observations of the physical system enable more accurate estimates of the input distribution parameters. With these improvements, the space of plausible models shrinks, allowing the system to simulate fewer models in greater detail.

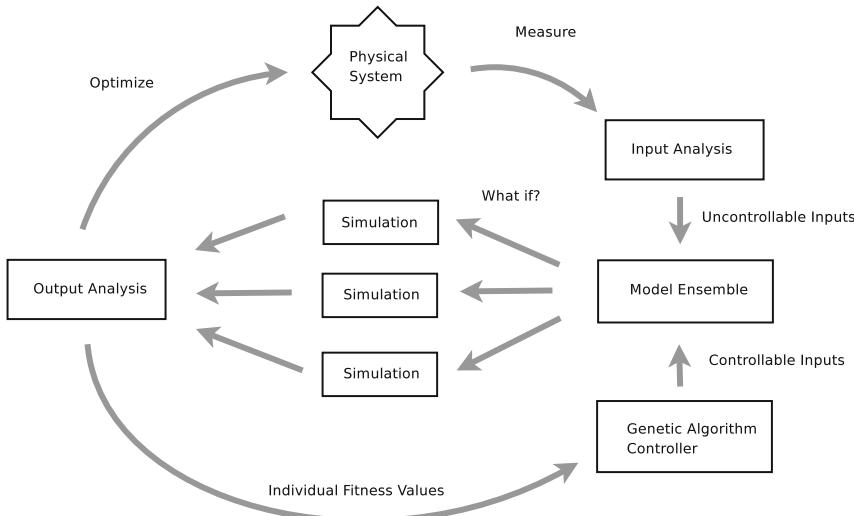


Fig. 2.4 Symbiotic adaptive multisimulation.

2.4.1.2 Combined Model Ensembles

In SAMS, a combined model ensemble (CME) is a specification for conducting a series of simulation experiments involving a single individual from a population of potential system configurations. The goal of these experiments is to test an individual in multiple possible environments. Each simulation experiment examines the individual in the context of a set of uncontrollable factors representing a single possible environment. An objective fitness of the individual for a particular environment is obtained as an output from a simulation experiment. The resulting fitness values from all experiments with the individual in each respective environment are then averaged together to obtain an overall fitness for the individual that is used to determine its probability for reproduction and survival within the genetic search.

The layout of a CME is shown in Table 2.1. The possible environments with which the individual is to be tested are determined by a PME of uncontrollable factors making up the left-hand side of the table. Copies of the individual are paired with each row of the PME. A single row of the CME thus includes both a sampled set of values from the uncontrollable factor distributions and a copy of the individual that specifies the settings of the controllable model factors. Taken together, these two sets of elements form a single plausible model described by the entire row of values in the CME.

Note that for a given CME, the same individual is paired with each row of the PME. Furthermore, the same PME is combined with each individual in the population to create a set of unique CMEs, one for each individual. Since the same PME is used in each CME, each individual is evaluated using the same environmental conditions. In essence, the uncontrollable factor settings of the PME, which change each time a new PME is created, become a dynamic fitness landscape.

Table 2.1 A combined model ensemble of n models is composed of a partial model ensemble of m sampled variables, X , and an individual of ℓ genes, G . Each row represents a single model for which one or more simulation replications should be performed. Note that the same individual is used in each model.

Uncontrollable factors				Controllable factors			
X_{11}	X_{12}	\dots	X_{1m}	G_1	G_2	\dots	G_ℓ
X_{21}	X_{22}	\dots	X_{2m}	G_1	G_2	\dots	G_ℓ
\vdots	\vdots	\ddots	\vdots	\vdots	\vdots	\ddots	\vdots
X_{n1}	X_{n2}	\dots	X_{nm}	G_1	G_2	\dots	G_ℓ

2.4.2

Local Adaptation: First-Order Change via Particle Swarm Optimizer

Self-organization is a kind of aggregate behavior that is often associated with complex adaptive systems. Self-organization as a property of an engineered system has been described as being one in which individual agents or units respond to local stimuli to achieve through a division of labor the efficient performance of some task (Collier and Taylor, 2004). The collective efficiency of task performance must be greater than what could be accomplished individually. Additionally, self-organization involves the creation of an equilibrium state that arises from the local interactions of agents Namatame and Sasaki (1998). This equilibrium may be achieved through either competition or cooperation.

In SAMS, particle swarm optimization (PSO) is used for implementing the learning element of adaptive agents. A broad introduction of PSO can be found in Kennedy and Eberhart (2001). We use PSO as a continuous numeric optimization technique in which a potential solution to a problem is characterized as a point in some n -dimensional space, with the number of dimensions being equal to the number of decision variables. As the name suggests, PSO uses a population of potential solutions. These solutions “fly” through the problem space over time. As each particle moves through the problem space, it records the best solution, p_{best} , that it has found so far as well as the best solution, g_{best} , discovered by the other members of the swarm. Particles tend to gravitate toward these two positions over time as they search for better solutions. This strategy corresponds to local adaptation (e.g., second order of change).

In Bratton and Kennedy (2007), a precise specification of the canonical PSO is given. In this specification, each dimension d of the velocity vector v of a particle i is calculated according to Eq. (2.1). In this equation, it can be seen that v is a function of the particle’s current position x , personal best p_i , and global best p_g . A constriction factor χ is included to prevent explosive growth of the particle’s velocity. In addition, randomness is included by ε_1 and ε_2 , which are independent uniform-

Algorithm 2.1 A genetic algorithm.

```

1: pop[m] ← createInitialPopulation()
2: for i = 1 to m do
3:   pop[i] ← calculateFitness(pop[i])
4: end for
5: repeat
6:   children[n]
7:   for i = 1 to n do
8:     parents[2] ← selectParents(pop)
9:     children[i] ← crossover(parents)
10:    children[i] ← mutatation(children[i])
11:    children[i] ← calculateFitness(children[i])
12:  end for
13:  pop[m] ← selectSurvivors(pop,children)
14: until termination = true

```

ly random variables that govern cognition and social influence, respectively. The constants c_1 and c_2 affect the relative contributions from the cognition and social components. These components also appear in many sources listed as φ_1 and φ_2 .

$$v_{id} = \chi(v_{id} + c_1 \varepsilon_1(p_{id} - x_{id}) + c_2 \varepsilon_2(p_{gd} - x_{id})) . \quad (2.1)$$

2.4.3

The Learning Layer: Genetic Search of Potential System Configurations

SAMS evolves potential system configurations for use within the physical system. Unlike the creation of a PME, however, a search of the space of potential system configurations requires the exploitation of a different type of system information, namely, the performance characteristics of the configuration. Evolutionary algorithms exploit this type of information to improve a search as they are well suited for optimization problems (DeJong, 2006). Among evolutionary algorithms, GAs have a feature useful to SAMS, which is the distinction between genotypic and phenotypic representation. A genetic encoding of the characteristics of a potential system configuration allows the same system to be interpreted in different ways and in varying levels of detail.

The structure of a typical GA is shown in Algorithm 2.1. An initial population of m individuals is randomly created and the fitness values of the resulting individuals are evaluated. The algorithm then enters a loop in which successive generations of individuals are evolved. An inner loop continues until a group of n children are created. Two individuals are chosen to be parents. Crossover combines the characteristics of both parents to create a new child. A mutation operator is then applied that may modify the child with characteristics not possessed by either parent. The fitness of the new child is then evaluated.

Once the children have been created, the total population of individuals is then reduced back to m . Depending on the design of the GA, survivor selection may or may not involve direct competition between parents and children. The evolutionary

process continues until some termination condition (such as a fixed number of fitness evaluations) is reached.

2.4.4

SAMS Component Architecture

SAMS implementation, designed to operate in mission-critical environments, is based on an independent component architecture in which the individual components of the system could execute in parallel and communicate via message passing (Braude, 2004). This could be especially useful for entities operating in the physical environment that would likely not have the hardware resources to process simulations. Rather, these entities would only need some means of measuring the physical system and sending those measurements to the components running the simulations. Figure 2.5 illustrates the essential components of a SAMS system. Observations from the physical system are passed to an input exploration component (IEC) responsible for conducting input analysis and selecting appropriate distributions for the uncontrollable model factors. Samples from these distributions are

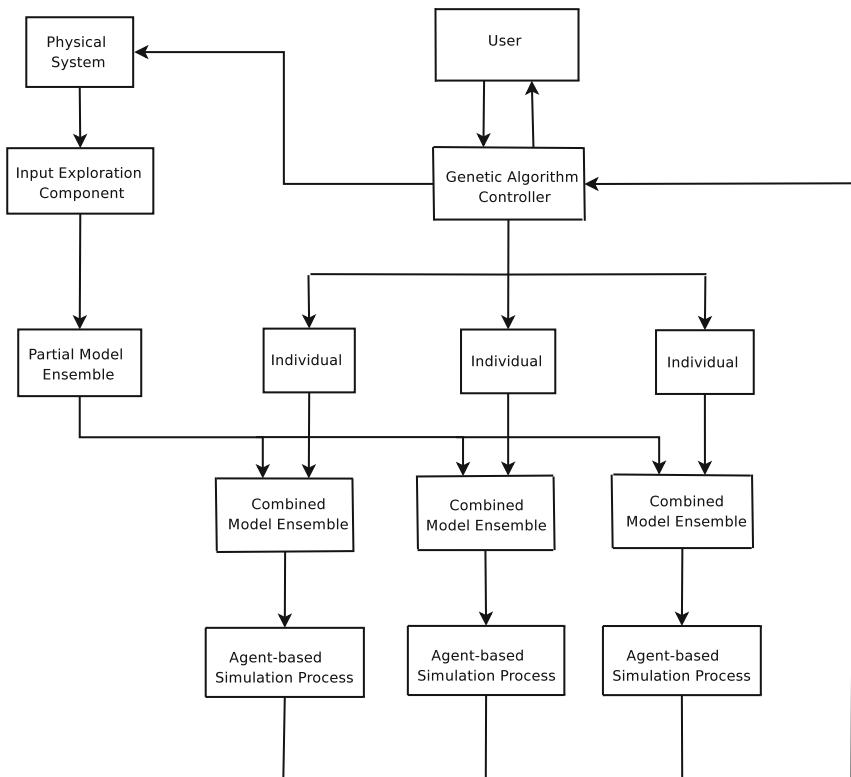


Fig. 2.5 Components of the SAMS framework.

then used to create a PME, a copy of which is integrated with each individual to form one CME for each agent-based simulation process (ASP).

The genetic algorithm controller (GAC) is responsible for evolving the population of individuals that are used to form the CMEs and a number of agent simulation instances (ASPs) to simulate CMEs in parallel. Ideally, each ASP is mapped to one or more CPU cores. If the population used by the GAC is large, or if hardware resources are limited, multiple ASPs can run on a single core. If there is an excess of hardware, a SAMS implementation should be capable of offloading models within the CME to multiple CPU cores. Outputs from the ASPs take the form of objective fitness values of the individuals that have been averaged across all of the replications specified by the CME. These are passed to the GAC, which then assigns the fitness values to the simulated individuals before continuing execution of the GA. Ideally, it should be possible for a user to interact with the GAC in real time to examine the individuals generated and possibly seed new configurations to the population.

2.4.4.1 Component Interaction Dynamics in SAMS

The sequence of operations that occur within the main execution loop of a SAMS implementation are shown in a UML sequence diagram in Figure 2.6. Active components and processes, displayed as boxes across the top of the diagram, include the GAC, agent-based simulation processes (of which several run simultaneously), an IEC, and the physical system. An initial message is passed from the GAC to the IEC requesting a new PME. The IEC takes observations from the physical system and uses them to produce uncontrollable factor settings that are incorporated into the PME. The PME is then passed to the GAC, which uses them to create a CME for each ASP.

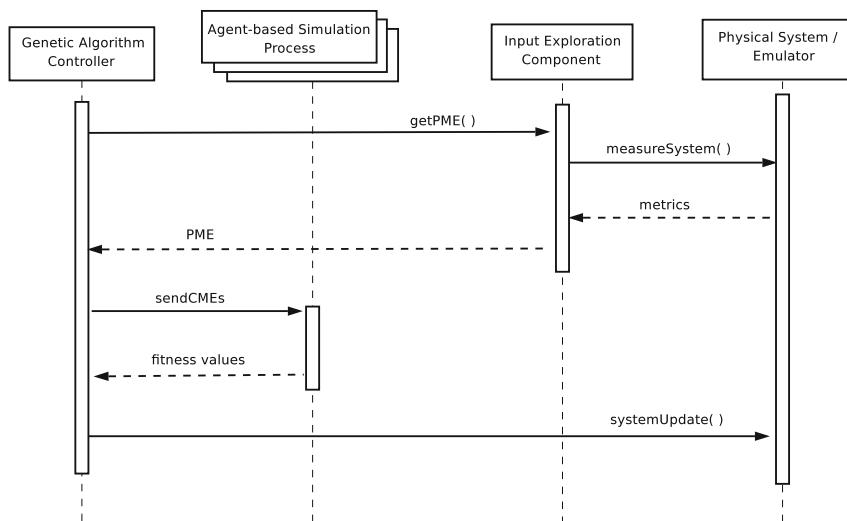


Fig. 2.6 Component interaction in SAMS.

When created, the CMEs are passed to the ASPs and simulated in parallel. Simulation results are examined on a per-CME basis so that a fitness can be assigned to a given individual based on its performance against the environmental settings specified in the PME. After fitness values have been assigned to individuals, the GAC evolves the population. Individuals with higher fitness are given preference for producing offspring that are created using genetic operators of crossover and mutation. Due to the need to maintain a naturally parallel structure for the algorithm, large numbers of children should be generated before selecting survivors.

In addition to evolving the existing population, the GAC also selects the most fit individual of each generation to update the physical system's configuration. This continual process of dynamically updating the physical system helps ensure that the physical system is responding correctly to observed changes in the environment.

2.4.4.2 Design and Implementation

A SAMS application was developed in order to examine the effectiveness of the SAMS methodology. The application is capable of spawning multiple ASPs with which to simulate plausible models, while the GAC and system emulator execute in their own CPU process. The application was written in C, providing exceptional performance, and offers potential for scalability as interprocess communication only occurs during the transmission of CMEs and fitness values for individuals. Each run of the parallel application requires several thousand model replications, making experimentation on a single CPU impractical. As an alternative, experiments with the application were performed on an SGI Altix 350 shared memory supercomputer operated by the Alabama Supercomputer Authority. This system has a total of 144 CPU cores, although the job queuing system typically allocates only a fraction of these to the user. While shared memory is not a requirement of the application, the Altix is capable of running applications that use MPI, and was therefore chosen over the Cray XD1 supercomputer.

2.4.4.3 Software Design and Implementation

The SAMS application, including the agent-based simulation with which it operates, was developed using functional decomposition and is integrated into a single executable. The application was developed and compiled using Suse Linux GCC 3.3.3 on the Altix. It was intended to be portable to other platforms and only has dependencies with MPI and the standard C libraries. MPI allows the user to run the application with a variable number of processes. In the current version, a separate ASP is allocated to all but one of the processes which is reserved for the GAC. In addition to the parallel application, a standalone version of the UAV simulation has been developed for testing. This version includes a rudimentary 2-D graphical display that makes use of OpenGL and the Simple DirectMedia Layer for rendering in the X-Window system. The purpose of this display is to allow a simple visual examination of UAV movement behavior within the simulation. Figure 2.7 presents the screenshots of UAV simulation used as a testbed in this study.

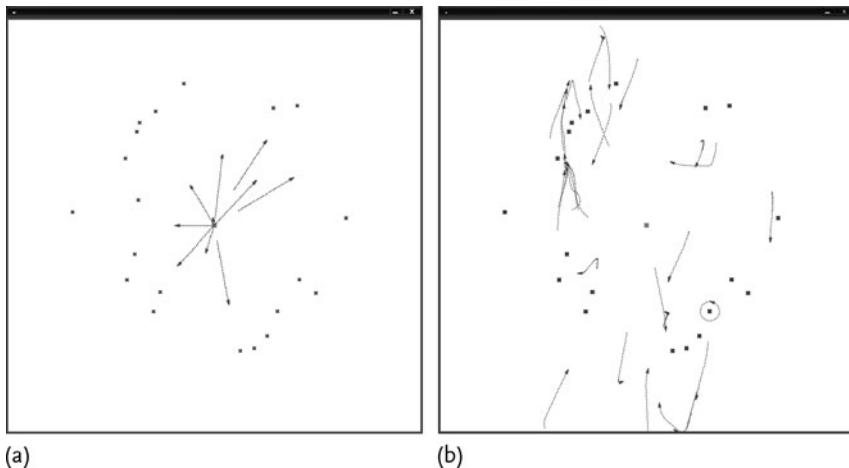


Fig. 2.7 UAVs begin cooperative behavior to mass their fires on individual targets. (a) Initial State; (b) Attack.

2.4.4.4 System Emulator

Since experimentation with physical UAVs was not possible for this study, a system emulator was incorporated into the parallel application. The emulator is an additional simulation that resides in the same CPU process as the GAC. Unlike the simulations in the ASPs, the input factors of the emulator are not modified by the creation of a CME. The emulator is assumed to have the true settings of the physical system. These settings are unknown to the IEC, GAC, and ASPs. However, the IEC does begin with an initial distribution for each uncontrollable factor that includes the true value for the respective setting in the emulator.

2.5

Case Study: UAV Search and Attack Scenario

To perform an assessment of the potential of the SAMS methodology, a model based on the features of complex adaptive systems was integrated into the parallel application. The field of autonomous UAV cooperation provides a natural environment from which to create such a model. To this end, an agent-based model of an autonomous UAV team in a search-and-attack mission was developed. The UAVs in this model interact through local communication only. There is no global system of coordination or prior intelligence of targets. A rule-based approach to UAV movement is implemented. This set of movement rules, which is based on general knowledge of the problem domain, results in a robust performance element for UAVs capable of both independent and cooperative actions. These rules are implementations of the steering behaviors described in Reynolds (1999). Similar implementations of these steering behaviors have been featured in a number of autonomous UAV studies including Price (2006) and Crowther (2004).

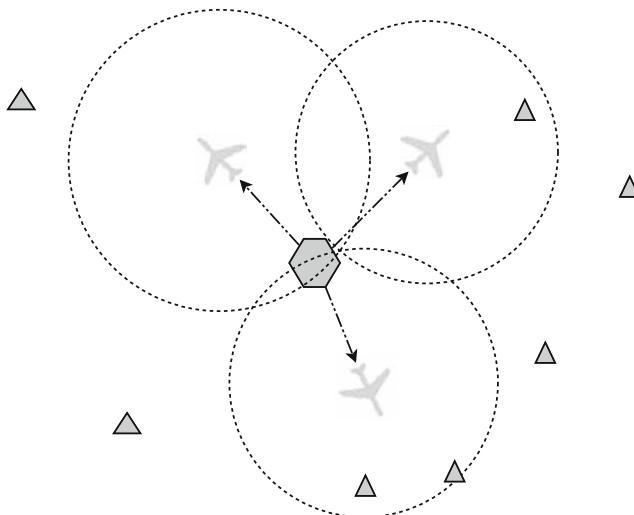


Fig. 2.8 The search-and-attack scenario shortly after simulation start. UAVs depart from the centrally located airbase (represented by a hexagon) with uniformly random initial movement vectors. The sensor envelopes of individual UAVs are represented by dotted circles.

The search-and-attack scenario takes place in a 2-dimensional space of equal dimensions. UAVs begin play from a base located at the center of the map. Targets are distributed randomly across the map and their positions are initially unknown to the UAVs. Once they are launched, the UAVs must find and destroy all of the targets as quickly as possible. Figure 2.8 depicts the opening simulation steps in which UAVs are in the process of launching from the base (represented by a gray hexagon) and sweeping the map. Targets (represented by gray triangles) within the sensor envelope of a UAV have a chance of being discovered while those outside remain hidden.

Progression of the model is simulated through time-stepped execution, dividing simulation time into a number of equal-size increments. During a simulation step, each UAV may act. The results of these actions are updated synchronously and may affect actions performed by other UAVs in the same time step.

2.5.1

Input Factors

The UAV model has numerous variables that can potentially be parameterized. However, in order to more easily assess the model (and the effectiveness of SAMS as a whole), the number of parameters has been kept small. It was theorized that propagation of target information within the UAV team as well as the command structure of the team itself would play decisive roles in determining success or failure in a given simulation run. These aspects of the model were parameterized

Table 2.2 Inputs include both uncontrollable and controllable factors. Uncontrollable factors represent enemy characteristics and environmental conditions while controllable factors represent aspects of the UAV team that may be modified.

Input factor	Controllable?	Range
Leader	Yes	0,1
Cooperation Threshold	Yes	$0 \leq x \leq 1$
Maximum Communication Range	No	$0 \leq x \leq 100$
Maximum Target Visibility	No	$0 < x \leq 100$
Toughness	No	$1 \leq x \leq 10$

as shown in Table 2.2. The leader and cooperation threshold parameters are controllable factors evolved by the GAC. Maximum communication range limits the extent of local communication among UAVs and is intended to loosely model interference from ground clutter or active radio jamming. Maximum target visibility models both natural target concealment provided by terrain as well as active camouflaging techniques. Toughness determines the amount of damage a target can sustain from UAVs before being eliminated. These last three factors are initially unknown to the SAMS application. Estimates of their actual values are produced by the IEC.

2.5.2

Agent Specifications

Each UAV, U , within the model possesses a number of member variables and constants. These are listed in Table 2.3 and represent both the physical state of a UAV as well as its internal cognitive state during a given time step. The physical state includes the UAV's movement characteristics. The cognitive state of a UAV is based on an associated particle in a distributed PSO. Each variable or constant has a data type, some of which are vectors. The initial values for some of these variables apply only to simulations that occur when the SAMS application is initiated. Subsequent simulations that are created in ASPs may have different values depending on observations made within the emulator. These variables are indicated with a \dagger . Some variables are also directly evolved by the GAC and thus their initial settings are based on the values encoded within an individual's genotype.

The member variables of the base are shown in Table 2.4. The base provides two service queues: fuel and launch, each with one service resource. In the ASP simulations produced by the first generation of CMEs, the fuel queue is empty and the launch queue is full. This reflects the starting condition of the model in which UAVs are considered to be fueled but not yet launched. Later ASP simulations may have different initial settings for these queues to reflect the servicing of UAVs in the emulator.

Table 2.3 The member variables of a UAV affect both its movement physics in 2-D simulation space and the movement of its particle within the space defined by its decision variables.

Name	Symbol	Type	Initial value	Constant?
mass	m	double	2	yes
position	r	double[2]	$(0, 0)^\dagger$	no
velocity	v	double[2]	$(0, 0)^\dagger$	no
maximum force	$ F _{\max}$	double	1.0	yes
maximum speed	$ v _{\max}$	double	0.5	yes
fuel	f	double	1.0^\dagger	no
leader flag	l	Boolean	by genotype	yes
cooperation threshold	ct	double	by genotype	yes
particle position	x	double[2]	$U(0, 1)^\dagger, U(10, 50)^\dagger$	no
particle velocity	v	double[2]	$U(-0.25, 0.25)^\dagger, U(-10, 10)^\dagger$	no
personal best	p	double[2]	x^\dagger	no
local best	p_1	double[2]	x^\dagger	no

Table 2.4 In addition to its location, the airbase consists of a single-server fuel queue and a single-server launch queue. All UAVs are considered fueled but not yet launched when a simulation begins, hence the initial settings of those queues. A \dagger indicates an initial setting that is dependent on the emulator state.

Name	Symbol	Type	Initial value	Constant?
position	r	double[2]	$(50, 50)$	yes
fuel queue	S_{fuel}	list	empty †	no
launch queue	S_{launch}	list	full †	no

2.5.2.1 UAV Movement

Each UAV is modeled as a point with mass, m , maximum velocity, $|v|_{\max}$, and maximum thrust, $|F|_{\max}$. Thrust may be applied in any direction in an effort to steer the UAV toward a desired location, but the UAV's mass decreases maneuverability due to inertia. The basic rule for UAV movement is *seek*, which causes a UAV to move toward some specified position on the map. In addition to seek, the model has the following basic movement rules: *offset seek*, *arrival*, *orbit*, *cohesion*, *separation*, and *alignment*. Offset seek allows a UAV to approach a position from a slight offset, similar to the way in which an aircraft might maneuver into a strafing run against a ground target. This is a modification of the *offset pursuit* steering behavior described in Reynolds (1999). The arrival rule is identical to seek except that it causes the UAV to decelerate as it approaches a position. This is useful for simulat-

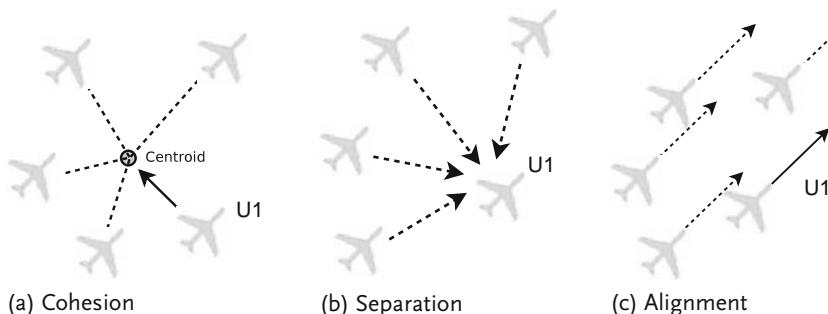


Fig. 2.9 UAV movements.

ing a UAV's landing approach to the base for refueling. It is an implementation of a steering behavior of the same name in Reynolds (1999). The orbit rule is a modification of the *path following* behavior found in Reynolds (1999) that allows a UAV to orbit a position from a desired distance.

The cohesion, separation, and alignment rules are useful for coordinating group behavior among UAVs. Cohesion causes a UAV to move toward the average position, or centroid, of a group of other UAVs. In contrast, separation causes a UAV to move away from other UAVs. Separation is scaled so that the separating force becomes stronger as a UAV gets closer to another UAV. Finally, alignment causes a UAV to align its velocity vector with the velocities of other UAVs. When taken together as a linear combination with weight constants for each rule, the advanced movement rule, *flocking*, occurs. These rules are shown in Figure 2.9.

Another advanced movement rule implemented in the model is leader *following*. Like flocking, this rule will affect group coordination among UAVs. The flocking rule, however, is completely decentralized, relying on no commands or guidance from a leader. The movement and direction of a flock are therefore emergent, arising from the interaction of movements made by individual flock mates. In leader following, UAVs will attempt to steer toward a position immediately behind another UAV that is designated as a leader. The separation rule will also be incorporated in leader following to prevent UAVs from clustering too tightly. This is shown in

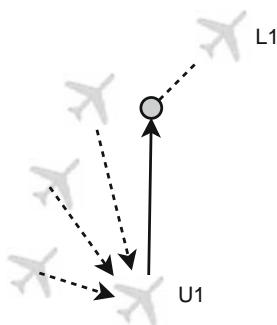


Fig. 2.10 In leader following, a cooperating UAV (U1) attempts to move toward a position behind the nearest leader (L1). This movement is somewhat inhibited by separation from other cooperating UAVs.

Figure 2.10. The leader is completely unaware of the other UAVs following it and behaves exactly as if it were acting independently.

2.5.2.2 UAV Communication

UAVs coordinate their behavior through local communication which is limited by the Maximum Communication Range input factor. UAVs receive all messages that are transmitted as long as the distance from the sender is less than this range. All UAVs from whom a given UAV receives messages are considered to be within its neighborhood, which affects its cooperative behavior. A message transmitted from a given UAV includes a list of targets detected by that UAV as well as the best position of the UAV's particle, p . UAVs do not retransmit messages that they have received, thus resulting in a single hop wireless network as shown in Figure 2.11.

2.5.2.3 Target Distribution

Targets are placed on the map according to a randomly distributed polar coordinate relative to the base in the map's center. The angle is uniformly distributed from 0 to

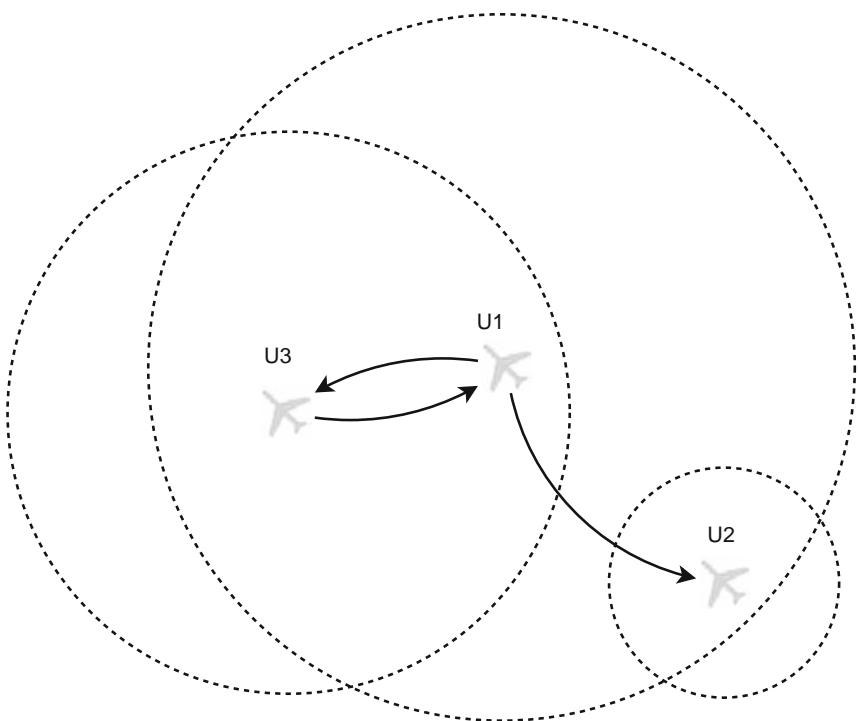


Fig. 2.11 Communication range, indicated by dotted lines, affects the ability of UAVs to send messages, indicated by solid lines. In a single hop network, U2 can receive data from U1 but not from U3. In a multihop network, U1 can act as a transceiver, passing a message from U3 to U2.

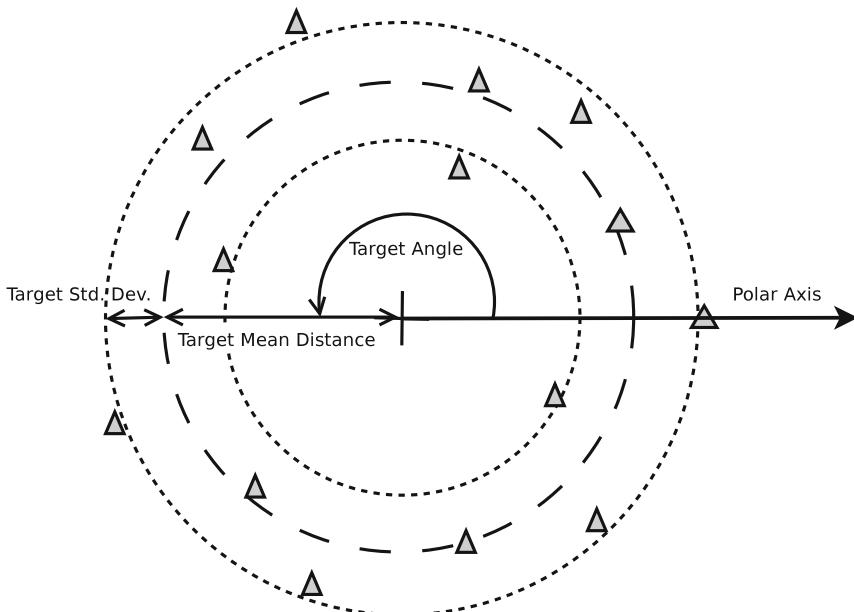


Fig. 2.12 Targets are positioned at a polar coordinate relative to a distribution point represented by the crosshair in the center of the diagram. The target angle is distributed uniformly in the interval $[0, 2\pi]$ in radians. The radial coordinate, r , is normally distributed according to the model settings of mean target distance and target standard deviation.

2π radians. The distance is normally distributed with the model setting mean target distance determining the μ parameter and target standard deviation determining σ . As shown in Figure 2.12, this results in targets positioned in a ring encircling the base.

2.5.2.4 Target Detection

During each simulation time step, a UAV may discover a target either by making a successful detection attempt or by receiving the target's location data via a message from another UAV. Detection attempts are possible within a maximum sensor range of a given UAV, defined by the maximum target visibility model setting, which is also an uncontrollable input factor. The probability of a given UAV detecting a given target decreases linearly as the distance between the UAV and the target increases. Equation (2.2), below, illustrates this rule.

$$p = 1 - \frac{\text{Target Distance}}{\text{Maximum Target Visibility}} . \quad (2.2)$$

Target detection must be reattempted during each simulation time step so it is possible that previously detected targets may revert to an undetected status. A UAV that detects a target will include the target's position data in its communication

message during that same time tick. At the end of the time step, however, all targets are cleared from a UAV's memory, so it is not possible for a UAV to receive target information from another UAV in one time step and then retransmit that target information in the next time step.

2.5.2.5 Combat

In order to attack a target in a given time step, the UAV must have either successfully detected the target in the same step or received the target's position data from another UAV that made a successful detection attempt. UAVs always attack the nearest known target once during each time step regardless of the direction of the target relative to the UAV. However, only known targets whose distance to the UAV does not exceed the weapon range model setting may be attacked. Attacks always hit and inflict a normally distributed amount of damage with parameters determined by model settings of average weapon effect and weapon wfect standard deviation. Once a target's toughness variable has been reduced to 0 or less from UAV attacks, it is considered eliminated and removed from play. Targets do not repopulate the map once removed. In addition, targets do not attack UAVs.

In some studies of autonomous UAV teams, such as that by Lua et al. (2003), the notion of multipoint attack has been of interest. This is essentially the idea of flanking an opponent. This concept has been implemented in the UAV model to reflect an additional benefit of cooperative behavior. When at least two UAVs attack the same target in the same time step, and the smallest angle between them relative to the target is at least $2/3\pi$ radians, a multipoint attack bonus is applied. These conditions are illustrated in Figure 2.13. When applied, this bonus doubles the damage inflicted by every UAV attacking the target in that time step.

2.5.2.6 Supply

A simple model of resupply is implemented in the UAV model to penalize inefficient UAV behavior. While UAVs are considered to have unlimited ammunition, their fuel supply is finite. This supply is initially set to 1. A certain amount of fuel is consumed each time step that a UAV is in flight according to Eq. (2.3), where m is

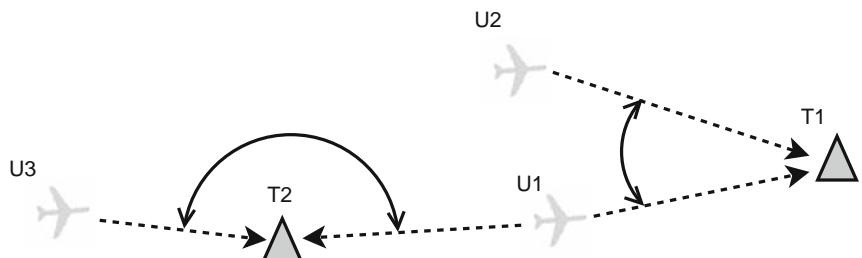


Fig. 2.13 A multipoint attack bonus occurs when the shortest angle between any two attacking UAVs relative to the target is at least $2/3\pi$ radians. In this case, a bonus occurs if U1 attacks T2 in conjunction with U3, but not if U1 attacks T1 instead.

the mass of the UAV, v_t and v_{t-1} are the UAV's velocity vectors during the current and previous time steps, and C is a constant equal to the Fuel Consumption model setting.

$$\text{fuel}_t = \text{fuel}_{t-1} - (m * (|v_t - v_{t-1}|) * C). \quad (2.3)$$

Once the fuel supply of a UAV is exhausted, it must return to the base for resupply. (UAVs are assumed to have a spare capacity for the return trip.) Once at the base, it is placed in the base's fuel queue. The service time for the fuel queue is a number of time steps uniformly distributed by the model settings minimum fuel time and maximum fuel time. Once serviced, the UAV is placed in the base's launch queue, also with a uniform distribution and associated model settings for the distribution parameters. Both queues are single server, and while a UAV is at the base it may not send or receive messages from UAVs in flight.

2.5.2.7 Decision Making Behavior of UAVs

The decision element incorporates the movement rules into a decision tree. As shown in Figure 2.14, the various movement rules occupy the leaf nodes within the tree. The decision process begins at the top of the tree. If a UAV's fuel is exhausted, the arrival rule is selected with the base as the desired movement location. Otherwise, the UAV must choose whether or not to act cooperatively or independently during that time step.

This decision is based on a cooperation threshold, ct , and a current cooperation value, stored in α . The value of ct is fixed for the duration of a simulation run, as it is determined by the genotype of the simulated individual. The cooperation value

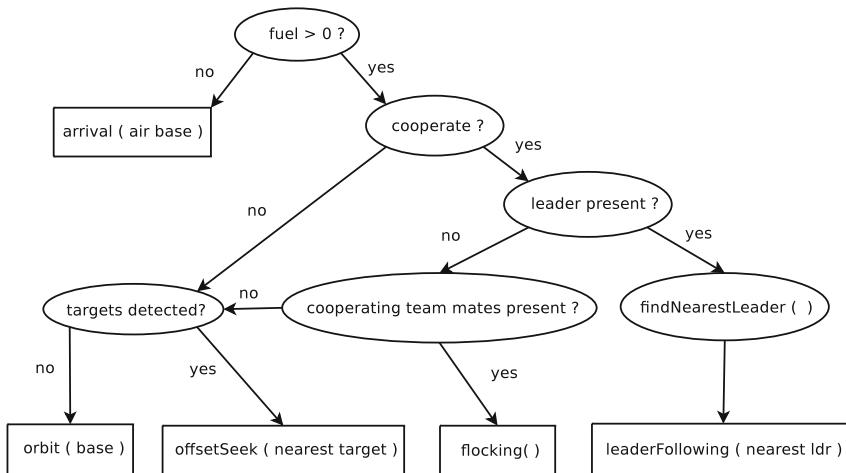


Fig. 2.14 Communication range, indicated by dotted lines, affects the ability of UAVs to send messages, indicated by solid lines. In a single-hop network, U2 can receive data from U1 but not from U3. In a multihop network, U1 can act as a transceiver, passing a message from U3 to U2.

is modified by the learning element. If cooperation is not greater than ct , the UAV will choose cooperative behavior. If a leader is within the UAV's neighborhood, leader following will occur. Otherwise, the UAV will flock with other cooperating UAVs if any are present in its neighborhood. When independent behavior is chosen and known targets are present, the UAV will perform the offset seek rule with the nearest target as its destination. If no known targets are present, the UAV will perform the orbit rule using a Base Distance variable stored in x . The behavior described in the UAV decision tree applies both to leader and nonleader UAVs. However, a leader cannot act as a leader and perform cooperative behavior simultaneously. Therefore, a UAV with the leader flag is not recognized by other UAVs as a leader during any time step in which it chooses to cooperate.

2.5.2.8 Design of the UAV Learning Element

The learning element of a UAV has two features. The first feature is a distributed particle swarm that adjusts two decision variables: cooperation and base distance. Each UAV has a single particle associated with it. The position, x , and velocity, v , of this particle are updated periodically, according to a specified sampling interval. During this update, if other UAVs are within communication range, the UAV will exchange personal best values, p , and potentially obtain a new local best value, p_l . The fitness of a particle's position varies from 0 to 1 and is determined by maximizing the objective function Eq. (2.4). In this function, a is the number of attacks performed by the UAV, k is the number of the UAV's kills, and d is the number of target detections propagated by the UAV to itself and its teammates. These variables are counted during the current sampling interval only. They do not accumulate from one interval to the next.

$$\max(x) = 1 - \frac{1}{a^{(k+1)} + 2} - \frac{1}{d + 2}. \quad (2.4)$$

The second feature of the learning element is a string of 4 bits that models a rudimentary cognitive bias, or "personality", and is generated by the GAC of the parallel application as a part of the UAV team's specification. The first bit sets the leader flag value, l . This value determines whether or not other UAVs consider the UAV to be a leader. The other three bits determine the cooperation threshold, ct , which determines how easily the UAV decides to cooperate with other UAVs. Each value of these three bits represents one of eight values uniformly ranging from 0 to 1. Thus, when the cooperation decision variable stored in x is less than ct , it will choose to cooperate with other UAVs.

2.6

Validation and Preliminary Experimentation with SAMS

To assess the suitability of the UAV model for use in the parallel application, a number of experiments were performed to gain insight into the interactions between various controllable and uncontrollable factors. The first set of experiments was

conducted for face validity and eliminated factors that were insignificant to the performance. In CASs behavior transitions naturally between equilibrium points through environmental adaptation and self-organization. These preliminary experiments are also used to determine if the model behavior exhibits this specific characteristic of CASs.

2.6.1

Face Validity of the UAV Model

In keeping with the theory that a useful model should force autonomous UAVs to adopt a balance between independent and cooperative behavior, the intent is to examine the interactions between the number of leaders in a team, the cooperation thresholds of its members, and the communication range. Toughness is also included in the experiments as this affects the length of a mission and therefore the amount of time the team has to adapt. Given this intent, a 2^4 factorial experimental design is imposed. Low and high settings for toughness and communication radius are {1, 10} and {6, 60}, respectively. Settings for the number of leaders are {0, 4}. It is posited that leaders would need lower cooperation thresholds than their nonleader counterparts, since only noncooperating leaders are recognized by other UAVs. Therefore, leaders and nonleaders are given different cooperation thresholds for each setting, low or high. Leader settings are defined as {0.05, 0.1}, while non-leaders use settings of {0.5, 0.9}. The objective for a mission is to minimize the number of time steps needed to eliminate all targets in play. The observations for this metric are listed under the Average Result column. The number of time steps required to eliminate all targets was selected as the performance objective because it subsumes the goals of reconnaissance, combat, and economy of resources. Simulations running longer than 5000 time steps were thrown out and repeated. For each experiment, 30 replicates were obtained in an attempt to improve the odds that the average Mission Complete scores would be normally distributed.

As can be seen in Table 2.5, certain UAV team configurations appeared to perform better in certain settings. Teams with no leaders and low cooperation values tended to perform better when the communication range was increased. Conversely, a slight advantage appeared to fall toward teams with leaders when the communication range was decreased. For all replicates, target max visibility and weapon range were set to 5. Average weapon effect and weapon effect standard deviation were set to 0.05 and 0.03, respectively.

One interesting phenomenon observed in some of the experiments was the behavior of the UAVs' average cooperation values due to the influence of the distributed PSO. Figure 2.15 illustrates a single replication from one set of experiments, which was not included in the original experimental design, but appears instructive. This case involved a high communication range setting of 60 and one leader UAV. As discussed earlier, the distributed PSO evaluates the PSO objective function and updates its decision variables (including cooperation and base distance) once per 50 time steps. This period is referred to as the sampling interval, shown across the bottom of the plot. Thus, at sampling interval 10, 500 time steps have

Table 2.5 The initial tests of the UAV model involved four model factors examined with high and low values for each factor. Toughness and communication range settings were {1, 10} and {6, 60}, respectively. Settings for the number of leaders in a team

were {0, 4}. Cooperation threshold settings differed depending on whether a UAV was a leader or nonleader. These settings were {0.05, 0.1} for leaders and {0.5, 0.9} for nonleaders.

Treatment No.	Uncontrollable factors		Controllable factors		Mission complete	
	Tough.	Comm. range	Leaders	coop.	Average result	Standard dev.
1	Low	Low	Low	Low	462.60	74.28
2	Low	Low	Low	High	468.57	89.52
3	Low	Low	High	Low	433.20	50.32
4	Low	Low	High	High	446.93	79.44
5	Low	High	Low	Low	468.30	132.48
6	Low	High	Low	High	643.33	225.85
7	Low	High	High	Low	473.47	109.47
8	Low	High	High	High	556.87	119.6
9	High	Low	Low	Low	1826.63	139.56
10	High	Low	Low	High	2350.9	293.15
11	High	Low	High	Low	1777.24	129.68
12	High	Low	High	High	1995.90	219.96
13	High	High	Low	Low	1769.18	246.63
14	High	High	Low	High	1926.7	453.9
15	High	High	High	Low	1885.40	526.5
16	High	High	High	High	2475.71	814.61

elapsed in the simulation. This plot tracks the change in the average of all cooperation values within the UAV team across a single model replication. In this case, by about the tenth sampling interval, the distributed PSO had evolved the team away from cooperative behavior. This coincided with a dramatic drop in the target population. Although possibly the effect of random chance, it may be that this change allowed the team to spread out sufficiently to locate and destroy the targets. This case is potentially instructive because it indicates that the communication range may have a significant interaction with the number of leaders within a team and the cooperation thresholds of its members.

The behavior shown previously in Figure 2.15 contrasts with what occurs when the communication range is set to a low value. Figure 2.16 depicts that when the communication range is set to 6, cooperation is more likely to occur. In this case, the average value of the PSO cooperation value stayed relatively close to 0.5. The transitioning behavior to equilibrium in both cases is typical of CAS systems and helped us instill confidence in the learning mechanism.

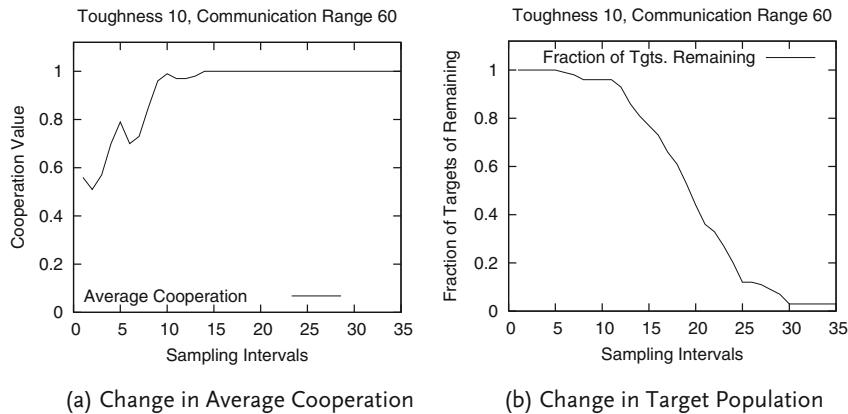


Fig. 2.15 Effect of cooperation on target population with high communication range – during a single model replication, UAVs choose not to cooperate when long communication ranges are in effect and only one leader is present. In this case, the cooperation threshold was set to 0.9 for all UAVs except for the leader, which had a cooperation threshold of 0.1.

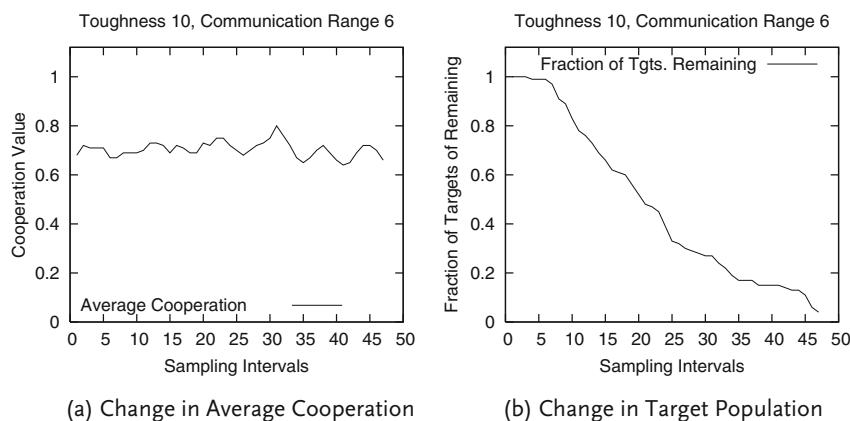


Fig. 2.16 Effect of cooperation on target population with low communication range – during a single model replication, UAVs are more likely to cooperate when low communication range settings are in effect. This example involved a cooperation threshold of 0.9 for all nonleader UAVs and 1 leader with a cooperation threshold of 0.1.

2.6.2

Experiments with the Parallel SAMS Application

The goal of computational experimentation for this thesis was to provide an initial understanding of the potential of SAMS as an S2 methodology. To this end, it was hoped that successful experimentation would help to answer a central question:

- *Can the use of symbiotic simulation through ensembles of plausible models improve a physical system's performance?*

2.6.2.1 Emulator Objective and Individual Fitness

Experimentation with the parallel SAMS application involved the use of a system emulator, rather than an actual physical system. This emulator is itself a simulation with the same structural model of autonomous UAV behavior used by the ASPs. It only differs in terms of the values used for its input factors and in that its controllable factor settings are not fixed for the duration of its execution. Similar to the model replications used for face validity, the performance objective of the emulator is to minimize the number of time steps required to eliminate all targets. Therefore, to synchronize the performance of the simulations executed in the ASPs with that of the emulator, the fitness of an individual is defined to be the average number of time steps required to eliminate all targets across all replications specified by that individual's CME.

The emulator is an ordinary UAV model simulation that receives controllable factor updates from SAMS. Therefore, it was possible to examine the performance of the parallel SAMS application by running an emulator with uncontrollable factor settings identical to those used in the experiments from Table 2.5. Since the emulator and standalone models use the same model structure, their performance can be compared as long their uncontrollable factors and fixed model settings are identical. Our position is that the emulator's system would have an advantage over the system in a standalone model. The controllable factors of leader flags and cooperation thresholds can be modified by SAMS to the benefit of the emulator, whereas the system in the standalone model has controllable factor settings that are determined in advance and fixed for the duration of its execution. In order to compare the performance of the emulator with SAMS against a standalone model, all fixed model settings such as weapon range, average weapon effect, weapon effect standard deviation, average target distance, and target standard deviation were set to the identical settings used in the experiments described in the previous section, as these settings were held constant across all treatments in that group. Furthermore, a subset of the treatments corresponding to one pair of settings of the uncontrollable factors was selected.

2.6.2.2 GA Design

One intent with SAMS is for the GAC to produce a large degree of exploratory behavior initially, followed by increased exploitative search behavior as the physical system changes. With this in mind, proportional selection was chosen as a parent selection operator for these experiments. The variation operators used for these experiments included one-point crossover, which obtains a single cut point that determines how the chromosomes will be divided and recombined to produce children, as shown in Figure 2.17. Note that one or both of the children shown may actually be created. For these experiments, however, two children were created for each crossover operation. The other variation operator, mutation rate, was set to 0.05 so

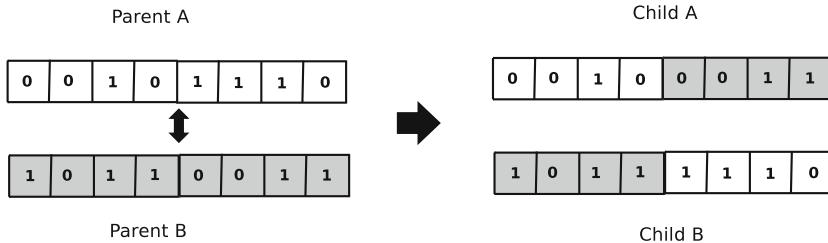


Fig. 2.17 One-point crossover: in one-point crossover, a single cut point is randomly selected along the length of each parent's chromosome. All genes to the left of the cut point are passed on to the child directly to the parent's right, while everything on the left is exchanged and given to the opposite child.

that, on average, 4 bits from each chromosome were flipped. Each individual represented 20 UAVs and thus is 80 bit long. In addition, a nonoverlapping survival model was selected so that all children survive while parents automatically die. This type of GA, which balances a relatively low selection pressure with low rates of variation from one-point crossover and low mutation, is suitable for encouraging the exploration of controllable factors early in the physical system's development.

2.6.2.3 Emulator Results

The major limitation of these experiments was that the IEC was not yet implemented. Therefore, certain assumptions were made regarding emulator/physical system measurement and PME generation. The work of the IEC was simulated within the experiments. At runtime, the simulated IEC is initialized with uniform random distributions for three uncontrollable factors: toughness, communication range, and target max visibility. Note that target max visibility was held constant across all treatments presented in Section 2.5.1, but it is considered one of the uncontrollable factors for these experiments as mentioned in Section 2.4.1. The true values used for these factors within the emulator were toughness 10, communication range 6, and maximum target visibility 5. These settings were identical to those used in treatments 9, 10, 11, and 12 in Section 2.5.1.

The parameterized distributions within the simulated IEC for the uncontrollable factors are arbitrarily selected to be uniform distributions of length 30 about each factor's true setting (truncated by zero as a minimum parameter). This included $U(0, 25)$ for toughness, $U(0, 20)$ for maximum target visibility, and $U(0, 21)$ for communication range. To simulate the effect of increasing information available from the IEC, the length of each distribution is divided by half during each iteration of the GAC's main loop, remaining centered about each true setting. For example, at the second main loop iteration (generation 2 in the GAC and time step 200 in the emulator) the distribution for communication range was set to $U(0, 13.5)$. Then, at the third iteration it was set to $U(2.25, 9.75)$ and so on.

At each iteration, a PME of 30 rows and 3 columns (one for each uncontrollable factor) was created. When combined with an individual, each CME therefore had

Table 2.6 The results of the parallel application compared to the best performing standalone model. The parallel application provides significantly improved performance on average, but with higher variance.

Experiment	Average time steps	Standard dev.
Emulator with SAMS	1597.5	164.34
Standalone model with best average fitness	1777.24	129.68

30 rows and 43 columns (3 columns for the uncontrollable factors and 40 columns representing 2 genes for each of the 20 UAVs in an individual). Each row, representing a single parameterized model, runs for one replication, resulting in 30 replications for each CME. A population of 20 individuals mapped to 20 ASPs was used. This resulted in 600 model runs during each iteration of the main loop. The emulator replications included, on average, 16 generations. Thus, on average, approximately 9600 model replications were performed by the ASPs for each replication of the emulator. More replications for each row of a CME and a larger population would have been desirable but were not used due to project time constraints.

For the experiment, 30 replications of the SAMS application on the Altix supercomputer were run. As can be seen in Table 2.6, the performance improvement of the emulator is noticeable in terms of the number of time steps for mission completion compared to the best performing UAV team configuration among the standalone model tests.

A possible explanation for the increased standard deviation in SAMS may be that the number of replications performed for each CME were simply insufficient to properly assess the fitness of the CME's respective individual. This might also explain the initial decrease in fitness in Figure 2.18 as the initial iterations of the main loop involve significant uncertainty in the uncontrollable factors. While these experiments seem encouraging, more experimentation is needed to explore this issue in detail. In particular, experiments with larger numbers of model replications need to be performed. Secondly, the possibilities for modification of the GA used by SAMS have barely been scratched. Much research is needed to identify what design features of a GA (or EA) contribute most to improved performance.

2.7

Summary

This chapter examines the need for dynamic model updating for symbiotic simulation of systems involving interacting agents with complex, nonlinear behavior. Our position is that these systems may not be effectively studied with traditional simulation and systems engineering techniques that rely on valid, authoritative models of the physical system. Instead, techniques such as multisimulation and explorato-

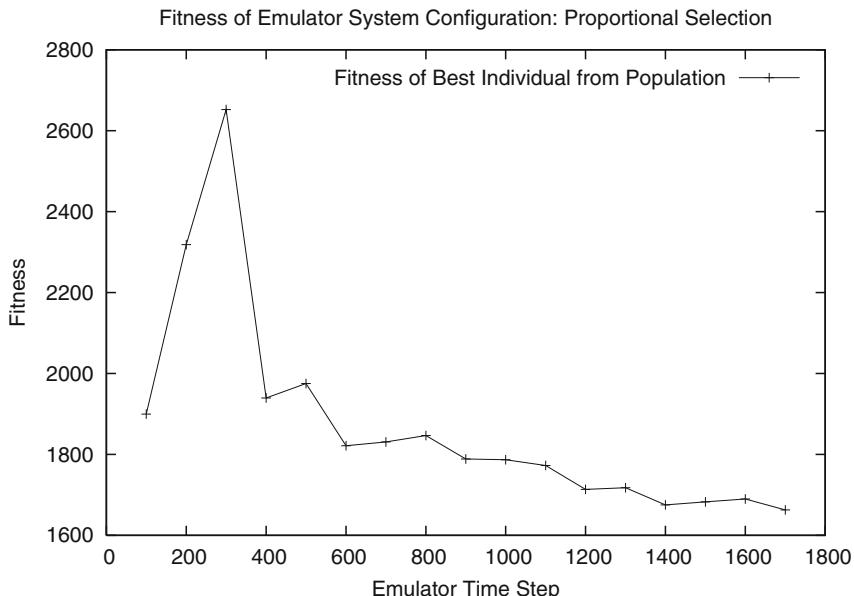


Fig. 2.18 Fitness of emulator system configuration: proportional selection – during a single replication of the SAMS parallel application, the current emulator system configuration is set by the current most fit individual in the

GAC's population. With proportional selection, an initial worsening of fitness is typically followed by a slow improvement. Every 100 time steps, the emulator's system configuration is updated with the most fit individual.

ry analysis, which experiment with an ensemble of plausible models, have been developed to deal with these problems.

The SAMS approach involves a hybrid exploration strategy to study an ensemble of plausible models. When parameterized to account for input uncertainty in controllable and uncontrollable factors, SAMS is able to dynamically update a system emulator resulting in improved performance. This benefit is realized with the help of a GA that evolves potential system configurations over the lifetime of the system emulator and that can be used to update the emulator. These updates consist of adaptive strategies that are passed on to the agents operating within the emulator. This initial study of SAMS as a simulation methodology has shown encouraging results, but has also left many problems unsolved. In particular, further experimentation with larger numbers of model replications is required. Also, experimentation with additional GA designs can be performed to understand the features that make an evolutionary algorithm suitable for SAMS. The understanding gained could provide further improvements to the methodology in terms of speed and robustness. Other significant problems that remain are the incorporation of appropriate multiresolution modeling, input analysis for estimating uncontrollable factor distributions, and handling of structural uncertainty. Given these challenges, Symbiotic Adaptive Multisimulation appears to be a rich opportunity for further study.

References

- Bankes, S. (1993) Exploratory modeling for policy analysis. *Operations Research*, **43** (1), 435–449.
- Bratton, D. and Kennedy, J. (2007) *Defining a standard for particle swarm optimization*. Proceedings of the 2007 IEEE Swarm Intelligence Symposium, IEEE, Honolulu, HI, pp. 120–127.
- Braude, E.J. (2004) *Software Design: From Programming to Architecture*, John Wiley & Sons, New York.
- Collier, T.C. and Taylor, C. (2004) Self-organization in sensor networks. *Journal of Parallel and Distributed Computing*, **64** (7), 866–873.
- Crowther, B. (2004) Flocking of autonomous unmanned air vehicles. *Aeronautical Journal*, **107** (1068), 111–124.
- Davis, P.K. and Bigelow, J.H. (2000) *Exploratory analysis enabled by multiresolution, multiperspective modeling*. Proceedings of the 2000 Winter Simulation Conference, pp. 127–134.
- Davis, P.K. and Bigelow, J.H. (1988) The role of uncertainty in assessing the nato-pact central region balance. *RAND N-2839. The RAND Corporation*, Santa Monica, CA.
- DeJong, K.A. (2006) *Evolutionary Computation: A Unified Approach*, MIT Press, Cambridge, MA.
- Dreو, J., Petrowski, A., Siarry, P. and Taillard, E. (2006) *Metaheuristics for Hard Optimization: Methods and Case Studies*, Springer-Verlag, Berlin, Germany.
- Endsley, M.R. (1995) Measures of situation awareness in dynamic systems. *Journal for Human Factors*, **37** (1), 65–84.
- Fujimoto, R.M., Lunceford, D., Page, E. and Uhrmacher, A.M. (2002) Grand challenges for modeling and simulation: Dagstuhl report. Fujimoto, R.M., Lunceford, D., Page, E. and Uhrmacher, A.M. (2002) Grand challenges for modeling and simulation: Dagstuhl report.
- Goldberg, D.E. (1989) *Genetic algorithms in search, optimization, and machine learning*, Addison-Wesley, Reading, MA.
- Kennedy, J. and Eberhart, R.C. (2001) *Swarm Intelligence*, Morgan Kaufmann Publishers, San Francisco, CA.
- Lua, C.A., Altenburg, K. and Nygard, K.E. (2003) *Synchronized multi-point attack by autonomous reactive vehicles with simple local communication*. Proceedings of the 2003 IEEE Swarm Intelligence Symposium. SIS 03, pp. 95–102. IEEE, Indianapolis, IN.
- Namatame, A. and Sasaki, T. (1998) Self-organization of complex adaptive systems as a society of rational agents. *Artificial Life and Robotics*, **2** (4), 189–195.
- Price, I.C. (2006) *Evolving self-organized behavior for homogeneous and heterogeneous uav or ucav swarms*. Master's thesis, Air Force Institute of Technology, Wright-Patterson Air Force Base, OH.
- Reynolds, C. (1999) *Steering behaviors for autonomous characters*, <http://www.red3d.com/cwr/papers/1999/gdc99steer.html> (accessed 31 October 2007).
- Wooldridge, M. (2002) *Introduction to Multiagent Systems*, Wiley.
- Yilmaz, L. (2004) *Dynamic model updating in simulation with multimodels: A taxonomy and a generic agent-based architecture*. Proceedings of SCSC 2004 – Summer Computer Simulation Conference, pp. 3–8.
- Yilmaz, L. (2007) Toward next generation simulation-based computational tools for conflict and peace studies. *Social Science Computer Review*, **25** (1), 48–60.
- Yilmaz, L., Lim, A., Bowen, S. and Ören, T. (2007) *Requirements and design principles for multisimulation with multiresolution, multistage models*. Proceedings of the 2007 ACM/IEEE Winter Simulation Conference, pp. 823–832.

Part Two Agents and Modeling and Simulation

3

Agents: Agenthood, Agent Architectures, and Agent Taxonomies

Andreas Tolk and Adelinde M. Uhrmacher

3.1

Introduction

The agent metaphor is based on developments in diverse computer science areas such as distributed systems, software engineering, and artificial intelligence. It has been strongly influenced by the research results of other disciplines as well, in particular sociology, biology, systems and decision science, and many others. Those diverse areas of research are reflected in multiple facets that characterize agents.

This shared parenthood also explains the similarity with certain approaches – as agents cannot deny their roots – and consequently the need for distinction. For example, agents have been characterized as reflective and concurrent objects (Gasser and Briot, 1992) and as embedded artificial intelligence (Russell and Norvig, 1995), migrating agents have been described as mobile code plus state plus autonomy (Kotz and Gray, 1999), agents should represent human behavior as placeholders in applications (Silverman, 2001), and so on. Thus, the properties of being “autonomous” and “flexible” – often referred to as “adaptive” – are perhaps the most prominent characteristics of agents (Wooldridge and Jennings, 1995) and are also considered to hold the most promise for future development (Jennings et al., 1998).

In this chapter, we will look at some typical agent properties and describe some methods and architectures that have proven helpful in bringing about these desirable agent characteristics. In addition, we will explore the relation to simulation and systems engineering and, finally, discuss some possibilities for structuring the realm of agents, starting with taxonomies. We will also develop a three-dimensional space, taking into account characteristics of agents, their environments, and the community of agents. This will help locate agents and reflects the many facets of agents and agent systems. As such, this chapter uses examples of historical and state-of-the-art contributions to highlight concepts of interest in the scope of this book.

3.2

Agenthood

Several papers have dealt with the question of what makes an agent and agent, from the social sciences (Epstein and Axtell, 1996) to the many implementation-specific viewpoints given in various journal contributions and proceedings. We will focus on those aspects that support the systems engineer who has to decide whether the application of agent-based technologies appears promising for his or her project. Therefore, we will give an overview of popular definitions, highlighting the aspects of particular interest to systems engineering.

3.2.1

Defining Agents

The question “Is it an agent, or just a program?” was posed by Franklin and Graesser (Franklin and Graesser, 1996) and already reveals some of the problems in defining an agent. The paper belongs to a series of efforts aimed at defining agents during the 1990s. At that time, there was a lot of interest in such definitions, as agents had just been introduced as a new paradigm. Over the years, this interest in finding a formal consensus decreased and emphasis was placed on application domains of the agent paradigm. Fortunately, as other scientific areas show as well, a commonly agreed-upon definition is not a prerequisite for the success of a concept in practice.

To answer the above question, Franklin and Graesser discuss several definitions of prominent agent researchers.

The notion of agents suggested by Russell and Norvig (Russell and Norvig, 1995), that “an agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through effectors,” applies to all programs whose outputs are based on inputs.

Other definitions take the type of environment into account, that is, whether it is dynamic and complex, and also assume some autonomy and goal-directedness of the agent, for example, “autonomous agents are computational systems that inhabit some complex dynamic environment, sense and act autonomously in this environment, and by doing so realize a set of goals or tasks for which they are designed.” (Maes, 1995). Nothing is said about how this autonomy or goal-directedness will be achieved by a concrete agent.

In contrast, (Hayes-Roth, 1995) emphasizes the traditional artificial intelligence view on agents by assuming that all agents use explicitly reasoning mechanisms: “Intelligent agents continuously perform three functions: perception of dynamic conditions in the environment; action to affect conditions in the environment; and reasoning to interpret perceptions, solve problems, draw inferences, and determine actions.”.

Other researchers follow the argumentation line of Brooks, who provocatively raised the question of whether the design of intelligently behaving systems requires representation or reasoning (Brooks, 1991a,b). Brooks emphasizes the reactivity

of agents as one of the intrinsic features of agents that has to be combined with means for deliberation (Stone and Veloso, 2000).

Wooldridge and Jennings (Wooldridge and Jennings, 1995) define an agent basically as a hardware- or software-based computer system that has the following properties:

- *Autonomy*: agents operate without the direct intervention of humans or others and have some kind of control over their actions and internal state;
- *Social ability*: agents interact with other agents (and possibly humans) via some kind of agent-communication language;
- *Reactivity*: agents perceive their environment (which may be the physical world, a user via a graphical user interface, a collection of other agents, the Internet, or perhaps all of these combined) and respond in a timely fashion to changes that occur in it;
- *Proactiveness*: agents do not simply act in response to their environment; they are able to exhibit goal-directed behavior by taking the initiative.

Again, no assumption is made as to how to achieve certain characteristics. However, central problems of agents are taken into account, that is, how to mediate between reactivity and proactiveness and how to interact with other agents and, potentially, humans. The environment is only implicitly considered: in order to react to something, the agent must perceive it. This deficiency has been addressed in later publications: “an agent is a computer system, situated in some environment, that is capable of flexible autonomous action in order to meet its design objectives....There are thus three key concepts in our definition: situatedness, autonomy, and flexibility” (Jennings et al., 1998).

In the context of this paper, to highlight the aspects of particular interest to systems engineering, we propose the following working definition:

- The agent is situated, it *perceives* its environment, and it *acts* in its environment. The environment includes typically other agents, other partly dynamic objects and passive ones that are, for example, subject to manipulation by the agent. *Communication* with other agents is of particular interest in systems comprising multiple agents, as agents can collaborate and compete for tasks. This latter characteristic has also been referred to as *social ability*.
- The agent should be *autonomous*, in the sense that it can operate without the direct intervention of humans or others, and autonomy requires control over its own state and behavior.
- To be *flexible* for an agent means to mediate between reactive behavior, being able to react to changes in its environment, and deliberativeness to pursue its goals. A suitable mediation is one of the critical aspects for an agent to achieve its tasks in a dynamic environment. An agent can act upon its knowledge, its rules, beliefs, operators, goals, and experiences, and so on and *adapt* to new constraints and requirements – or even new environments – as required. For example, new situations might ask for new goals, and new experiences might lead to new behavior rules. Also, being *mobile* adds to the flexibility of an agent.

3.2.2

Situated Environment and Agent Society

Typically, we are not only faced with one agent, but with multiple agents in their environment. In what follows we will understand agent systems as systems that are comprised of a set of objects. We distinguish between active and passive objects. In general, passive objects do not change their attributes driven by behavior of their own; they are often exclusively subject to manipulation. In contrast, active objects have a behavior of their own; agents are a subset of these active objects. In multi-agent systems, generally each agent has incomplete information or capabilities for solving the problem at hand, data and control are decentralized, and computation happens asynchronously.

In Figure 3.1 an agent is situated in its environment, it perceives its environment (including other agents), maps the perception to an internal representation, based on its knowledge and goals, and it communicates and acts with other agents. Its environment contains other objects, passive ones like obstacles, but also active ones, like the ball, which has a dynamic of its own (once kicked) and which has to be taken into account. An agent's perception of the environment can be incomplete (it does not know all of what is true, for example, the agent does not perceive the triangle and the second block) and uncertain (nor is all true what it believes to be true). Agents are bounded with respect to their knowledge and, due to the dynamic environment, to the time they have to come up with a decision and to act.

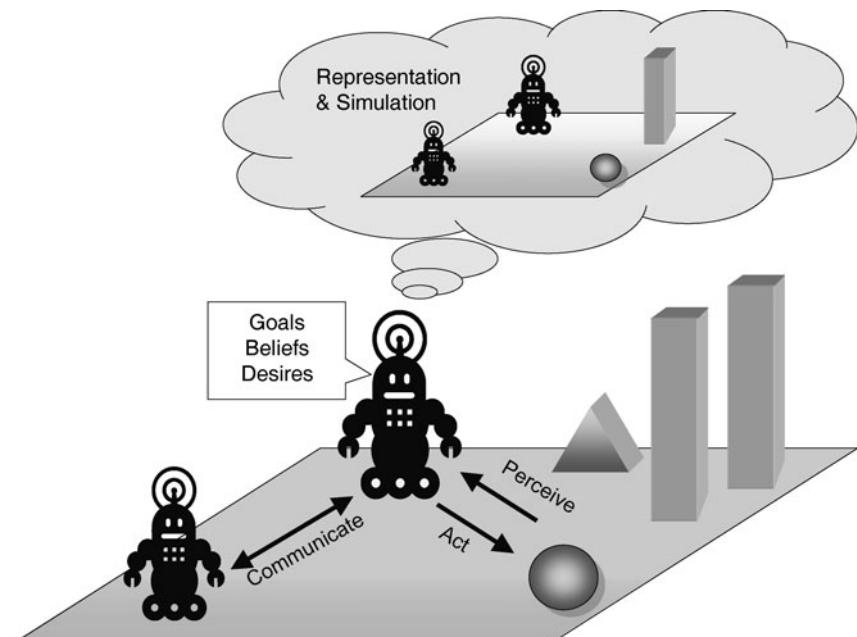


Fig. 3.1 Agents in their situated environment.

In this regard, they are not unlike humans: in making decisions, humans operate within a complex and often changing environment with limited cognitive capabilities, time, and other resources. Hence, decision making is only rational within the bounds imposed on decision makers.

Tversky and Kahneman (Tversky and Kahneman, 1974) identified a number of heuristics and biases that humans use to make decisions. These studies aim to bring classical and analytic decision theorists into conformity with findings in cognitive psychology. Thus, heuristics often yield cost-effective solutions compared to classical methods in terms of time and mental effort, a fact that is also exploited by traditional methods in artificial intelligence.

3.3

Agent Architectures

Usually, agents are defined as entities with a clear boundary to their environment. Agents interact with other agents and their environment. Agents act by communicating with other agents, including human beings, and by interacting via sensors and effectors with the nonagent environment.

This skeletal definition complicates the distinction between agents and objects. Accordingly, the close relationship between agent- and object-oriented approaches has been subject to a multitude of discussions and publications, for example, (Franklin and Graesser, 1996; Gasser and Briot, 1992). Objects are typically perceived as a good starting point for designing agents. However, obviously agents are more than objects. The question is how are anthropomorphically colored properties, such as being intelligent, autonomous, social, mobile, cooperative, and rational – or, in our case, being situated, autonomous, and flexible – realized.

To realize these characteristics, several agent architectures are available. As in the first section, we will not exclude promising approaches by drawing the lines too narrow, but we will highlight examples of architectures that expose aspects of interest to systems engineering.

3.3.1

Realizing Situatedness

Agents exist in an environment, with which they interact:

- *Perceiving* refers to the observable properties of the objects and associations of the environment the agent is associated in. This is supported by *sensors*. Different sensors can support the observation of different properties, and reasoning capabilities can reveal even hidden properties and objects in the environment.
- *Acting* results typically in changed properties of the objects and associations of the environment in which the agent is situated. Acting might also take the form of communicating with other agents. However, as this requires quite different methods, we will discuss this separately.

A closer look at sensor types is particularly important for the sensomotoric variant of agents, such as robots. Infrared sensors provide other data about the environment that bumpers or vision tools do not. In addition, translating data into knowledge – such as “there is a ball in front and a player to the left” – requires different efforts, which under real-time constraints limits the value of the information obtained. In these scenarios, different sensors are typically combined. RoboCup (Marques and Lima, 2001) can serve as an example.

The background knowledge the agent has about its environment is crucial. In order to know about a class of objects or relations with objects the agent must be able to use an internal representation of the class or relations in at least the following ways: receive information about the class or relations, generate elements, recognize members and discriminate them from other class members, answer questions about the class, and take into account information about changes in the class members. This includes the ability to conduct (mental) simulations using the recognized class characteristics to create an anticipation for the agent, so that he can be proactive.

The task of structuring the knowledge base such that it is sufficiently flexible to enable sensemaking about all observable objects without having to assume that all classes and relations are known *a priori* is not trivial. Among others, Sowa introduced several ideas that are applicable in this context to describe physical and abstract entities for knowledge representation with the objective of making the knowledge applicable for computational support (Sowa, 2000).

As the sensors, together with the background knowledge the agent has about its environment, constrain the agent’s understanding of the environment in which it is situated, so do the effectors (again together with the agent’s knowledge that it can do these things) limit its ability to manipulate its environment. We find the most diversity of actuators in agents that are working in physical environments, for example, wheels, legs, and grippers. These sensors and effectors are designed to interacting with the “nonagent” environment; however, as most often a world is not inhabited by one agent but by several agents, and tasks cannot be achieved by one agent alone, communication and cooperation between agents plays a central role.

In general, agents are designed independently of each other. This can make communication and cooperation difficult. In traditional systems engineering, unambiguous information exchange can be realized by using application-specific syntax and semantics that are shared between all system components (or agents) (Tolk et al., 2007). In addition, all system components can be assumed to be benevolent, as they are developed by one engineering team. Between independently developed agents, such a common language, which specifies *how they talk* (syntax) and *what they talk about* (semantics) does not necessarily exist, and neither does *an a priori benevolence* exist.

Therefore, a metalanguage has been proposed to describe the structure of knowledge as well as content. Since 1996, the Foundation for Intelligent Physical Agents (FIPA) has played a crucial role in the development of agent standards, including standards for agent communication. Those comprise standards for interaction

protocols (guaranteeing some autonomy; they will be discussed in the section on autonomy), for communicative acts, and for content language (F.I.P.A.).

The *Knowledge Query and Manipulation Language* (KQML) is complemented by the *Knowledge Interchange Format* (KIF). KQML (Group) is the “outer language,” which defines various communicative verbs in the form of performatives, for example, request, inform, or inquire, thereby adopting concepts of the speech act theory (Searle, 1969). KIF is the “content language” describing knowledge in the form of logical expressions (Genesereth and Fikes, 1992). Both KQML and KIF are part of the FIPA–ACL (Agent Communication Language) (F.I.P.A.). Current efforts focus on developing suitable approaches for the semantic web, for example, DAML+OIL [Defense Advanced Research Project Agency (DARPA) Agent Markup Language plus Ontology Integration Language] has been developed to support the identification, matching, and composition of services. See (Botelho et al., 2002) for a comparison of DAML+OIL, Electronic Business using eXtensible Markup Language (ebXML), FIPA Semantic Language (FIPA-SL), and KIF.

3.3.2

Realizing Autonomy

Agents and objects can be distinguished depending on their degree of autonomy. Agents have their own thread of control, so multiagent systems are typically multithreaded, while in a standard object model normally only one single thread of control exists (Jennings et al., 1998). In addition, agents do not invoke methods upon one another, but rather they request actions to be performed. In objects, this decision lies with the object that invokes the method, whereas in agents the decision lies with the agent that receives the request.

How is this realized? Typically, agents use some form of reflection to achieve this. They are able to reason about their own state, the incoming method invocation, and the agent requesting the service from them. Depending on this information, they are able to grant or not grant the request. Most mobile agent platforms are implemented in Java and make use of the Java reflection mechanism, which is used for migration and for invoking methods (Pham and Karmouch, 1998). They also associate their own thread of control with each agent. In Mole, for example, each method invocation leads to the generation of new threads (Baumann et al., 1997).

The phrase “objects do it for free, agents for money” (Jennings et al., 1998), however, implies, in addition to these more technical concerns, something else: it is possible to negotiate with agents. This motivated the development of negotiation protocols (Jennings et al., 2001), such as the contract net 20 years ago (Smith, 1988), and more recently the development of trading strategies to support e-commerce, addressing bidding problems (coming up with good heuristics in the face of uncertain prices) and interactions in different market types, for example, simultaneous ascending auctions and continuous double auctions (Wellman et al., 2007).

As agents are autonomous, this might also imply that they are pursuing their own interests, possibly at the expense of the community. Thus, a traditional socio-logical question reappears in the realm of agents: how do I achieve suitable results in a community of selfish agents? Not surprisingly, in this context game-theoretic approaches are exploited as they provide a mathematical framework to analyze and to construct rational behavior. *Rational choice models* (RCMs) are used in such diverse fields as economics, sociology, political science, management science, and operation research. Von Neumann and Morgenstern introduced the idea that rational choice should maximize expected subjective utility (von Neumann and Morgenstern, 1953). For two-agent zero-sum games, optimal strategies are suggested by the Nash equilibrium. However, for non-zero-sum games equilibria do not necessarily suggest desirable behavior strategies, at least not from the point of view of the community, and thus repetitive games are needed to let cooperation evolve (Axelrod, 1997). It is even worse in many-person competitive situations in which players form coalitions. Although for many competitive situations game theory will not offer the solution, it helps to illuminate the problem and to interpret the competitive interactions and possible results in a different manner (Parsons et al., 2002).

3.3.3

Realizing Flexibility

To achieve the desired flexibility, a combination of reactivity and deliberation is needed. An unknown and dynamic environment requires agents that are able to adapt their behavior easily to environmental changes. Based on hard-wired behavior patterns, reactive agents have demonstrated their skills in playing Pengo and discussed critically the need for plans (Agre and Chapman, 1990) and avoiding obstacles while exploring their environment (Brooks, 1986).

In contrast, artificial intelligence methods have long taken closed and static worlds for granted. Planning is one field where the influence of the agent metaphor becomes particularly vivid. Traditionally, planning systems generate plans based on their current internal knowledge, which comprises basic knowledge and environmental observations up to that time. The generated plan is executed step by step – in the hope the internal model the plan is based upon and the underlying goals are still valid – which might not be true in an open dynamic environment.

The *Belief–Desire–Intention* (BDI) architecture (Bratman et al., 1988) is perhaps the most prominent agent architecture. In this architecture, the agent's memory is structured into four sections: the data input from the perception is mapped to a set of *beliefs* about the environment. The goals to be achieved are stored in the *intentions* stack. Plans that can be executed are stored in the *plan* stack. Long-term goals are defined in the *desire* section. BDI was successfully implemented and applied to real-world problems, for example, for air-traffic management applications (Rao and Georgeff, 1991, 1995), for fault diagnosis of the Space Shuttle, and for business process control (d'Inverno et al., 1997). The algorithm shows an adaptation of the original BDI architecture (Wooldridge, 2002).

In contrast to the original algorithm, the enhanced version checks at each step whether the goal has already been fulfilled or whether the goal has in the meantime become impossible, both of which might be the case due to the dynamics of the environment. In addition, it checks regularly the environment to reconsider its options. The procedure “reconsider” achieves a balance between agents that are overly committed to their goal and those that are too opportunistic to reach any goals. Furthermore, actions might have gone wrong or something else might have happened that no longer allows agents to execute the next plan steps. In all these cases, a new plan is needed.

Mobility offers another dimension of flexibility. Platforms for mobile agent systems aim at supporting the design of autonomous programs that are capable of physically moving themselves through communication networks in order to perform specific service tasks. The mobility of agents is motivated by the desire to utilize efficiently geographically distributed services. Agents are dispatched to the remote server site, where they perform the necessary interactions locally. After completing the task at one site they return with the results or continue their journey to visit other sites. Thus, instead of an iterated back and forth between client and server, one round trip of an agent does the job. Compared with message passing and remote procedure calls, mobile code allows one to conserve bandwidth and reduce latency, particularly if an information resource provides only low-level operations. The processing of high-level tasks by mobile code renders transferring intermediate data across the network superfluous. If the connection between client and server breaks down during performance of the operation, it does not affect that operation.

The idea of a self-controlled program execution near the data source has been proposed as a better, more efficient, and more flexible mode of communication with two goals: reduction of network traffic and asynchronous interaction. It appears similar to process migration concepts. However, mobile agents also exhibit some of the other characteristics of agents, such as being autonomous, which makes them not well suited to deal with load balancing in distributed simulation (although approaches do exist (Fukuda et al., 2001)). Agents appear to be more suitable for open environments. However, here other problems are lurking. Mobile agents treat the network as an “agent-friendly” environment, which is not necessarily true. Particularly security issues – like how to prevent a malicious host from taking advantage of mobile agents exposing their properties – are among the central challenges mobile agent system design faces (Gray et al., 1998).

Mole represents a Java-based mobile agent system (Baumann et al., 1997). Mole agents reside in locations that are embedded in the Mole runtime system, the engine. The engine transforms and forwards messages between the locations and the network. Each engine might comprise a set of locations (Figure 3.2). They offer certain services to the agent and represent the source and destination of moving agents. Mole agents are equipped with a set of methods, for example, for migrating, remote procedure calls (RPC), sending and receiving messages, and for handling the individual life cycle. In addition, Mole agents can use the entire functionality of Java, only constrained by the security model employed. Agents can comprise a dy-

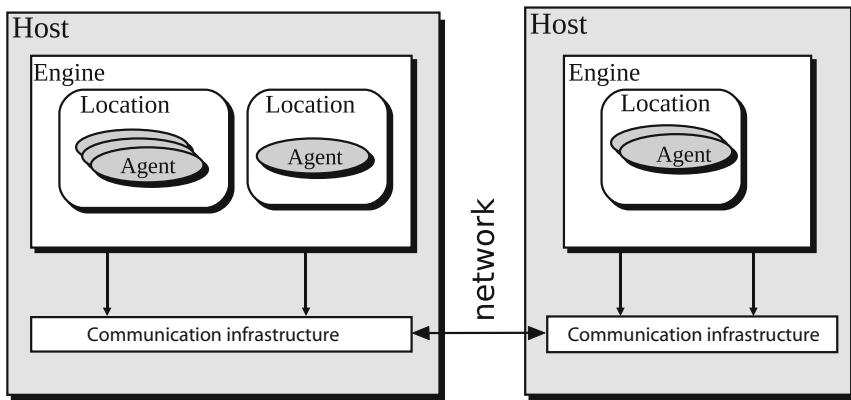


Fig. 3.2 Mobile agents in Mole.

namic set of concurrent running or waiting threads and are not restricted to one line of activity.

3.3.4

Architectures and Characteristics

The architectural frame shown in Figure 3.3 was proposed in (Moya and Tolk, 2007) in support of discussing how the agent characteristics discussed in our session on agenthood can be realized. It is kept simple on purpose, as we do not want to prescribe a solution, but just make developers aware of domains that need to be taken into account. If a domain is not covered in a solution, it should not be covered by purposeful design. As such, Figure 3.3 is a guideline for system engineers and simulation and agent developers. It is neither complete nor exclusive.

There are three external and four internal architectural domains identified in (Moya and Tolk, 2007). The external domains comprise those functions needed within an agent to interact with its environment:

- The perception domain observes the environment. Using its sensors, the agent receives signals from its environment and sends this information to the internal sensemaking domain.
- The action domain comprises the effectors. If the agent acts in its environment, the necessary functions are placed here. Its task is to perform tasks from the internal decision making domain.
- The communication domain exchanges information with other agents or humans. If it receives information, it is sent to the internal sensemaking domain. Its task is to send information from the internal decision making domain.

The internal domains categorize the functions needed for the agent to act and adapt as an autonomous object. The four domains identified here are as follows:

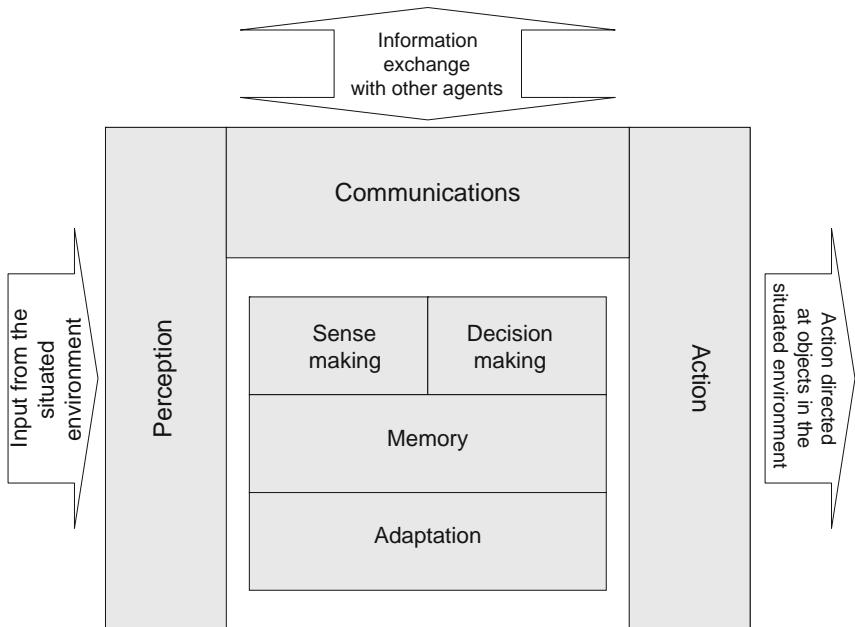


Fig. 3.3 Architectural frame addressing main agent characteristics.

- The sensemaking domain receives input (sensors and communication) and maps this information to the internal representation, that is, part of the memory domain. This domain potentially comprises data correlation and data fusion methods, data mediation capabilities, methods to cope with uncertain, incomplete, and contradictory data, and so on.
- The decision making domain supports reactive as well as deliberative methods, as discussed in this chapter. It uses the information stored in the memory domain and triggers communications and actions.
- The adaptation domain may be connected with perception and action as well, but that is not a necessary requirement. The function group updates the information in the memory domain to reflect current goals, tasks, and desires.
- The memory domain stores all information needed for the agent to perform its tasks. It is possible to distinguish between long-term and short-term memory, different methods to represent knowledge can be used alternatively or in hybrid modes, and so on.

This architectural framework is not intended to replace more concrete architectures used as examples in this chapter. It is a blueprint that needs to be adapted and made concrete for applications by extending and enhancing this hub based on the requirements for the project. For example, the critical components of planning, understanding, and anticipation enabling the agent to reason need to be

detailed in the decision making component. Similarly, how the learning feature is realized needs to be captured when implementing a solution based on this blueprint.

3.4

Agenthood Implications for Practical Applications

In this section, we will evaluate the implications of agent characteristics particularly for systems engineering applications. The focus lies on contributions of agents to such applications. Other chapters will deal with related topics in more detail. However, in order to understand the characteristics better from a practical viewpoint, some overview is useful in the context of agents and agenthood as defined here.

Systems engineering is increasingly concerned with designing “systems of systems” (Keating et al., 2003). Traditional systems engineering defines a system within specified system borders as a well-specified set of functionality that is offered to the user via equally well-defined access points. Information is exchanged with external systems via well-engineered interfaces across the system borders. In particular, in netcentric environments – such as in electronic business environments (Davis and Ritsko, 2003) or the Global Information Grid envisioned by the US Armed Forces (of Defense, Sep. 19, 2002) – these borders between systems become fluid. Instead of functions, services are provided that can be composed to deliver the required functionality “on the fly”. As the developers of such services need to communicate their assumptions and constraints to other service users – as this cannot be done by an engineering team as is the case in traditional systems engineering – communication, matchmaking, and ontologies become essential if we are interested in compositions that deliver meaningful results on the conceptual level. This is true for the composition of Web services in general (Alesso and Smith, 2005; Argawal et al., 2005) and in the composition of models in particular (Tolk et al., 2007).

Systems of systems can be designed with a set of characteristics that are also associated with agents, such as working independently, functioning in a distributed and adaptive manner, showing emergent behavior and evolutionary development, and so on. However, systems of systems do not necessarily need to work autonomously. They do not all require their own thread of control, nor do they need the ability to reason about their own capabilities and their environment. Some agent characteristics can be directly applied, while others are of less importance. Thus, systems of systems do not translate one to one to the agent metaphor.

In this section, we put the emphasis on “adapting” to changed requirements and environments rather than on “interacting” with a dynamic environment. We will not address the issue of whether an object-, agent-, component-, or service-oriented approach – or a combination thereof – appears to be the most promising for engineering such a system. This needs to be decided individually by the engineer. Sometimes the differences between these concepts seem to blur. We want to point

out that no best practices exist so far in this domain, but individual evaluation is necessary.

- Agents are not (just) services (Dickinson and Wooldridge, 2005), although agents can easily be used for realizing Web services.
- Compared to objects, agents embody a stronger notion of autonomy (Odell, 2002). Agents can be interpreted as reflective concurrent objects and a community of agents as a special type of concurrent, distributed system. They also have the introspective and intercessory capabilities to update their behavior.
- Components can be used to implement agents (Melo et al., 2004). However, neither an individualized thread of control nor functioning in a dynamic environment is associated with components (Casagni and Lyell, 2003).

3.4.1

Systems Engineering, Simulation, and Agents

Due to their shared characteristics, agents have started to play a central role in engineering systems, as does simulation. Agents, in particular agent architectures, help in designing communication and coordination protocols in a system. Simulation helps to answer questions about the achieved behavior, performance, and robustness. Thus, they give the first feedback regarding the quality of the design. In addition, simulation can be used for decision support by providing “what if” scenarios as well as for training and education purposes. Although the intentions of the simulation varies – and with this the type of modeling and simulation used – agents are playing an increasingly central role in all these areas. Thus, agents are likely to replace to a certain degree objects that have traditionally been used in systems engineering (Oliver et al., 1997) and modeling as well as simulation (Uhrmacher, 1997). The impact that agents have on modeling and simulation is manifold and has been subject to a series of explorations, for example, (Uhrmacher et al., 2001; Yilmaz and Tolk, 2006; Uhrmacher and Röhl, 2007; Uhrmacher and Weyns, to appear 2008). Agents are used as a metaphor for modeling, as a programming metaphor to design distributed simulation systems, and simulation is used for a better understanding of multiagent system behavior and for designing and evaluating software agents. Figure 3.4 shows the high-level relations between these domains.

The entire process of systems engineering might be the subject of a simulation study. In this case, the general work flow is typically of interest. Simulation supports optimizing workflow processes, and animation enlightens the interplay between the different processes. In particular, if workflow processes are adaptive, then workflows can be realized as communities of agents (Singh and Huhns, 1999), and consequently their simulation and animation require an agent-based approach.

If less abstract processes are of interest, then simulation allows a detailed inspection of specific aspects of engineering systems. Thus, simulation has traditionally focused on technical processes. However, the importance of considering decision processes and human behavior is meanwhile widely acknowledged, for example,

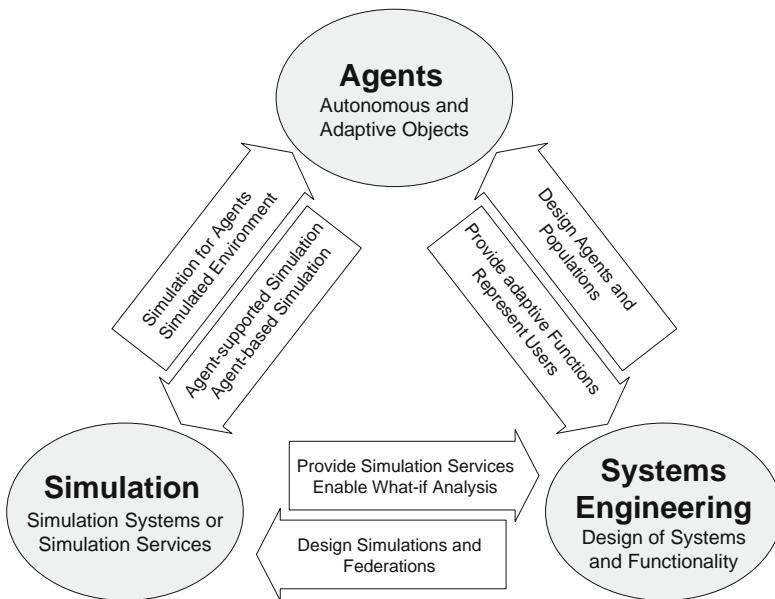


Fig. 3.4 Relating systems engineering, simulation, and agents.

in supply chain planning and management (Eulalia et al., 2007) or traffic scenarios (Balmer et al., 2004). Generally, because they take anthropomorphic aspects into account, agent-based approaches are particularly promising and are consequently exploited in their full range.

Whereas in the above-described scenarios the decision processes are part of the simulation, simulation is also used for decision support. This online simulation typically uses data from external sources. If a simulation system interacts directly with the dynamic environment, then this type of simulation system can benefit from being designed as software agents, as shown in (Low et al., 2005). Clearly, the constraints of this type of simulation are quite different from “offline” simulation scenarios and are quite differently affected by the agents’ approach as they adopt methods developed for software agents’ design to achieve the desired properties of the simulation system rather than exploit techniques to take human behavior into account.

3.4.2

Modeling and Simulating Human Behavior for Systems Engineering

Sociology, ecology, and civil engineering belong to the traditional application areas of the agent metaphor. In all these areas, the description of human actors have traditionally played a central role. The agent metaphor could build on well-established approaches like microsimulation in sociology (Halpin, 1999), individual-based modeling in ecology (Grimm and Railsback, 2005), or micro models of traffic systems (Pursula, 1999).

Particularly in the social realm – where phenomena such as cooperation, coordination, group and coalition formation, and the evolution of conventions and norms are of interest – multiagent-based approaches have flourished in recent decades (Axtell, 2000; Conte et al., 1998; Gilbert and Troitzsch, 1999). Often agents exploit only local rules to generate the pattern of interest at the macro level (Epstein, 1999). If large numbers of agents are simulated, then deliberative agents require significant computational effort, which most agent-based simulation systems are not well prepared to face (Theodoropoulos et al., 2008). Another reason for the hesitance to adopt deliberative agents is the problem of defining and validating this type of model. With the heterogeneity of interaction patterns, abstraction, and time scales, the complexity of the model increases and the overall interpretation becomes difficult. In addition, empirical studies to validate these models, including the detailed decision processes, are rare (Doran, 1997). Thus, collecting the information and data required for this type of detailed human behavior model requires considerable effort (Silverman et al., 2008).

Sometimes, however, lack of data is the very reason to delve into speculation based on agent-based modeling and simulation. The following case study gives an example of the successful application of agents to represent human behavior for systems evaluation.

Food shortages threatened the standard of living in urban communities in the Middle Ages and in early modern times. Local governments tried to control market dynamics, but it is unclear whether the typical market reaction of rising prices of foodstuffs and wages could be moderated in the long run by an intervention in markets. To answer this question, econometric analysis of historical time series data is a viable approach if data are available. However, precise data on many economic variables, for example, on population figures and on environmental factors, is mostly missing. A multiagent-based model of the premodern urban economy is used to explore the effects of different crisis management strategies on the economic and demographic developments in an urban community experiencing a food shortage. Merchants, craftsmen, and laborers are represented as utility-based and reactive agents assuming rational choices of agents and expecting them to maximize their utility and their profits. This behavior of actors is coordinated by the markets where agents sell and buy goods. The supply and demand of grain, commodities, services, and labor is assumed to follow basic principles derived from neoclassical economic theory. In addition to these numerous reactive agents, a deliberative single agent is added to the model and represents a group of decision makers within the urban society and can be described as the town's local authorities. An overview of the resulting structures is shown in Figure 3.5. The conducted simulation runs showed that interventions in markets turn out to very likely reduce the wealth-destroying consequences of food shortages or even famines (Ewert et al., 2007) and, thus, support establishing certain instruments in dealing with the aftermath of disasters.

When it comes to traffic simulation, microscopic models have a long tradition (Pursula, 1999). However, the microscopic view refers often to the spatial segmentation of traffic areas for the realization of which, for example, cellular

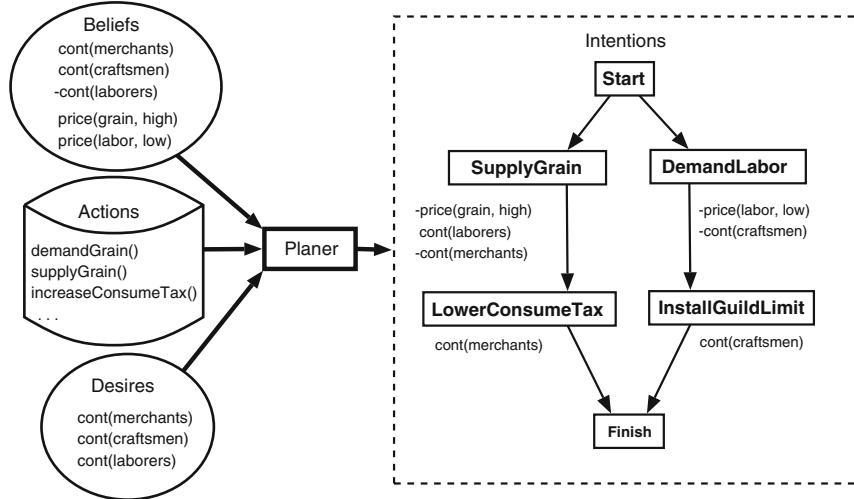


Fig. 3.5 Deliberate agents to explore the effect of different intervention strategies.

automata are exploited (Balmer et al., 2004; Esser and Schreckenberg, 1997). Agent-oriented approaches allow one to focus on the individuals and their decision processes who are involved in the traffic scenario. Thus, travelers' behavior and their interaction with intelligent, state-of-the-art transportation technologies are taken into account addressing the salient complexity of urban traffic scenarios in a more adequate manner. However, the rich cognitive models of drivers' behavior imply more effort in simulating models and defining and validating them. Just as in sociology, relating microscopic and macroscopic behavior is a central challenge (Sahraoui and Jayakrishnan, 2005) that, when it comes to multiagent systems, is reflected in global, central control on the one hand and decentralized, evolving strategies on the other (Bazzan et al., 2005).

Benefits for the diverse application areas in systems engineering can be seen in the natural metaphor to capture human behaviors and decision processes. The different agent architectures, and protocols for communication and coordination, provide the means to model and simulate the behavior and decision processes of individuals, small groups, or even entire populations, and their communication and cooperation patterns. As the integration of humans and technical aspects in systems engineering takes hold, for example, (Booher, 2003), and given the traditional central role of modeling and simulation in systems engineering, for example, (Smith, 1962), agent-based modeling and simulation constitutes a major asset for systems engineering. However, these benefits do not come for free; typically agent-based models imply more effort in all stages of the modeling and simulation process, during the definition of models, setting up the experiments, executing the models, interpreting the results, and validating the models. Thus, agent-based models should be used when no simpler alternative exists. To avoid the pitfalls of agent-based design (Wooldridge and Jennings, 1998), a modeler should know

when to constrain the internal richness of and the interaction patterns between model components for reasons of simplicity, easier validation, and interpretation of the overall model. As with the use of modeling approaches, the use of agents should be accompanied by a thorough analysis of why this approach is more adequate than others given the system under study and the questions to be answered.

3.4.3

Simulation-Based Testing in Systems Engineering

The rise of the agent metaphor has undoubtedly helped in establishing modeling and simulation in recent decades in the above areas as it has facilitated enriching rather technical models with anthropomorphic aspects. In contrast, other application fields for agent-based modeling and simulation in systems engineering emerged only recently. As already mentioned, systems engineering is becoming increasingly concerned with highly concurrent, distributed, and adaptive systems embracing human, organizational, and technical details. The design of these systems provides new challenges, and adopting the agent metaphor might answer some of them.

Software agents are supposed to successfully accomplish specified tasks in an environment that is open, dynamic, and not completely accessible. Agent-based systems are often mission critical (or even safety critical) and, like other software systems, must be tested and evaluated before being deployed. The agent's environment is crucial for evaluating its performance (Helleboogh et al., 2007). This motivates another relation between agents and simulation, namely that simulation can help in the design of multiagent systems by providing the necessary synthetic realistic and dynamic environment. S.Hanks et al. (1993) propose to use such simulated environments as test-bed for agents, as this approach allows evaluating the agent behavior under all desired circumstances. They have been used to get a better understanding of the general behavior and performance of certain agent strategies. In (Schattenberg and Uhrmacher, 2001), different types of commitments of planning agents – such as when to stick to certain goals or when to take new options into consideration – have been explored in the Tileworld scenario. A crucial question remains: how to transfer the insights gained in those small world scenarios to realistic, specific applications.

Generally, testing checks an implementation against specified requirements. To this end, a set of test cases is generated to cover the requirements. If testing cannot be done exhaustively, selecting suitable test cases becomes crucial. Model-based testing reduces the set of test cases explicitly. Abstractions are developed that take the purpose of testing into account (Broy et al., 2005). Simulation-based testing of software agents makes abstraction explicit including data abstraction, temporal abstraction, communication abstraction, and functional abstraction (Röhl and Uhrmacher, 2005). This simulation-based testing can be applied to the evaluation of a deliberative agent in its virtual environment, for example, in (Gierke et al., 2006) for evaluating the Autominder (Pollack, 2006) software (Figure 3.6), and for evaluating communities of reactive agents, for example, different user models and protocols in mobile ad hoc networks (Röhl et al., 2007).

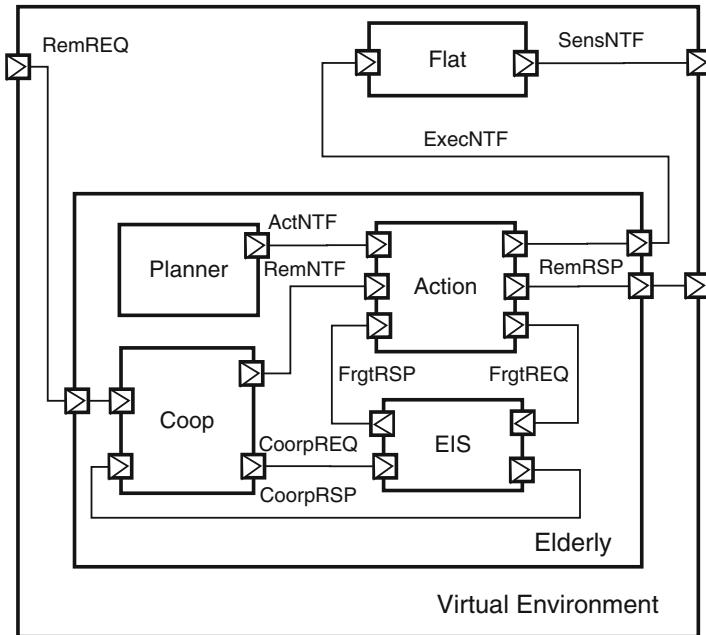


Fig. 3.6 Simulation-based environment for testing Autominder.

In this type of testing, the representation of humans plays an important role. If, for example, a smart environment supports elderly people at home, a valid – or at least plausible – model of these elderly people is required to test the developed software. A human behavior representation (HBR) is a “model that mimics either the behavior of a single human or the collective action of a team of humans” (Panel on Modeling Human Behavior and Simulations, Command Decision Making: Representations for Military, 1998). Research on HBRs has largely been driven by the military community – under the lead of the former US Department of Defense Modeling and Simulation Office (DMSO) (see (Cacciabue, 1998)) – and the social sciences community (Schmidt, 2000; Suleiman et al., 2000). Both communities traditionally apply areas of simulation and have supported a significant line of research on human–machine interaction (HMI). As in other fields, using the agent metaphor the focus has shifted from reactive toward “representation of the decision making processes [...] of humans” (Wise et al., 2001) in HBR, and thus toward deliberative approaches. It should be pointed out that the main objective of this work was to exploit human cognition to increase the efficiency of testing. The goal was not to add to the body of knowledge on human cognition.

The emphasis on deliberative approaches becomes apparent in the generic structure of HBRs suggested by Harmon et al. (Harmon et al., 2001), see Figure 3.7. The figure highlights some typical agent features: the model maintains an internal state and contains a representation of its working environment. In order to interpret percepts, update the state, and select an action that is likely to help achieve high-

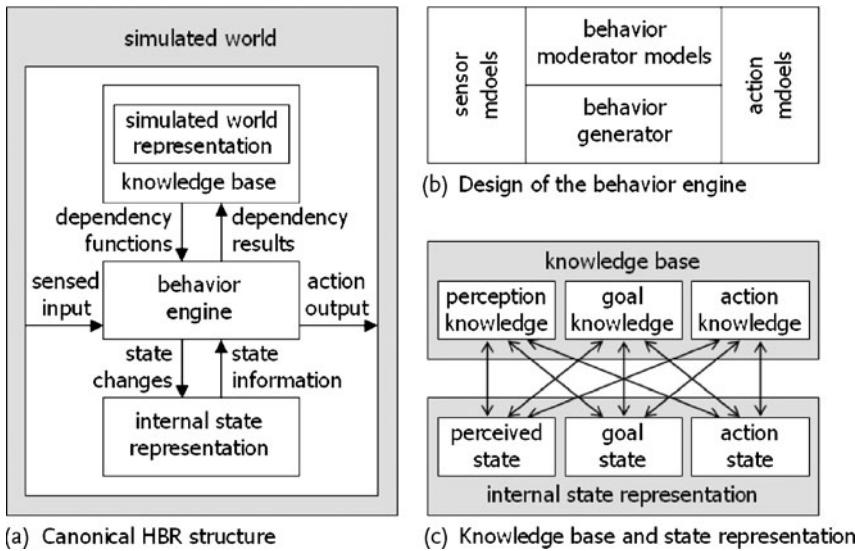


Fig. 3.7 Canonical HBR structure. From (Harmon et al., 2001).

priority goals, the behavior engine uses a knowledge base (KB). For every feature tracked in the internal state, the KB has to provide adaptation rules and (optional) prior knowledge. The behavior engine is internally divided into four subcomponents: a sensor and a motor model that couples the HBR with its environment, a behavior generator responsible for selecting an appropriate action depending on the current situation – the cognitive processing – and behavior moderator models. The latter must be understood as noncognitive factors that boost or limit the cognitive performance of the HBR. Harmon et al. decide between internal moderators (such as personality features or emotions) and external stressors, which reflect a high cognitive workload or physiological conditions (e.g. fatigue, weather).

3.4.4

Simulation as Support for Decision Making in Systems Engineering

A major application domain of simulation is to support decisions by carrying out “what-if” analysis to evaluate complex, dynamic operating scenarios. Using simulation in such a way has several implications. One is that the simulation has to run faster than real time. Designing efficient simulation systems is not trivial, particularly if the simulation contains multiple lightweight and heavyweight agents. Furthermore, current data can be exploited to be fed into the simulation during runtime. Although this is not a new idea, particularly in the context of supporting decisions, for example, in (Krusche and Tolk, 2000; Wieland, 1998) the idea of “symbiotic” simulation stresses the benefit that both simulation and physical system receive by their close interaction (DAGSTUHL, 2002; Ayed et al., 2008). The simulation benefits from the data, as its predictions will be more accurate, and

the simulation results can be validated, and the physical system benefits from the simulation as its operation will run in an optimized manner based on the online analysis of the simulation results. Thus, longer cycle times in the process of model update, analysis, and verification can be avoided and what-if analysis to respond to abrupt changes in the system is facilitated. This approach is also called data driven.

As the simulation engine runs under time constraints and interacts with a physical system, it can be designed using the agent metaphor itself (Low et al., 2005); this is particularly true if concurrent, distributed online simulation engines have to be synchronized (Fujimoto et al., 2007). This collection of autonomous online simulations offer the potential of increased accuracy, responsiveness, and robustness compared to centralized approaches. Unlike other distributed simulations they are created bottom-up rather than top-down, and they are data driven. Those real-time, data-driven simulations can be deployed on multiple platforms: vehicles, roadside cabinets, transportation management center servers, and so on. Simulations close to data sources via in-vehicle and roadside online simulators can be executed and can immediately share observations and predictions (Fujimoto et al., 2007).

3.4.5

Implications for Modeling and Simulation Methods

Not surprisingly, object- and agent-oriented approaches in modeling and simulation are closely related (Uhrmacher, 1997). Agents have a comparatively long tradition in modeling and simulation. For example, the concept of endomorphic models refers to models that have a (meta-)model about the model (Zeigler, 1990), which implies possibilities for self-reflection and beliefs about its environment. The impact of agent-based modeling expresses itself in specific simulation tools, modeling, and simulation methods. Many of the simulation tools are open source projects – implemented in Java – that provide a kernel containing simulation utility functions and offer some domain-specific libraries. Some require programming knowledge, like Repast (North and Macal, 2007), while others support a graphical description of models, like SeSAM (Klügle, 2008).

Only a few of the simulation tools that are in use for simulating multiagents exploit specific modeling formalisms. As agent activities are typically perceived as discrete or hybrid, we find the adoption of modeling formalisms that support hybrid systems modeling, for example, hybrid automata in Charon (Alur et al., 2000) or discrete event modeling like DEvs in James II (Himmelsbach and Röhl, 2008; Uhrmacher, 2001). The use of modeling formalisms is often limited to the description of select aspects of agents, like communication protocols between agents in Petri nets (Cost et al., 1999) or state changes of agents as hybrid automata (Alur et al., 2000).

To adopt formalisms like DEvs, Petri nets, or hybrid automata for a more comprehensive agent modeling means their adaptation. Many formalisms assume a static communication and composition structure, whereas dynamic patterns of interaction and composition including migration are salient features of agents. In DEvs (Zeigler, 2000), this problem is addressed in various extensions introducing

a controller at the level of the coupled model, as described in (Barros, 1997). This new type of controller is responsible for creating, deleting, or changing the interaction of agents. Alternatively, models can be equipped with the ability to change their own structure in a reflective manner, as described in (Uhrmacher, 2001). Similar developments can be observed in adopting Petri nets as a formalism to describe the dynamics of multiagent systems. In (Asperti and Busi, 1996), mobile Petri nets were introduced to demonstrate that the firing of transitions produced new nets, or in reference nets tokens are interpreted as nets in order to support migrating agents (Köhler et al., 2003). Not too surprisingly, extensions have been suggested for hybrid automata more recently as well. An example is given in (Kratz et al., 2006), where extensions are defined to cope with dynamic patterns of interaction and composition, and thus support a dynamic reconfiguration of the system.

Agent-oriented approaches have also had an effect on simulation methods. Uhrmacher (2002) shows that several new challenges arise in the context of evaluating software agents by simulation-based approaches. A prominent example for such new challenges documented in detail in (Himmelsbach et al., 2003) is the synchronization problem of simulation software implementing the testbed and the software agents under test. Asynchronous interaction provides a loose coupling between simulation and agents, as is exemplified in RoboCup and RoboCup Rescue scenarios (Takahashi, 2008). However, if the agent is invoked as an external source and time pressure plays a role, a suitable time model has to be found that relates external resource consumption to simulation time (Riley, 2003).

Another challenge refers to efficiently simulating those multiagent systems (Theodoropoulos et al., 2008; Uhrmacher and Kullick, 2000). Parallel distributed simulation engines face different problems. Look-aheads are often difficult to define, so conservative parallel distributed approaches are not very effective. Due to the dynamic patterns of interaction and composition, rollbacks implied by optimistic simulations tend to be expensive. If, in addition, external sources interact with the simulation and data are stored outside the simulation, rollbacks might lead to inconsistencies. In addition to synchronization, the organization of large shared state space also poses specific problems. Agents act and often interact in an environment assessing and accessing information (Helleboogh et al., 2007). If a simulation runs in a distributed environment, optimizing access to the shared data is crucial for achieving efficient simulation executions. General efficient strategies to address all these issues have been elusive, as they have shown to be highly model sensitive, for example, (Ewald et al., 2006). Driven by the variety of multiagent systems, which might comprise many thousands of lightweight agents and/or a small number of heavyweight deliberative agents, researchers have therefore started to focus on the design of a flexible simulation engine where different algorithms and data structures can be chosen and combined on demand (Himmelsbach and Uhrmacher, 2007; Himmelsbach and Röhl, 2008).

Another effect that agent research has had on modeling and simulation methods has been in the context of model composition and simulation interoperation (Tolk, 2006; Yilmaz and Tolk, 2006). Current simulation protocols are focused on the definition of standardized information exchange, like Protocol Data

Units in IEEE1278 (of Electrical and Engineers) or Federation Object Models in IEEE1516 (of Electrical and Engineers). This is not sufficient for model composition. In addition, the matching of simulation internal data to these information exchange elements is typically hard-coded. Supporting reuse at the modeling – or conceptual – level, other methods developed for agents have become of interest, for example, metadescription and ontologies for selecting suitable models and for relating different modeling formalisms (Tolk et al., 2007).

3.5

Agent Taxonomies

Taxonomies are defined as *tree structures of classifications* for a given set of objects. At the top of these structures are single classifications, which are the root nodes that apply to all objects. Nodes below these roots are more specific classifications that apply to subsets of the total set of classified objects. The main purpose is the classification of terms. The higher a term, the more universal it is; this means that leaves are the most specific terms of taxonomies. As with the other topics covered so far, the variety of agent approaches has led to various attempts to structure the realm of agents by developing taxonomies. So, not surprisingly, we find different agent taxonomies emphasizing different points of view in the agent literature.

As before, in this chapter we choose an approach that highlights some of the influential works in this domain focusing on the common ideas applicable in the context of using agents and simulations for and with systems engineering methods. These are examples and are neither complete nor exclusive. Furthermore, it should be pointed out that a merged view does not necessarily result in a tree structure, as different viewpoints lead to different categories, and these different categories cannot always be mapped to a perfect tree. This is well known from other sciences, in particular biology and chemistry, where different viewpoints often lead to different taxonomies.

Our approach is driven by applicability. We will focus on categories of particular interest when agent-directed simulation is applied in the context of systems engineering or vice versa. The resulting structure is a set of terms engineers need to agree on when describing their systems.

3.5.1

History and Application-Specific Taxonomies

One of the first papers written with the objective of setting up a common agent taxonomy was written by Franklin and Graesser (Franklin and Graesser, 1996). We already used several of their arguments to define agenthood. Franklin and Graesser analyzed several approaches to agent definitions; among those are the same definitions we referred to earlier in this chapter: Wooldridge and Jennings (Wooldridge and Jennings, 1995) and Hayes-Roth (Hayes-Roth, 1995). They use the following working definition: “An autonomous agent is a system situated within and a part

of an environment that senses that environment and acts on it, over time, in pursuit of its own agenda and so as to affect what it senses in the future". The resulting taxonomy is relatively simple. The autonomous agents are categorized into *biological*, *robotic*, and *computational* agents. The computational agents are subcategorized into *artificial* and *software* agents. The software agents are further categorized into *task-specific* agents, *entertainment* agents, and *viruses*. Figure 3.8 shows this taxonomy.

A very similar view was introduced by Sánchez in (Sánchez, 1997). His root term is the *software* agent. The reason we include his work in this chapter is that his motivation is very close to our goal in this chapter. In his report, he states that "first, computer scientists and software engineers have used agents as an abstraction to conceptualize, design and implement complex systems. This class of agents will be referred to as *programmer* agents. Second, agents may be viewed as autonomously migrating entities that act on behalf of network nodes in a distributed environment. This class will be termed *network* agents. Finally, agents have been proposed as an abstraction for end users to interact with computer systems. This view defines the class referred to as *user agents*". The user agents are then subcategorized into *information* agents (helping users to deal with unorganized and highly dynamic information spaces), *synthetic* agents, which create engaging environments for users by introducing lifelike characters into the computer interface, and *task* agents, which help users perform computer-supported tasks. The task agents are furthermore categorized into *personal* and *group* agents. This taxonomy is captured in Figure 3.9.

Before introducing our recommendations on structuring the agent space, two additional examples will be presented. The first approach categorizes agents within the domain of distributed artificial intelligence and, as such, introduces new aspects regarding applicable methods and system specifications. The second approach focuses on the agent characteristics of communications. Both approaches are given as examples of how taxonomies can be used in a more general as well as more specific context. What level of detail is right has to be decided by the engineers working on the underlying application.

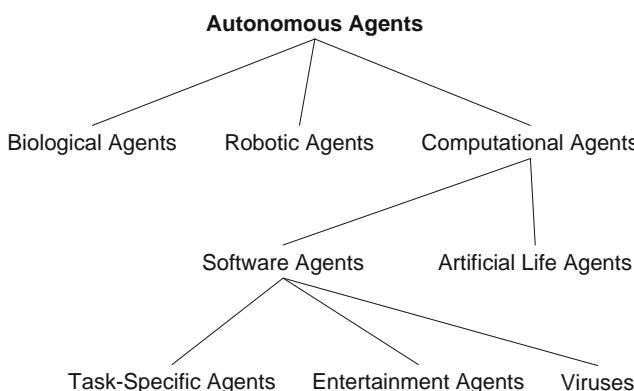


Fig. 3.8 Agent taxonomy of Franklin and Graesser (1996).

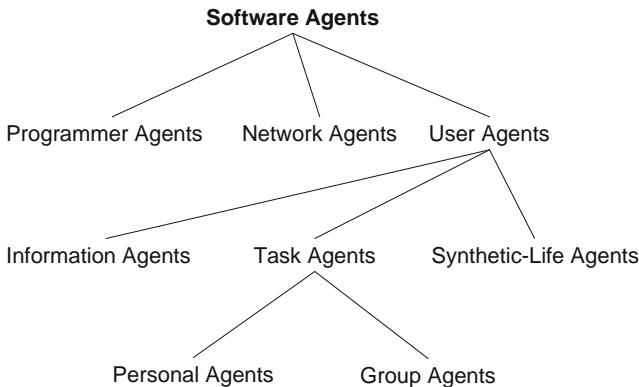


Fig. 3.9 Agent taxonomy of Sánchez (1997).

- Decker compiled a taxonomy of multiagent systems based on an extensive survey of applied research and development in industrial applications of multiagent systems in (Decker, 1996). His top categories are
 - The application domain in which the agent system will be applied,
 - The architecture of an individual agent, and
 - The architecture of the system in which the agents interact.
 He introduces subcategories to describe granularity (coarse vs. fine), heterogeneity of agent knowledge (redundant vs. specialized), methods of distributing control (benevolent vs. competitive, team vs. hierarchical, static vs. shifting roles), and communication methods (blackboard vs. messages, low-level vs. high-level content).
- When evaluating the possibility of using agents and agent-directed simulation for systems engineering, the principles applied by Decker become interesting, as they show a way to embed special taxonomical ideas of agents in the broader context of a system (as systems are designed to support an application domain in which the agent system will be applied).
- Van Dyke Parunak and colleagues introduced a design taxonomy of multiagent interactions in (Parunak et al., 2004). Their motivation was to better specify the agent characteristics of communications. As agents are designed for a purpose (“Agents do things together”), a supporting taxonomy was introduced to describe the joint behavior of agents. They define and relate correlation, coordination, cooperation (and contention), congruence, and coherence as their central terms. The taxonomy provides a basis for reviewing agent engineering methods and modeling languages. Later, Weyns and Holvoet extended this approach in (Weyns and Holvoet, 2003) regarding synchronous and asynchronous collaboration, adding an additional layer of detail.

Van Dyke Parunak introduced another interesting viewpoint: What do engineers need to know if they want to integrate an agent system into their solution?

In (Parunak, 1996), he introduces the categories system function (what functionality is provided), agent architecture (degree of heterogeneity, reactive vs. deliberative), and system architecture (communication, protocols, human involvement) as terms of interest in this context, which are directly mappable to the high-level categories identified in (Decker, 1996).

An alternative approach often seen in practice is to define software agents based on the application needs first and derive taxonomies of such agents from their definition “after the fact”, more or less as an afterthought in the process of documenting the implemented system. A summary of this approach and a proposal on defining agents in a systematic manner has been published in (Höppner, 2003). By comparing the resulting agent taxonomies, it is possible to generalize them into an upper taxonomy.

3.5.2

Categorizing the Agent Space

As stated before, we do not want to introduce another agent taxonomy but rather to highlight categories that deal with the important issues when agent-directed simulation and systems engineering come together. These categories are based on the framework proposed by Moya and Tolk (Moya and Tolk, 2007). This work is not standardized and does not claim to be exclusive or complete. However, in contrast to many earlier taxonomy approaches, this effort also takes the environment into account.

The emphasis of most taxonomy works mentioned before are the agent characteristics. Besides the work of Wooldridge and Jennings (Wooldridge and Jennings, 1995; Wooldridge, 2000), additional overview publications from related domains, such as (Nwana, 1995), contribute to defining the categories. The following categories are proposed to define agent characteristics:

- *Perception*: How the agent perceives its environment is defined by complete or partial access, the accuracy of the perception, and the recognition process, as well as other points.
- *Decision making*: Reasoning and decision making were dealt with in detail earlier in this chapter.
- *Communication*: Categories and subcategories define the method of negotiation (such as broker, blackboard, and so on) and protocols and languages used.
- *Action*: Finally, the way to act needs to be categorized. Subcategories define if the actions are, for example, reactive, deliberative, or preplanned, or if hybrid forms are used.

It is noteworthy that speech act theory (Searle, 1969) considers communication to be a special form of action, and therefore it can be argued whether communication is just a subset of action. However, most publications distinguish between communication and action as different concepts, which is the view supported in this chapter as well.

In addition to agent characteristics, *situated environment* and *agent society* are introduced as new top-level categories for agent systems.

- The *situated environment* is everything around the autonomous agent. However, for the taxonomy we exclude other agents as they are treated in the agent society. The environment itself is characterized by five categories (Wooldridge, 2000):
 - The environment can be accessible or nonaccessible. This category is different from the question of how the agent perceives what is available and if it can make sense of what it perceives.
 - In deterministic environments, an action has one effect. In stochastic environments, this is not the case.
 - The environment can be episodic or sequential. In episodic environments, the action is only relevant for the current episode. In sequential environments, an action may have effects in future states as well.
 - The environment can be static or dynamic.
 - The environment can be discrete or continuous. Furthermore, discrete environments can differ at the level of resolution, accuracy, and granularity.
- The agent society of an agent system is the collection of all agents and how they act in the situated environment. It is characterized by size (number of agents) and diversity (types of agents). As subcategories, we introduce the following concepts:
 - The number of agents within the population can vary between large-scale numbers of several thousand agents and just a few agents.
 - The agents can cooperate with each other or be in competition. In addition, all mixed forms are possible, such as coalitions that cooperate with each other but compete with others.
 - The society can comprise agents that are all of the same type (homogeneous society) or agents of different types (heterogeneous society). It is worth mentioning that even agents of the same type can exhibit different behavior depending on their state and initialization.
 - The society can be open or closed. In open societies, anyone can contribute agents and add them to the society. In closed societies, the number of contributors is limited by constraints. Again, mixed forms are possible.

Figure 3.10 displays the top-level categories for the proposed taxonomy reflecting the characteristics for agenthood. The categories are neither complete nor exclusive, but they have proven to be a valuable guideline in several projects.

As pointed out before, this resulting categorization is the starting point for a community that is meant to become a hub, and it needs to be extended with respect to scope and resolution. However, the alignment of agenthood characteristics, architectural constraints, and taxonomical structures is essential. Yilmaz and Tolk (2008) extend the ideas described in this section in the context of agent-based decision support.

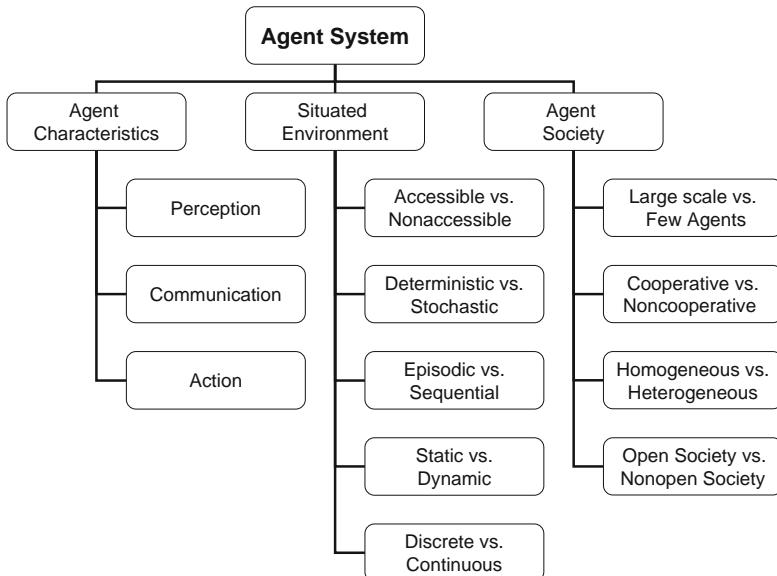


Fig. 3.10 Agent taxonomy reflecting agenthood characteristics.

3.6 Concluding Discussion

Agents appear as a natural successor of objects. In this chapter, we described agents as autonomous and adaptive objects as well. They undoubtedly have a significant impact on modeling and simulation, but the opposite is also true: simulation as an experimental technique improves the understanding of agent systems and their design. Agents offer for simulation:

- A nonmonolithic flexible model design, characterized by a dynamic pattern of interaction and composition,
- A natural metaphor and suitable methods to describe human behavior representation,
- Methods to support the interoperation of simulation systems.

In the opposite direction, simulation can significantly contribute to the agent software engineering process, by supporting:

- Testing of individual agents,
- Testing of agent societies,
- Systematic simulation-based generation of test cases,
- A better understanding of the environment in which the agent is situated, an aspect that has recently attracted attention (Weyns et al., 2005).

Both simulation and agents contribute to systems engineering, the more so the more those systems exhibit some of the characteristics of agents, for example, be-

ing flexible, autonomous, and situated. Agent-based approaches to systems engineering offer potential advantages over alternative traditional approaches in systems engineering:

- The adaptability of agent-based systems offers increased flexibility and robustness in dynamic or unpredictable environments.
- Agents are able to react to opportunities but at the same time to balance this against their commitment to specific goals, an ability that comes in handy in many situations, from controlling individual robots up to entire enterprise processes.
- Agent methods can be exploited to facilitate knowledge sharing within an organization.

However, agents offer not only advantages. There is also the danger that agents will be used in simulation and systems engineering merely for the sake of using agents, although traditional methods are applicable and may even be preferable in some cases. The agent metaphor is flexible enough to allow such misuse. As pointed out by Shoham (Shoham, 1994), it is “*perfectly coherent to treat a light switch as a very cooperative agent with the capability of transmitting current at will.... It does not buy us anything, since we essentially understand the mechanism sufficiently to have a simpler, mechanistic description of the behavior*”. Only if the use of agents has a real benefit for the system to be designed or the solution for a problem to be solved will an engineer apply this technology, see also (of Electrical and Engineers, 2006). Without doubt, agent-directed simulation and systems engineering play an important role in systems engineering, but the use of agent-based approaches must be motivated by necessities derived from the problem to be solved.

Wooldridge and Jennings (Wooldridge and Jennings, 1998) enumerate several mistakes observed in real-world agent-based applications that also apply to modeling, simulation, and systems engineering.

- *Overselling agents*: The limits of agents are not taken into account, which is often based on insufficient understanding of agents – or the problem to be solved.
- *Getting dogmatic*: Agents are applied for the sake of applying agents. Wooldridge and Jennings warn against “getting religious about agents”.
- *Confusing prototypes with systems*: The challenge is not to develop a prototype system consisting of a few interacting agents doing some semieuful task, but to develop a solution that is sufficiently robust and reliable to be used in practice.
- *Insufficiency of pragmatic and engineering aspects*: Methodologies for engineering agent-based systems are still under development.

In summary, despite these critical words at the end of the chapter, agents as autonomous and adaptive objects have great potential. If applied in the right context using well-defined engineering methods, agent-based applications in general – and ADS in particular – can contribute significantly to systems engineering. This book

and this chapter aim to contribute to the development of these engineering methods and to the education of professionals and students.

References

- Agre, P.E. and Chapman, D. (1990) What are plans for? *Designing Autonomous Agents: Theory and Practice from Biology to Engineering and Back* (ed. P. Maes), pp. 17–34. The MIT Press: Cambridge, MA, USA.
- Alessio, H.P. and Smith, C.F. (2005) *Developing semantic web services*, A.K. Peters, Ltd.
- Alur, R., Grosu, R., Hur, Y., Kumar, V. and Lee, I. (2000) *Modular specification of hybrid systems in charon*. Proceedings of the 3rd International Workshop on Hybrid Systems: Computation and Control.
- Argawal, S., Handschuh, S. and Staab, S. (2005) Annotation, composition and invocation of semantic web services. *Journal of Web Semantics*, 2 (1), 1–24.
- Aspertini, A. and Busi, N. (1996) Mobile Petri nets. Technical Report UBLCS-96-10, University of Bologna.
- Axelrod, R. (1997) *The Complexity of Cooperation: Agent-Based Models of Competition and Collaboration*. Princeton Studies in Complexity, Princeton.
- Axtell, R. (2000) Why agents? on the varied motivations for agent computing in the social sciences. Working Paper No. 17.
- Aydt, H., Turner, S.T., Cai, W. and Low M.Y.H. (2008) An agent-based generic framework for symbiotic simulation systems, in *Multi-Agent Systems: Simulation and Applications* (eds A.M. Uhrmacher and D. Weyns), Taylor and Francis.
- Balmer, M., Cetin, N., Nagel, K. and Raney, B. (2004) *Towards truly agent-based traffic and mobility simulations*. AAMAS '04: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems, Washington, DC, USA, IEEE Computer Society, pp. 60–67.
- Barros F.J. (1997) Modeling formalisms for dynamic structure systems. *ACM Transactions on Modeling and Computer Simulation*, 7 (4), 501–515. <http://doi.acm.org/10.1145/268403.268423>.
- Baumann, J., Hohl, F. and Rothermel, K. (1997) Mole – concepts of a mobile agent system. Technical Report TR-1997-15. citeseer.ist.psu.edu/baumann97mole.html.
- Bazzan, A., Klügl, F. and Ossowski, S. (2005) *Agents in Traffic and Transportation* (Whitestein Series in Software Agent Technologies), Birkhauser.
- Boohr, R.H. (ed.) (2003) *Handbook of Human Systems Integration*, John Wiley & Sons, New York, NY.
- Botelho, L., Willmott, S., Zhang, T. and Dale, J. (2002) Review of content languages suitable for agent-agent communication. Technical report.
- Bratman, M.E., Israel, D. and Pollack, M.E. (1988) Plans and resource-bounded practical reasoning. *Computational Intelligence*, 4, 349–355.
- Brooks, A.R. (1991a) Intelligence without representation. *Artificial Intelligence*, 47, 139–159.
- Brooks, R. (1986) A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2, 14–23.
- Brooks, R.A. (1991b) *Intelligence without reason*. Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI-91), (eds J. Myopoulos and R. Reiter), Sydney, Australia, Morgan Kaufmann publishers Inc., San Mateo, CA, USA, pp. 569–595. citeseer.ist.psu.edu/article/brooks91intelligence.html.
- Broy, M., Jonsson, B., Pieter Katoen, J., Leucker, M. and Pretschner, A. (2005) *Model-Based Testing of Reactive Systems: Advanced Lectures (Lecture Notes in Computer Science)*, Springer-Verlag New York, Inc., Secaucus, NJ, USA.
- Cacciabue, P.C. (1998) Modelling and simulation of human behaviour in system control, Advances in industrial control. Springer, London.
- Casagni, M. and Lyell, M. (2003) *Comparison of two component frameworks: the fipa-compliant multi-agent system and the web-centric j2ee platform*. ICSE '03: Proceedings of the 25th International Con-

- ference on Software Engineering, Washington, DC, USA, IEEE Computer Society, pp. 341–351.
- Chalupsky, H., Finn, T., Fritzson, R., McKay, D., Shapiro, S., and Wiederhold, G. (1992) *An overview of KQML: A knowledge query and manipulation language*, Defense Advanced Research Project Agency (DARPA) White Paper, Washington, DC.
- Conte, R., Gilbert, N. and Sichman, J.S. (1998) MAS and social simulation: A suitable commitment. *MABS'98*, vol. 1534 of *Lecture Notes in Computer Science*, Springer, pp. 1–9.
- Cost, R.S., Chen, Y., Finin, T., Labrou, Y.K. and Peng, Y. (1999) *Modeling agent conversations with colored Petri nets*. Working notes of the Autonomous Agents '99 Workshop on Specifying and Implementing Conversation Policies, Seattle, WA.
- Davis, A.G. and Ritsko, J.J. (2003) Special issue on e-business management. *IBM Systems Journal*, 42 (3).
- Decker, K.S. (1996) Distributed artificial intelligence testbeds, in *Foundations of Distributed Artificial Intelligence*, (eds G.M.P. O'Hare and N.R. Jennings), John Wiley and Sons, New York, NY, pp. 119–138.
- Dickinson, I. and Wooldridge, M. (2005) *Agents are not (just) web services: considering bdi agents and web services*. Proceedings of the 2005 Workshop on Service-Oriented Computing and Agent-Based Engineering (SOCABE'2005).
- d'Inverno, M., Kinny, D., Luck, M. and Wooldridge, M. (1997) A formal specification of dMARS, in *Agent Theories, Architectures, and Languages*, vol. 1365 of *Lecture Notes In Computer Science*, Springer, Berlin/Heidelberg, pp. 155–176.
- Doran, J. (1997) From computer simulation to artificial societies. *Transactions of the Society for Computer Simulation*, 14 (2), 69–77. Special Issue: Multi-Agent Systems and Simulation.
- Epstein, J.M. (1999) Agent-based computational models and generative social science. *Complexity*, 4 (5), 41–60.
- Epstein, J.M. and Axtell, R. (1996) *Growing Artificial Societies – Social Science from the Bottom Up*. Washington, Brookings Institution Press/MIT Press.
- Esser, J. and Schreckenberg, M. (1997) Microscopic simulation of urban traffic based on cellular automata. *International Journal of Modern Physics C*, 8 (5).
- Eulalia, S., Frayrat, M.J. and D'Amours, S. (2007) Agent-based simulation for distributed supply chain planning: Conceptual modelling, analysis, and illustration. Technical Report CIRRELT-11-2007, CIRRELT.
- Ewald, R., Chen, D., Theodoropoulos, K.G., Lees, M., Logan, B., Ogura, T. and Uhrmacher, A.M. (2006) *Performance analysis of shared data access algorithms for distributed simulation of MAS*. 20th Workshop on Principles of Advanced and Distributed Simulation (PADS'06), Los Alamitos, CA, USA, IEEE Computer Society, pp. 29–36.
- Ewert, U.C., Röhl, M. and Uhrmacher, A.M. (2007) Hunger and market dynamics in pre-modern urban communities: insights into the effects of market intervention from a multi-agent model. *Historical Social Research*, 32 (4), 122–150.
- FIPA The foundation for intelligent, physical agents. <http://www.fipa.org/>. Last date of access: 30 June 2009.
- Franklin, S. and Graesser, A. (1996) *Is it an agent, or just a program? A taxonomy for autonomous agents*. Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages.
- Fujimoto, R.M., Lunceford, W.H., Page, E.H., and Uhrmacher, A.M. (2002) Grand challenges for modeling and simulation seminar at Dagstuhl – a report. Technical Report Dagstuhl Seminar Report 02351, Schloss Dagstuhl, Germany.
- Fujimoto, R., Hunter, M., Sirichoke, J., Palekar, M., Kim, H. and Suh, W. (2007a) *Ad hoc distributed simulations*. PADS '07: Proceedings of the 21st International Workshop on Principles of Advanced and Distributed Simulation, Washington, DC, USA, IEEE Computer Society, pp. 15–24. <http://dx.doi.org/10.1109/PADS.2007.10>.
- Fujimoto, R.M., Guensler, R., Hunter, M., Schwan, K., Kim, H.K., Seshasayee, B., Sirichoke, J. and Suh, W. (2007b) *Ad hoc distributed simulation of surface transportation systems*. International Conference on Computational Science (1), pp. 1050–1057.

- Fukuda, M., Bic, L.F., Dillencourt, M.B. and Merchant, F. (2001) Messengers: Distributed programming using mobile agents. *Journal of Integrated Design and Process Science*, 5 (4), 95–112.
- Gasser, L. and Briot, J.P. (1992) *Object-based concurrent programming and distributed artificial intelligence*. Distributed Artificial Intelligence: Theory and Praxis, Kluwer Academic publishers, Boston, MA, USA, pp. 81–108. citeseer.ist.psu.edu/gasser92objectbased.html.
- Genesereth, M.R. and Fikes, R.E. (1992) Knowledge interchange format, version 3.0 reference manual. Technical Report Logic-92-1, Stanford, CA, USA.
- Gierke, M., Himmelsbach, J., Röhl, M. and Uhrmacher, A.M. (2006) Modeling and simulation of tests for agents, in *Multi-Agent System Technologies (MATES'06)*, vol. 4196/2006 of *Lecture Notes in Computer Science*, Springer Berlin/Heidelberg, pp. 49–60. URL http://dx.doi.org/10.1007/11872283_5.
- Gilbert, N. and Troitzsch, K.G. (1999) *Simulation for the Social Scientist*, Open University Press.
- Gray, R.S., Kotz, D., Cybenko, G. and Rus, D. (1998) D'agents: Security in a multiple-language, mobile-agent system. *Mobile Agents and Security*, London, UK, Springer Verlag, pp. 154–187.
- Grimm, V. and Railsback, S.F. (2005) *Individual-based Modeling and Ecology*. Theoretical and Computational Biology. Princeton University Press.
- Hanks, S., Pollack, M.E. and Cohen, P.R. (1993) Benchmarks, test beds, controlled experimentation and the design of agent architectures. *AI Magazine*, 14 (4), 17–42.
- Halpin, B. (1999) Simulation in sociology. *American Behavioral Scientist*, 42 (10), 1488–1508.
- Harmon, S.Y., Hoffman, C.W.D., Gonzalez, A.J., Knauf, R. and Barr, V.B. (2001) Validation of Human Behavior Representations. In Foundations for V & V in the 21st Century Workshop (Foundations '02), San Diego, CA, Society for Modeling and Simulation International, pp. B3-1–B3-34.
- Hayes-Roth, B. (1995) An architecture for adaptive intelligent systems. *Artificial Intelligence: Special Issue on Agents and Interactivity*, 72, 329–365.
- Helleboogh, A., Vizzari, G., Uhrmacher, A.M. and Fabien, M. (2007a) Modeling dynamic environments in multi-agent simulation. *Journal of Autonomous Agents and Multi-Agent Systems*, 14 (1), 87–116.
- Helleboogh, A., Vizzari, G., Uhrmacher, A.M. and Fabien, M. (2007b) Modeling dynamic environments in multi-agent simulation. *Journal of Autonomous Agents and Multi-Agent Systems*, 14 (1), 87–116.
- Himmelsbach, J. and Röhl, M. (2008) JAMES II – experiences and interpretations. *Agents, Simulation and Applications*, (eds A.M. Uhrmacher and D. Weyns), Taylor and Francis.
- Himmelsbach, J. and Uhrmacher, A.M. (2007) *Plug'n simulate*. Proceedings of the 40th Annual Simulation Symposium, IEEE Computer Society, pp. 137–143.
- Himmelsbach, J., Röhl, M. and Uhrmacher, A.M. (2003) *Simulation for testing software agents – an exploration based on JAMES*. Proceedings of the 2003 Winter Simulation Conference, (eds S. Chick, P.J.S. Sanchez, D. Ferin and D.J. Morrice).
- Höppner, S. (2003) An agents' definition framework and a methodology for deriving agents' taxonomies, in *KI 2003: Advances in Artificial Intelligence*, vol. 2821 of *Lecture Notes in Computer Science*, Springer Berlin/Heidelberg, pp. 618–632.
- Institute of Electrical and Electronics Engineers. IEEE standard 1278 series: Distributed simulation protocol (DIS). Obtainable via <http://www.ieee.org/portal/site>, Last date of access: 30 June 2009.
- Institute of Electrical and Electronics Engineers (2006) IEEE standard 1516 series: High level architecture (HLA). Obtainable via <http://www.ieee.org/portal/site>, Last date of access: 30 June 2009.
- Institute of Electrical and Electronics Engineers. *IEEE Code of Ethics*. Number 7.8 in IEEE Policy. February 2006.
- Jennings, N., Faratin, P., Lomuscio, A., Parsons, S., Sierra, C. and Wooldridge, M. (2001) Automated negotiation: prospects, methods and challenges. *International Journal of Group Decision and Negotiation*, 10 (2), 199–215.

- Jennings, N.R., Sycara, K. and Wooldridge, M. (1998) A roadmap of agent research and development. *Autonomous Agents and Multi-Agent Systems*, 1 (1), 7–38. <http://dx.doi.org/10.1023/A:1010090405266>.
- Keating, C., Rogers, R., Unal, R., Dryer, D., Sousa-Poza, A., Safford, R., Peterson, W. and Rabadi, G. (2003) System of systems engineering. *Engineering Management Journal*, 15 (3), 36–45.
- Klügle, F. (2008) Sesam: Visual programming and participatory simulation for agent-based models, in *Agents, Simulation and Applications*, (eds A.M. Uhrmacher and D. Weyns), Taylor and Francis.
- Köhler, M., Moldt, D. and Rölke, H. (2003) *Modelling mobility and mobile agents using nets within nets*, in Proceedings of the 24th International Conference on Application and Theory of Petri Nets 2003 (ICATPN 2003) (eds W. van der Aalst and E. Best), vol. 2679 of *Lecture Notes in Computer Science*, pp. 121–139. Springer.
- Kotz, D. and Gray, R.S. (1999) Mobile agents and the future of the internet. *SIGOPS Operating System Review*, 33 (3), 7–13. <http://doi.acm.org/10.1145/311124.311130>.
- Kratz, F., Sokolsky, O., Pappas, G.J. and Lee, I. (2006) R-Charon: a modeling language for reconfigurable hybrid systems. *Hybrid Systems: Computation and Control*, vol. 3927 of *Lecture Notes in Computer Science*, Springer, pp. 392–406.
- Krusche, S. and Tolk, A. (2000) Information processing as a key factor for modern federations of combat information systems. *Proceedings of the NATO Information System Technology (IST) Panel Symposium*, MP-049.
- Kruse, R., Schwecke, E. and Heinsohn, J. (1991) *Uncertainty and vagueness in knowledge based systems*, Springer, New York, NY.
- Low, M.Y.H., Lye, K.W., Lendermann, P., Turner, S.J., Chim, R.T.W. and Leo, S.H. (2005) An agent-based approach for managing symbiotic simulation of semiconductor assembly and test operation, in *AAMAS '05: Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, New York, NY, USA, ACM, pp. 85–92. <http://doi.acm.org/10.1145/1082473.1082809>.
- Maes, P. (1995) Artificial life meets entertainment: lifelike autonomous agents. *Communications of the ACM*, 38 (11), 108–114. <http://doi.acm.org/10.1145/219717.219808>.
- Marques, C.F. and Lima, P.U. (2001) Multi-sensor navigation for soccer robots, in *RoboCup*, pp. 144–153.
- Melo, F., Choren, R., Cerqueira, R., Lucena, C. and Blois, M. (2004) Deploying agents with the CORBA component model. *CD 2004*, vol. 3083 (eds W. Emmerich and A.L. Wolf), Springer, pp. 234–247.
- Moya, L.J. and Tolk, A. (2007) *Towards a taxonomy of agents and multi-agent systems*. Proceedings of the 2007 Spring Simulation Multiconference, vol. 2, pp. 11–18.
- North, M.J. and Macal, C.M. (2007) *Managing Business Complexity – Discovering Strategic Solutions with Agent-Based Modeling and Simulation*, Oxford University Press.
- North, M.J., Collier, N., Howe, T.R., Tatara, E.R., Ozik, J. and Macal, C.M. (2008) The repast simphony agent-based visual modeling system. *Agents, Simulation and Applications*, (eds A.M. Uhrmacher and D. Weyns), Taylor and Francis.
- Nwana, H.S. (1995) Software agents: An overview. *Knowledge Engineering Review*, 11 (2), 205–244.
- Odell, J.J. (2002) Objects and agents compared. *Journal of Object Technology*, 1 (1), 41–53.
- US Department of Defense. DoD directive 8100.1: Global information grid (GIG) overarching policy. Sep. 19, 2002.
- Oliver, D.W., Kelliher, T.P. and Keegan, J.G. (1997) *Engineering Complex Systems with Models and Objects*, McGraw Hill.
- Parsons, S., Gmytrasiewicz, P. and Woolridge, M. (2002) *Game Theory and Decision Theory in Agent-based Systems*, Kluwer Academic Publishers.
- Pew, R.W. and Mavor, A.S. (eds) (1998) *Panel on Modeling Human Behavior and Command Decision Making: Representations for Military Simulations*. National Research Council (1998) *Modeling Human and Organizational Behavior: Application to Military Simulations*. National Academy Press, Washington, DC.
- Pham, V.A. and Karmouch, A. (1998) Mobile software agents: an overview. *Communications Magazine, IEEE*, 36 (7), 26–37.

- Pollack, M.E. (2006) Autominder: A case study of assistive technology for elders with cognitive impairment. *Generations: The Journal of the American Society on Aging*, **30** (2), 67–69.
- Pursula, M. (1999) Simulation of traffic systems – an overview. *Journal of Geographic Information and Decision Analysis*, **3** (1), 1–8.
- Rao, A.S. and Georgeff, M.P. (1991) Modeling rational agents within a BDI-architecture. *Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning (KR'91)*, (eds J. Allen, R. Fikes and E. Sandewall), Morgan Kaufmann, pp. 473–484.
- Rao, A.S. and Georgeff, M.P. (1995) *BDI-agents: from theory to practice*. Proceedings of the First International Conference on Multiagent Systems, San Francisco, Springer.
- Riley, P. (2003) Spades a system for parallel-agent, discrete-event simulation. *AI Magazine*, **24** (2), 41–42.
- Röhl, M. and Uhrmacher, A.M. (2005) Controlled experimentation with agents – models and implementations. *Post-Proc. of the 5th Workshop on Engineering*, (eds M.-P. Gleizes, A. Omicini and F. Zambonelli), Springer. http://dx.doi.org/10.1007/11423355_21.
- Röhl, M., König-Ries, B. and Uhrmacher, A.M. (2007) An experimental frame for evaluating service trading in mobile ad-hoc networks. *Mobilität und Mobile Informationssysteme (MMS 2007)*, vol. 104 of *Lecture Notes in Computer Science*, pp. 37–48.
- Russell, S.J. and Norvig, P. (1995) *Artificial Intelligence: A Modern Approach*. Prentice Hall, Englewood Cliffs, NJ.
- El-Kader Sahraoui, A. and Jayakrishnan, R. (2005) Microscopic-macroscopic models systems integration: A simulation case study for ATMIS. *Simulation*, **81** (5), 353–363. <http://dx.doi.org/10.1177/0037549705052771>.
- Alfredo Sánchez, J. (1997) *A Taxonomy of Agents*. Interactive and Cooperative Technologies Lab, Universidad de las Américas Puebla, Cholula, Puebla, 72820 México.
- Schattenberg, B. and Uhrmacher, A.M. (2001) Planning agents in JAMES. *Proceedings of the IEEE*, **89** (2), 158–173.
- Schmidt, B. (2000) *The Modelling of Human Behaviour: Artificial Intelligence, Artificial Life, Psychology, Social Sciences*, SCS-Europe BVBA, Ghent, Belgium.
- Searle, J. (1969) *Speech Acts: An Essay in the Philosophy of Language*, Cambridge University Press.
- Shoham, Y. (1994) Multi-agent research in the knobotics group. *MAAMAW '92: Selected papers from the 4th European Workshop on Modelling Autonomous Agents in a Multi-Agent World, Artificial Social Systems*, Springer Berlin/Heidelberg, pp. 271–278.
- Silverman, B.G. (2001) More realistic human behavior models for agents in virtual worlds: Emotion, stress, and value ontologies.
- Silverman, B.G., Bharathy, G.K. and Kim G.J. (2008) Challenges of country modeling with databases, newsfeeds, and expert surveys, in *Multi-Agent Systems: Simulation and Applications* (eds A.M. Uhrmacher and D. Weynes), Taylor and Francis.
- Singh, M.P. and Huhns, M.N. (1999) Multiagent systems for workflow. *International Journal of Intelligent Systems in Accounting, Finance and Management*, **8**, 105–117.
- Smith, E.C. (1962) Simulation in systems engineering. *IBM-Systems Journal*, **1** (1), 33–50.
- Smith, R.G. (1988) The contract net protocol: high-level communication and control in a distributed problem solver, in *Readings in Distributed Artificial Intelligence*, (eds H.B. Alan and L. Gasser), Morgan Kaufmann, San Mateo, CA, USA, pp. 357–366.
- Sowa, J.F. (2000) *Knowledge Representation: Logical, Philosophical, and Computational Foundations*, Brooks/Cole, Pacific Grove, CA.
- Stone, P. and Veloso, M. (2000) Multiagent systems: A survey from a machine learning perspective. *Autonomous Robots*, **8** (3), 345–383.
- Suleiman, R., Troitzsch, K.G. and Gilbert, N. (eds) (2000) *Tools and Techniques for Social Science Simulation*, Physica-Verlag, Heidelberg.
- Takahashi, T. (2008) Robocup rescue – agent-based disaster simulation system: Challenges and lessons learned. *Agents, Simulation and Modelling*, **1** (1), 1–12.

- tion and Applications, (eds A.M. Uhrmacher and D. Weyns), Taylor and Francis.
- Theodoropoulos, G., Minson, R., Lees, M. and Ewald, R. (2008) Simulation engines for multi-agent systems. *Agents, Simulation and Applications*, (eds A.M. Uhrmacher and D. Weyns), Taylor and Francis.
- Tolk, A. (2006) What comes after the semantic web: Pads implications for the dynamic web. *20th Workshop on Principles of Advanced and Distributed Simulation (PADS'06)*, pp. 55–62. IEEE Computer Society.
- Tolk, A., Diallo, S.Y. and Turnitsa, C.D. (2007) Applying the levels of conceptual interoperability model in support of integratability, interoperability, and composableility for system-of-systems engineering. *Journal for Systemics, Cybernetics and Informatics*, 5 (5), 65–74.
- Tversky, A. and Kahneman, D. (1974) Judgment under uncertainty: Heuristics and biases. *Science*, 185, 1124–1131.
- Uhrmacher, A., Fishwick, P.A. and Zeigler, B. (eds) (2001) *Special Issue: Agents in Modeling and Simulation: Exploiting the Metaphor*. Proceedings of the IEEE, vol. 89, No. 2, 127–213.
- Uhrmacher, A.M. (2001) Dynamic structures in modeling and simulation – a reflective approach. *ACM Transactions on Modeling and Simulation*, vol. 11, No. 2, pp. 206–232.
- Uhrmacher, A.M. (2002) Simulation for agent-oriented software engineering. *First International Conference on Grand Challenges*, (eds W.H. Lunceford and E. Page), SCS, San Diego.
- Uhrmacher, A.M. (1997) Concepts of object- and agent-oriented simulation. *Transactions of the Society for Computer Simulation*, 14 (2), 59–67.
- Uhrmacher, A.M. and Kullick, G.B. (2000) “plug and test” – software agents in virtual environment. *Proceedings of the 2000 Winter Simulation Conference*, (eds J.A. Jones, R.R. Barton, K. Kang, and P.A. Fishwick), Orlando, Florida, USA, pp. 1722–1729. Wyndham Palace Resort & Spa. http://wwwmosi.informatik.uni-rostock.de/Papers_archiv/UhrmacherKullick.pdf.
- Uhrmacher, A.M. and Röhl, M. (2007) *Agent-Oriented Modeling in Simulation – Agents for Modeling and Modeling for Agents*, chapter 8, pp. 1–16, CRC.
- Uhrmacher, A.M. and Weyns, D. *Agents, Simulation and Applications*, Taylor and Francis, to appear 2008.
- Van Dyke Parunak, H. (1996) Applications of distributed artificial intelligence in industry, in *Foundations of Distributed Artificial Intelligence*, (eds G.M.P. O'Hare and N.R. Jennings), John Wiley and Sons, New York, NY, pp. 139–164.
- Van Dyke Parunak, H., Brueckner, S., Fleischer, M. and Odell, J. (2004) A design taxonomy of multi-agent interactions, in *Agent-Oriented Software Engineering IV 2003*, vol. 2935, pp. 123–137.
- von Neumann, J. and Morgenstern, O. (1953) *Theory of Games and Economic Behavior*, Princeton University Press, Princeton, NJ, 3rd edition.
- Wellman, M.P., Greenwald, A. and Stone, P. (2007) *Autonomous Bidding Agents – Strategies and Lessons from the Trading Agent Competition*, MIT Press.
- Weyns, D. and Holvoet, T. (2003) Synchronous versus asynchronous collaboration in situated multi-agent systems. *AAMAS '03: Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, New York, NY, USA, ACM, pp. 1156–1157.
- Weyns, D., Parunak, D.V.H., Michel, F., Holvoet, T. and Ferber, J. (2005) Environments for multi-agent systems, state-of-the-art and research challenges. *Environments for multi-agent systems*, vol. 3374 of *Lecture Notes in Computer Science*, pp. 1–48. Held with the 3th Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS, Springer-Verlag.
- Wieland, F. (1998) *Parallel simulation for aviation applications*. WSC '98: Proceedings of the 30th conference on Winter simulation, Los Alamitos, CA, USA, IEEE Computer Society Press, pp. 1191–1198.
- Wise, B.P., McDonald, M., Reuss, L.M. and Aronson, J. (2001) Task Order (TO) 69: ATM Human Behavior Modeling Approach Study. Technical report, National Air and Space Administration (NASA), Arlington, VA. Technical Advance in Air Transportation Concepts and Technologies (AATT).

- Wooldridge, M. (2002) *An Introduction to Multiagent Systems*, John Wiley and Sons Ltd., New York, NY.
- Wooldridge, M. and Jennings, N.R. (1995) Agent theories, architectures, and languages: a survey. *Intelligent Agents*, (eds Michael Wooldridge and Nicholas R. Jennings), Springer Berlin, pp. 1–22.
- Wooldridge, M. and Jennings, N.R. (1998) Pitfalls of agent-oriented development. *Proceedings of the 2nd International Conference on Autonomous Agents (Agents'98)*, (eds K.P. Sycara and M. Wooldridge), ACM Press, New York, pp. 385–391.
- Wooldridge, M.J. (2000) *Reasoning about Rational Agents*, The MIT Press, Cambridge, Massachusetts.
- Yilmaz, L. and Tolk, A. (2008) A unifying multi-model taxonomy and architecture for agent-based decision-support. *Intelligent Decision Making: An AI-Based Approach, Series: Studies in Computational Intelligence*, vol. 97 (eds G. Phillips-Wren, N. Ichalkaranje, and L. Jain), Springer.
- Yilmaz, L. and Tolk, A. (2006) Engineering ab initio dynamic interoperability and composability via agent-mediated introspective simulation. *Winter Simulation Conference*, pp. 1075–1182.
- Zeigler, B.P. (1986) Toward a simulation methodology for variable structure modeling. *Modeling and Simulation Methodology in the Artificial Intelligence Era*, (eds M.S. Elzas, T.I. Ören, and B.P. Zeigler), Amsterdam, North Holland.
- Zeigler, B.P. (1990) *Object-oriented simulation with hierarchical, modular models: intelligent agents and endomorphic systems*, Academic Press Professional, Inc., San Diego, CA, USA.
- Zeigler, B.P., Kim, T.G. and Praehofer, H. (2000) *Theory of Modeling and Simulation*, Academic Press, Inc., Orlando, FL, USA.

4**Agent-directed Simulation**

Levent Yilmaz and Tuncer I. Ören

4.1**Introduction**

The dynamic and distributed nature of simulation applications, the significance of exploratory analysis of complex phenomena, and the need for modeling the micro-level interactions, collaboration, and cooperation among real-world entities is bringing a shift in the way systems are being conceptualized (Yilmaz and Ören, 2005). Using intelligent agents in simulation models is based on the idea that it is possible to represent the behavior of active entities in the world in terms of the interactions of an assembly of agents with their own operational autonomy. The possibility to model complex situations whose overall structures emerge from interactions between individual entities and to cause structures on the macro level to emerge from models at the micro level is making agent paradigm a critical enabler in the modeling and simulation (M&S) of complex adaptive systems (Miller and Page, 2007).

Recent trends in technology as well as the use of simulation in exploring complex artificial and natural information processes (Denning, 2007; Luck et al., 2003) have made it clear that simulation model fidelity and complexity will continue to increase dramatically in the coming decades.

This chapter aims to provide a basic overview of the potential uses of agents for simulation, as well as the use of simulation technologies to study simulation for agents. Agent systems are defined as systems that are composed of a collection of goal-directed and autonomous physical, human, and logical software agents situated in an organizational context to cooperate via flexible and adaptive interaction and cognitive mechanisms to achieve objectives that cannot be achieved by an individual agent. Agent-directed simulation (ADS) is promoted as a unified and comprehensive framework that extends the narrow view of using agents simply as system or model specification metaphors. Rather, it is posited that ADS is comprehensive in the integration of agent and simulation technology. As shown in Figure 4.1, the model, simulator, and experimental framework (Zeigler et al., 2000) illustrate where agents can be used in M&S in supporting model conceptualiza-

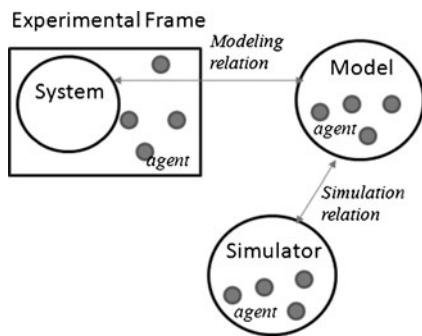


Fig. 4.1 Model, simulator, experimental framework.

tion, simulator design, and the realization of the context within the experimental framework.

Based on this characterization, agent-directed simulation consists of three distinct, yet related, areas that can be grouped under two categories as follows:

1. Simulation for agents involves the use of simulation modeling methodology and technologies to analyze, design, model, simulate, and test agent systems.
2. Agents for simulation: (1) agent-supported simulation deals with the use of agents as a backend and/or frontend support facility to enable computer assistance in simulation-based problem solving; (2) agent-based simulation, on the other hand, focuses on the use of agents for the generation of model behavior in a simulation study.

To outline the possibilities due to the synergy between agent and simulation technologies, we elaborate on each one of these dimensions. Specifically, in Section 4.2, an overview of agents and their role in system specification are provided. Section 4.3 focuses on categorizing the use of agents in simulation. The rest of the chapter builds on this categorization. First, in Section 4.4, the agent simulation dimension is introduced in detail by characterizing the elements of agent systems. A taxonomy of agent systems is presented to clarify the potential dimensions of analysis of agent systems. Elements of agent-based modeling such as cooperative knowledge processing, interaction, communication, cognition, and strategic action are delineated to substantiate the basic components of agent simulations when agents are used as the underlying specification formalism. Then, in Section 4.5, agent-based simulation is introduced to reveal the potential uses of agents in simulator design, as well as simulation integration. Agent-based simulation is promoted as a dimension in which agents are used to monitor, control, and coordinate model behavior generation. Section 4.6 elaborates on the use of agents as part of frontend and backend interface functions to facilitate realization of intelligent simulation infrastructures and model development environments. Finally, in Section 4.7, we summarize the proposed integrated ADS framework that expands horizons on the

synergy of agent and simulation technologies and conclude by suggesting potential avenues of further research.

4.2 Background

Software agents are entities that (1) are capable of acting in purely software and/or mixed hardware/software environments, (2) can communicate directly with other agents, (3) are driven by a set of goals, objectives, and tendencies, (4) possess skills to offer services, (4) perceive their environment, and (5) can generate autonomous behavior that tends toward satisfying its objectives (Ferber, 1999; Weiss, 1999).

4.2.1 Software Agents

Agents are autonomous software modules with the perception and social ability to perform goal-directed knowledge processing over time, on behalf of humans or other agents in software and physical environments. When agents operate in physical environments, they can be used in the implementation of intelligent machines and intelligent systems and can interact with their environment by sensors and effectors. The core knowledge processing abilities of agents include reasoning, motivation, planning, and decision making. The factors that may affect decision making of agents, such as personality, emotions, and cultural backgrounds, can also be embedded in agents. Additional abilities of agents are needed to increase their intelligence and trustworthiness. Abilities to make agents intelligent include anticipation (proactiveness), understanding, learning, and communication in natural and body language. Abilities to make agents trustworthy as well as assuring the sustainability of agent societies include being rational, responsible, and accountable. These lead to rationality, skillfulness, and morality (e.g., ethical agent, moral agent).

4.2.2 Complexity

Complexity is a pervading phenomenon in natural, social, business, artificial, engineered, or hybrid systems (Miller and Page, 2007). Cells, organisms, the ecosystem, markets, societies, governments, cities, regions, countries, large-scale software and hardware systems, the Internet – all are examples of complex systems (Parunak, 1999). Scientists use various forms of measurements and models to explore, understand, and elucidate the characteristics of such systems, while engineers build and design working artificial complex systems. As the engineered phenomena become more and more complex, we observe a convergence as engineers try to model the systems in an attempt to analyze them.

Complexity research mainly happens at the borders between various disciplines and thrives on interactions between engineering and the sciences. Predictive modeling comes from the context of theoretical science, with a bias toward deductive reasoning and a resulting preference for validity as a standard quality (Banks, 1993). The current trend in M&S treats the use of computer models as experimental science. In this new era, the purpose of M&S in dealing with complexity is not necessarily to predict the outcome of a system, rather it is to reveal and understand the complex and aggregate system behaviors that emerge from the interactions of the various individuals involved. This viewpoint is based on the observation that emergent engineering applications are becoming dynamic, adaptive, and open systems (Little, 2005), for which the tools of the traditional closed systems viewpoint are limited. More specifically, it is suggested in Little (2005) that if our critical infrastructures are to continue to provide vital services safely and reliably, the linkages between people, organizations, and technology need to be fully understood and managed holistically. As we start exploring the state space of such systems, the types of M&S applications espoused in this chapter will gradually increase and find their place within the engineering domain.

4.2.3

Complex Systems of Systems

The nature of the problems in systems engineering is increasingly becoming distributed. Supervision of telecommunication, transport, and power networks, as well as management subsystems, is distributed across the nodes of the network. Hence, a large number of operators and programs need to work collectively in a cooperative manner to control such systems to achieve desired objectives. Individual operators and programs often have a partial view of the overall system, and their actions should be efficiently coordinated, so that the overall system can react in a desired manner. The following observations regarding the nature of a modern and complex system of systems call for a mechanism to distribute activities and intelligence in terms of the interaction between relatively independent and autonomous agents.

- *Problems are widely distributed:* Decisions and control actions are not only taken by system entities, but also individuals (actors) carrying out functions at various levels. The allocation of roles to decision makers and the propagation of performance criteria to influence local objectives are critical in specifying the behavior of the system.
- *Systems are becoming goal-directed and adaptive:* In well-adapted systems goals and constraints are often implicit and embedded in the process. In a dynamic environment, an effective realization depends on self-organizing and adaptive mechanisms that are in place to change the properties of the process to meet current needs.
- *System processes evolve over time:* Processes are frequently modified to update the structure and mechanisms to keep some measure related to the relevant performance objective near an optimum. Control of adaptation, however,

is distributed across all components and subsystems. A useful and credible model for the analysis of the process and prediction of responses to changes in circumstances must reflect the mechanisms underlying the evolution of the process.

- *Sociotechnical systems are human-centered:* What people actually do, how they communicate and collaborate, how they solve problems, resolve conflicts, and learn behavior matters in the outcome of a process. Hence, representing activities requires modeling communication, collaboration, team work, conflict resolution, and tool and technology usage.
- *Software-intensive systems engineering is moving toward designs using concepts of autonomous and interacting entities:* The history of software-intensive systems development is moving toward the use of abstractions that suggest designs in terms of assemblies of entities, which are widely distributed and autonomous. Objects with autonomy and adaptiveness are presented as the basic elements of design in such systems analysis and design methodologies.

4.2.4

Software Agents within the Spectrum of Computational Paradigms

Computational paradigms are active research topics. The place of software agents within the spectrum of computational paradigms is given in Table 4.1. This spectrum covers procedural and intentional knowledge processing (Table 4.1), as well as goal-directed knowledge processing (Table 4.2) and corresponding algorithmic, declarative, interactive (event-based), AI-based, and agent-based computational paradigms. The focus in this spectrum is the shift of cognitive tasks from users to the system.

Procedural knowledge processing corresponds to an algorithmic programming paradigm that covers unstructured, structured, and object-oriented programming. In algorithmic programming, the analyst develops an algorithm, the programmer transforms the algorithm and generates a program, and the user activates the “whole” program. In this case, the system executes the compiled or interpreted program code. In algorithmic programming, the cognitive aspect of knowledge processing is performed by the analyst and the programmer; the system does not contribute to this aspect. Some simulation programs including continuous system simulation, activity-based simulation, and discrete-event system simulation are based on procedural knowledge processing and are algorithmic programming.

Intentional knowledge processing corresponds to a declarative programming paradigm. In this case, the user specifies the problem and the system (or a program generator transforms the specification into code, that is, generates a code). Then, the system executes the (compiled or interpreted) code. In M&S, from the very beginning the preferred computational paradigm has been declarative programming (except, of course, the activities of the specialists in Fortran, C, and the like). In declarative programming some of the workload (i. e., program generation

Table 4.1 Spectrum of computational paradigms (procedural and intentional knowledge processing).

Type of knowledge processing	Computational paradigm	Role of people	Role of computational system
Procedural	<i>Algorithmic</i> (unstructured, structured, object- orientated) <i>programming</i>	<ul style="list-style-type: none"> – Analyst <i>develops</i> an algorithm. – Programmer <i>transforms</i> the algorithm and generates a program. – User <i>activates</i> the “whole” program. 	<ul style="list-style-type: none"> – System compiles or interprets user’s program. – System executes the (compiled or interpreted) program code.
Intentional	Declarative	<ul style="list-style-type: none"> – User <i>specifies</i> the problem 	<ul style="list-style-type: none"> – System <i>generates</i> a code (program generator <i>transforms</i> the specification into code). – System <i>executes</i> the (compiled or interpreted) code.

from problem specification) is removed from the user; however, the system does not perform any cognitive ability.

Goal-directed knowledge processing has three corresponding computational paradigms, that is, interactive (event-based), AI-based, and agent-based computational paradigms. The *event-based computational paradigm* is the essence of interactive knowledge processing. Only the modules activated by the user are executed. The system assures the execution of the (already existing problem-independent) software modules corresponding to the selected functionalities. Still cognitive knowledge processing (in this case deciding what should be done and how to activate necessary modules) is performed by the user. The *AI-based computational paradigm* is the essence of heuristic, rule-based, and frame-based programming. The knowledge engineer specifies the rules and the user specifies the facts and the goal. A problem-independent inference engine is prepared only once. The system (inference engine) determines the order in which the rules have to be executed and executes the rules until the goal is satisfied; hence the system takes some of the cognitive load from the user. The *agent-based paradigm* takes over the cognitive knowledge processing load from the user and leaves him/her the task of specifying the goals at a high level and has the agent(s) finding a solution to the problem.

Table 4.2 Spectrum of computational paradigms (goal-directed knowledge processing).

Type of knowledge processing	Computational paradigm	Role of people	Role of computational system
Goal-directed	Interactive (Event-based)	– User activates functions to be performed. (Indirectly activates the module(s) of code to be executed.)	– System assures the execution of the (already existing problem-independent) software modules corresponding to the selected functionalities.
	AI-based (heuristic, rule-based, or frame-based computation)	– Knowledge engineer specifies the rules. – User specifies the goal and the facts (initial conditions). – (Problem independent) inference engine is prepared only once.	– System (inference engine) determines the order in which the rules have to be executed and executes the rules until the goal is satisfied.
	Agent-based	– User specifies the goal and delegates finding a solution of the problem to agent(s).	Agents – process the goal (identifies subgoals, sequence them, and determine additional knowledge requirements). – perceive their environment; – perform goal-directed knowledge processing; – decide which (other agent or nonagent) software modules to activate or subdelegate the task(s).

Agents process the goal (to identify subgoals, to sequence them, and to determine additional knowledge processing requirements), perceive their environment, perform goal-directed knowledge processing, and decide which other (agent or nonagent) software modules to activate to solve a problem or to subdelegate task(s). The agent-based computational paradigm has the advantages of the AI-based paradigm (rule-based, fuzzy-rule-based) and takes interactivity to a next level. In the event-based interactive paradigm, the user can choose a software module to activate; in the agent-based paradigm, an agent can deliberate and choose to activate another agent or a nonagent software module.

4.3

Categorizing the Use of Agents in Simulation

Agents are often viewed as design metaphors in the development of models for simulation and gaming. Yet this narrow view limits the potential of agents in improving various other dimensions of simulation. To this end, Figure 4.2 presents a unified paradigm of ADS that consists of two categories as follows: (1) simulation for agents (agent simulation), that is, simulation of systems that can be modeled by agents (in engineering, human and social dynamics, military applications, etc.) and (2) agents for simulation that can be grouped under two groups: *agent-supported simulation* and *agent-based simulation*.

4.3.1

Agent Simulation

Agent simulation involves the use of simulation conceptual frameworks (e.g., discrete-event, activity scanning) and technologies to simulate the behavioral dynamics of agent systems by specifying and implementing the behavior of autonomous agents that function in parallel to achieve objectives via goal-directed behavior. In agent-based model specifications, agents possess high-level interaction mechanisms independent of the problem being solved. Communication protocols and mechanisms for interaction via task allocation, coordination of actions, and conflict resolution at varying levels of sophistication are primary elements of agent simulations. Simulating agent systems require understanding the basic principles, organizational mechanisms, and technologies underlying such systems. The principal aspects underlying such systems include the issues of action, cognitive aspects in decision making (Wooldridge, 2002), interaction, and adaptation. Organizational mechanisms for agent systems include means for interaction. That is, communication, collaboration, and coordination of tasks within an agent system require flexible protocols to facilitate the realization of cooperative or competitive

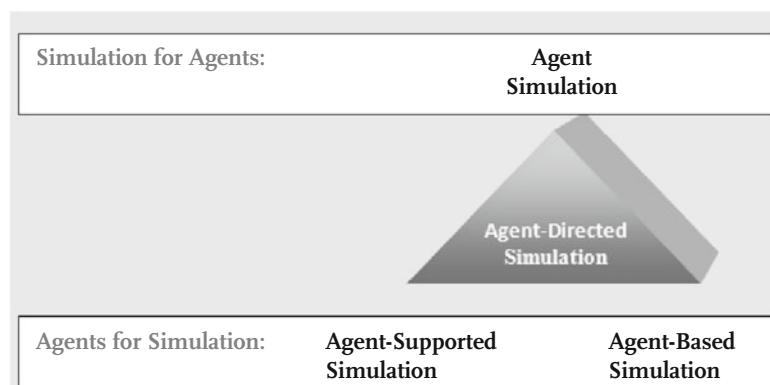


Fig. 4.2 A unified paradigm – agent-directed simulation.

behavior in agent societies (Weiss, 1999). Agent-based modeling, in which agents are used as design metaphors to conceptualize and specify agent systems, is becoming the most common methodology in agent simulation. Yet agent system simulation can be conducted with or without agents. Given a specification of an agent system via formalisms such as DEVS or an agent-based model, an appropriate simulator (e.g., DEVS simulator) can be used to generate model behavior. On the other hand, simulation(s) can be used to realize the deliberation architecture of agents in an agent system by simulating the cognitive processes in decision making. While embedding simulations within the deliberation architecture of agents is not necessarily a novel concept on its own, it is still an unexplored territory in multiagent system design and implementation.

4.3.2

Agent-Based Simulation

Agent-based simulation is the use of agent technology to monitor and generate model behavior. This is similar to the use of AI techniques for the generation of model behavior (e.g., qualitative simulation and knowledge-based simulation). The development of novel and advanced simulation methodologies such as *multisimulation* (Yilmaz and Phillips, 2007) suggests the use of intelligent agents as simulator coordinators, where runtime decisions for model staging and updating take place to facilitate dynamic composability. The perception feature of agents makes them pertinent for monitoring tasks. Also, agent-based simulation is useful for having complex experiments and deliberative knowledge processing such as planning, deciding, and reasoning. Agents are also promoted and demonstrated as critical enablers to improve composability and interoperability of simulation models (Yilmaz and Paspuletti, 2005) and software in general (Genesereth and Singh, 1993).

4.3.3

Agent-Supported Simulation

Agent-supported simulation deals with the use of agents as a support facility to augment simulations and enable computer assistance by enhancing cognitive capabilities in problem specification and solving. Hence, agent-supported simulation involves the use of intelligent agents to improve simulation and gaming infrastructures or environments. Agent-supported simulation is used for the following purposes:

1. to provide computer assistance for frontend and/or backend interface functions;
2. to process elements of a simulation study symbolically (for example, for consistency checks and built-in reliability); and
3. to provide cognitive abilities to the elements of a simulation study, such as learning or understanding abilities.

For instance, in simulations with defense applications, agents are used as support facilities to

1. fuse, integrate, and deconflict the information presented by the decision maker;
2. generate alarms based on the recognition of specific patterns;
3. filter, sort, track, and prioritize the disseminated information; and
4. generate contingency plans and courses of action.

4.4

Agent Simulation

M&S of agent systems involves the design and realization of agent organizations to solve problems, enable training, or perform computational experiments involving phenomena that are distributed and autonomous in nature. Agent simulation requires engineering various control, coordination, and communication mechanisms to govern entities to mimic solving distributed problems or distributed solving of problems by applying AI-oriented techniques. The control mechanisms for the desired behavior are engineered through allocation of tasks for collaboration, coordination of tasks, and resolution of conflicts within the organization. Various agent models and architectures (Wooldridge, 2002) have been proposed to emulate the cognitive, reactive, and proactive behavior that distinguishes agents from objects and actors (Agha and Hewitt, 1985). Agent-based techniques constitute the fundamental building blocks of such models, specifically, agent communication, interaction protocols such as coordination, cooperation, and negotiation, distributed planning, and learning. Agent simulation can be performed with or without agents. If agents are used in modeling the system or phenomena of interest, the purpose of the study is often based on the insight to be gained and exploring how local interactions between heterogeneous and autonomous agents generate regularities and patterns of interest. Generation of regularity and macro-level behavior in terms of agents requires identification and specification of micro-level rules of behavior that are sufficient to generate the patterns of interest.

4.4.1

A Metamodel for Agent System Models

The term *agent systems* is applied to systems that are comprised at least of the following structural elements:

- An environment (E) that encapsulates the agents, objects, and resources;
- A set of objects (O) that are perceived and acted upon by agents;
- A set of agents (A) that represent active entities.;
- A set of resources (R) used by the agents to perform tasks to achieve their objectives;

- A distributed knowledge base that represents for each agent the mental model that characterizes beliefs about the environment, the self, and other agents in the system;
- A set of tasks defined in terms of actions of agents. These actions operate on the objects with the goal of changing the state of the environment in accordance with the objectives of the agent;
- A set of operations that enable agents to perceive, produce, transform, and manipulate objects in the environment.

Figure 4.3 depicts a metamodel for the specification of the structure of agent system models. Modeling agent systems for simulation requires explicit specification of agents, objects, operations, beliefs of agents over the environment and agents, and resources. Domain-specific tasks involve a sequence of actions performed by agents to change the state of the environment to bring about the desired outcomes. The tasks and operations need to be coordinated to achieve the results in an effective and efficient manner. Hence, interaction and communication mechanisms need to be devised to control the operation of the agent society.

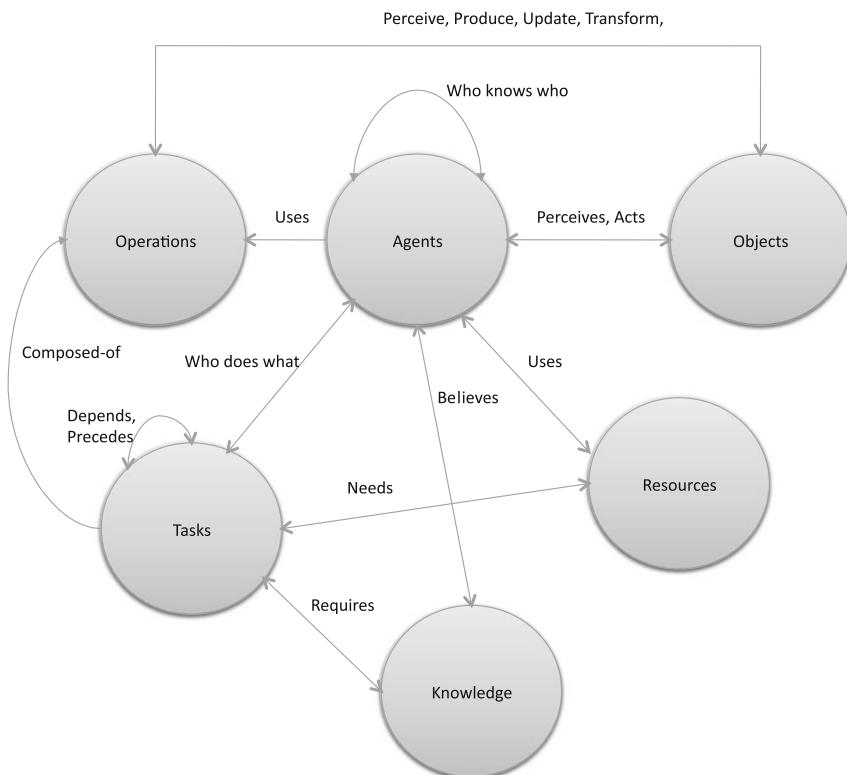


Fig. 4.3 A conceptual metamodel for agent system modeling.

4.4.2

A Taxonomy for Modeling Agent System Models

Modeling agent systems requires understanding the design space and factors affecting the choice and assembly of the building blocks defined above. Agent systems are situated within an environment that lies outside of the boundaries of the system. The major influence of the environment on the behavior of an agent system is predicated on two main factors: complexity and rate of change. Complexity refers to the number and diversity of elements in the environment. Rate of change refers to how rapidly these elements change. Together, these two factors help measure the degree of uncertainty of the environment.

As depicted in Figure 4.4, agent systems can be classified as stable, unstable, and volatile based on the degree of change and complexity. Models of agent systems without explicit strategies that form the basis of goals may fail to link the environment and its assumptions/expectations to the system. Common strategies in agent systems, depending on the application domain, involve various dimensions such as risk reduction, adaptability, innovation, high performance, and security. Deciding on a strategy early on in the design of an agent system model is critical, as high performance depends on how well the rest of the agent system

Based on	Additional Criteria						Type of Agent system			
<i>Structure and Dynamics of the Environment</i>	<i>Rate of change</i>	low				Stable				
		high	<i>Complexity (number and diversity of elements) is</i>			low	Unstable			
<i>Problem Solving Strategy</i>	<i>Degree of task uncertainty</i>	low				Coherent				
		high					Volatile			
<i>Organizational Structure</i>	<i>Complexity of the structure is</i>	low		<i>Formalization of the interaction and actions is</i>	<i>Centralization (concentration of authority) is</i>	low	Organic			
		high	high			high	Bureaucratic			
<i>Decision Making Strategy</i>	<i>Agreements /Disagreements on the goal</i>	agree								
		disagree	<i>Agreements /Disagreements on the method (means)</i>		agree					
<i>Agent architecture</i>	<i>Agent is driven-by</i>	disagree								
		goal (teleonomic)	goal processing is	situuated			Garbage can			
<i>Control and Coordination Mechanism</i>	<i>Mechanism is characterized by</i>	<i>cognition (symbolic)</i>								
		mutual agreement								
		market								
		rules								

Fig. 4.4 A taxonomy for agent system design space.

structure fits the strategy. Strategies are decomposed into goals, which are then realized.

The complexity, centralization, and formalization dimensions play a critical role in the design of agent systems. Structural complexity can be viewed in terms of vertical complexity, which is defined by the levels of hierarchy, and horizontal complexity that emerges as a result of the differentiation among a collection of agents that depict distinct units at the same level of resolution. The larger the complexity, the higher is the need for communication. Centralization refers to the location of decision making. At a high level of centralization, the decisions are transferred to managers, while a low level of centralization facilitates decision making at the edge of the organizations, hence increasing the level of autonomy. Formalization is a measure of the extent to which rules and regulations govern the interaction and activities of agents. Given these three dimensions, agent systems at a high level can be classified as mechanistic, organic, and bureaucratic. Mechanistic forms of agent systems are characterized by high levels of complexity, formalization, and centralization. Organic forms of agent systems are characterized as the opposite of mechanistic forms. That is, they are simple, informal, and decentralized. Compared to objects, agents are viewed as autonomous entities that are able to reason, generate and evaluate alternative actions to select one that is consistent with their objectives.

Common decision making strategies in agent systems can be characterized in terms of the agreement/disagreement over goals and methods. If agents agree on the goal, they can utilize (bounded) rational models to generate a course of actions, as the ambiguity and uncertainty are at a minimum. However, if there is disagreement over the goals due to conflicts among goals, competition over scarce resources, agents can either negotiate to establish a coalition or use a garbage-can model, depending on the level of agreement on the means to achieve a consensus.

The choice of agent architecture in agent systems is a major factor in determining the level of intelligence and autonomy that can be exhibited in reaction to dynamic, unpredictable environments. Depending on what drives agents to act, one can classify agents as goal-driven or perception-driven. Perception-driven agents are reactive and use simple mechanisms to monitor the state of the environment to select proper rules to react and act on the objects of the environment. Goal-driven agents can further be classified on the basis of the nature of goal processing. Cognitive agents have a symbolic and explicit representation of the environment on which they can deliberate to select intentions and choose plans to achieve the goal associated with the selected intention. Alternatively, interpretive agents have motivation mechanisms that push them toward accomplishing a task or achieving a goal provided by the designer. Interpretive agents have a representation situated at a subsymbolic level integrated into their sensory-effector mechanisms.

4.4.3

Using Agents as Model Design Metaphors: Agent-Based Modeling

Agent-based modeling is predicated on a model specification perspective that adopts the agent paradigm for representing the behavior of systems. Such sys-

tems are often viewed as complex and adaptive, and the observed behavior is often characterized as emergent (Miller and Page, 2007), while the underlying process is viewed as generative (Epstein, 2006). The premise of the agent-based paradigm is based on the insight to be gained and exploring how local interactions between heterogeneous and autonomous agents generate regularities and patterns of interest. As such, agent-based simulations are advocated as powerful tools for computational experimentation to analyze complex systems.

4.4.3.1 Characteristics of Agent-Based Models

The generation of regularity and macro-level behavior in terms of agents requires the identification and specification of micro-level rules of behavior that are sufficient to generate the patterns of interest. The following characteristics of agent-based simulation help distinguish the paradigm from conventional simulation approaches.

- *Heterogeneity* dictates that individual units of abstraction (i. e., agents) differ from one another in terms of the properties and behavior they exhibit. Preferences, goals, options, decision making styles, social networks, locations, and culture dynamically change and adapt endogenously over time.
- *Autonomy* of agents dictates a paradigm, where there is no central and top-down control over agent behavior. That is, no central coordination protocols, controllers, or higher-level authority is posited ab initio. Note, however, that there is the possibility of feedback between emergent macro-level behavior and micro-level rules of interaction, and the agents that are dynamically created at runtime are expected to conform and be conditioned to the norms of the agent organization they are embedded.
- *Explicit environmental* representation provides a landscape in which agents interact and move. Coordination via space provided by the environment is a common strategy in bottom-up coordination among agents.
- *Local interactions* among agents take place over the space designated by the environment. Interactions take place through messages between agents or signals that are disseminated over the landscape. The propensity of signals and the neighbor relation in social networks are common enablers for local interaction among agents.
- *Bounded rationality* dictates that agents do not have global knowledge or infinite computational capacity. While agents are situated, goal-driven, and purposeful, they are not global optimizers due to lack of information. Rather, agents make decisions and select actions based on local information.

4.4.3.2 Elements of Agent-Based Models

Given the focus on interacting systems of agents, there are a number of core agent-based modeling elements that bring coherence and conceptual framework for developing simulations that are built on the premise of agent paradigm.

Input and Knowledge Processing In a typical computation, it is taken for granted that a computational unit (a software module, including a software agent) will have inputs and outputs. Inputs to a computational unit can be generated outside of the unit (exogenous input) or inside the unit (endogenous input) (Ören and Yilmaz, 2004). As seen in Figure 4.5, externally generated inputs are of two types: passively accepted (or imposed or forced inputs) and actively perceived inputs (or perceived inputs). Conventional inputs to computing units are provided by coupling, several types of argument passing, knowledge in a common area (including the use of noticeboards in artificial intelligence), message passing, and broadcasting. All of them are “forced inputs” to the computing unit, that is, the unit does not need to monitor or detect them. They are readily available. Passive inputs can be data, facts, forced events, sensation, and external goals imposed on a unit. Actively perceived exogenous inputs consist of perceptions (interpreted sensory data and selected events), perceived goals, and inputs evaluated with respect to acceptability, as well as inputs whose sources are evaluated with respect to reliability and credibility. Endogenous inputs are shown in Figure 4.6. An intelligent system can monitor itself and make deductions based on the observed internal facts or events or based on a lack of them. The result of introspection can then become internally generated (endogenous) perceived input. Endogenous input can be generated by anticipation of facts or events, as well as deliberation of past facts and/or events. Endogenous inputs can be internally generated questions, hypotheses, or goals.

Goal-directed behavior The goal processing in an agent-based model focuses on the intentions of agents. Agents with cognitive reasoning capabilities have a set of well-defined explicit goals that influence their actions. There are other models in which agent goals are implicit and rely on sustaining and attaining a specific level

Mode of Input	Type of Input
Passive acceptance of exogenous input (imposed or forced input)	<p>Types of access to input: coupling, argument passing, knowledge in a common area, message passing, broadcasting.</p> <p>Nature of input:</p> <ul style="list-style-type: none"> - Data, facts <ul style="list-style-type: none"> - Forced events - Sensation (converted sensory data: from analog to digital single or multisensor – sensor fusion) - External goals (imposed goals)
Active perception of exogenous input (perceived input)	<ul style="list-style-type: none"> - Perception (interpreted sensory data and selected events) (possibly anticipated) <ul style="list-style-type: none"> – includes: decoding, selection (filtering), recognition, regulation - Perceived goals - Evaluated inputs <ul style="list-style-type: none"> – evaluation of inputs (acceptability) – evaluation of source(s) of inputs (reliability, credibility)

Fig. 4.5 Types of exogenous inputs Ören and Yilmaz (2004).

Mode of Input	Type of Input
Active perception of endogenous input	- Introspection (perceived internal facts, events; or realization of lack of them)
Generation of endogenous input	- Anticipated facts and/or events (anticipatory systems) - Deliberation of past facts and/or events (deliberative systems)
Types of endogenous input	- Internally generated questions - Internally generated hypotheses by: -- Expectation-driven reasoning (Forward reasoning, or (Bottom-up reasoning, or (Data-driven reasoning) -- Model-driven reasoning - Internal goals (internally generated goals)

Fig. 4.6 Types of endogenous inputs Ören and Yilmaz (2004).

of resource to survive in the environment in which it is embedded. By manipulating agent goals, one can induce forces on agents' behavior, as well as a model's behavior.

Interaction Interaction requires exchanging information among agents. Information is exchanged in the form of messages. Designing a communication medium in agent-based models requires deciding the mode of communication and routing mechanisms. Agents can directly send messages to each other point-to-point or use indirect mechanisms such as broadcast or multicast. Recipients subscribe to specific types of events and react to published messages that relate to events of interest. Agents can also communicate via their environment by using *noticeboards* or *marks* that change the state of the environment. Such changes further influence the behavior of agents, which observe changes in the state of the environment. As such, communication through the environment is an indirect mode of communication. While messages and markers are common abstractions used as communication mediums, it is also possible to use signal propagation to selectively convey information based on spatial relations. Signal intensity often degrades as it propagates away from the resource, and as such, agents closer to a resource are more likely to react to events of interest.

Strategic action Agents receive messages, perceive inputs from the environment or other agents, and generate actions to influence agents and/or the environment itself. The interactions between agents are affected by the spatial configuration of the environment they are embedded in. Agents in a social network can interact with their evolving set of neighbors. Agents arrayed in a circular pattern interact only with agents they are connected with around the periphery. Whatever the form of the space, the agent interactions are mediated by the configuration that constrains the information flow and action. Agent activation can be controlled by a protocol that coordinates and synchronizes the activities according to normative mechanisms. In the agent-based paradigm, agents can also be activated asynchronously.

Under asynchronous activation, each agent decides when to activate, what information to process that is available at that time, and alter the information sphere that will affect the other agents when they are activated. The choice of action for an agent may follow a well-defined strategy that is represented in terms of rules that change the state of the environment in such a way that the resulting outcomes provide benefits or impose costs on the individual agents. By letting agents choose rules and reproduce in a way that improves their payoffs, one can induce adaptive behavior.

Cognitive and deliberative decision making The extent to which agents deliberate to select actions vary depends on the context in which agents are deployed. There are models, like the Game of Life, where agents are not capable of deliberation, and the scientific exploration is based on gaining insight about how simple rules result in macro-level properties. Yet there exist other models, such as Double Action Tournament and the annual Trading Agent Competition, where agents exhibit humanlike behavior to explore how regularities emerge as a result of interaction between individual complexities. In such models, social agents, unlike physical agents, are complex entities that can exhibit personality (Ghassem-Aghaee and Ören, 2003), emotional intelligence (Bates, 1994), and motivational behavior to form intentions. The design of such agents involves the question of how goals and intentions emerge and how they eventually lead to the execution of actions that change the state of the environment. Motivation theory (Muller, 1996) provides a sound basis for developing internal agent models.

4.4.4

Simulation of Agent Systems

Simulation of agent systems involves development agent simulators and associated control mechanisms that represent coordination (Ferber, 1999), collaboration via task allocation (Smith, 1980), conflict resolution (Kakehi and Tokoro, 1993), learning (Sen and Weiss, 1999), and planning (Weiss, 1999). At the micro level agents are represented by a variety of architectural representations spanning from reactive to cognitive (Sun, 2006) and proactive models. Agent simulators interpret the model of the agent to generate its behavior.

4.4.4.1 Agent Simulators

Various models of agents have been proposed to capture a wide spectrum of agency. While rule-based agents with simple stimulus-action pairs are common, the agent community has also developed agent architectures with varying levels of cognitive capabilities. The design of such agents is mainly influenced by control theory and cognitive psychology, respectively (Muller, 1996). Control theory investigates the agent-world relationship from the situated and reactive perspective of machine-oriented feedback control mechanisms. On the other hand, the issue of how goals and intentions of a human agent emerge and how they lead to the selection of actions that change the state of the environment is the subject of cognitive psy-

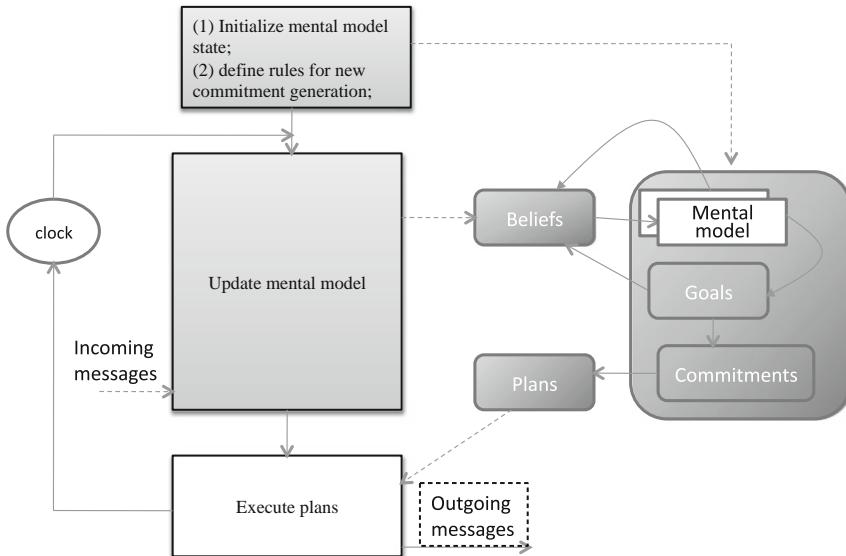


Fig. 4.7 A generic agent simulator (Shoham, 1993).

chology. Agent model interpreters simulate and generate the behavior specified by agent specifications. Shoham (1993), for instance, proposed an agent simulator in which the control of an agent is achieved by a generic agent interpreter running in a two-phase loop. In each cycle, the agent reads current messages and updates its mental state; this is followed by executing its commitments, which may result in revision of its beliefs. Figure 4.7 presents an extended version of Shoham's (Shoham, 1993) AGENT-0 interpreter.

The agent is constructed with an initial state, which is updated based on the incoming messages that enable the agent to perceive the state of the environment. Updating the mental model involves revision of the beliefs that depict the current set of facts about the world. Fuzzy logic can be used to represent the strength of the agent's beliefs. Beliefs direct selection of a mental model appropriate for the perceived situation. The mental model then facilitates the selection of desired goals, which are then used as input to the process of matching intentions to desires. Selected intentions or commitments provide a basis for the identification of proper plans that are comprised of one or more actions. These plans are then executed by the agent. As a result of the execution of the actions, the agent generates outgoing messages, which are received by other agents or the environment.

4.4.4.2 Simulating Task Allocation and Coordination

Agent systems involve multiple agents that combine their skills to perform collective work. Simulation of such systems requires mechanisms to identify capacities, skills, and resources of agents to decide who has to do what and using what resources, based on both the availability and quantity of resources, as well as con-

straints imposed by the environment. Given the complexity of tasks, it is critical to decompose tasks into subtasks and allocate them to agents with skills and resources that are capable of handling the task. Common strategies and modes in task allocation are either centralized or distributed. Centralized task allocation is often deployed in hierarchical subordination structures or flexible organizations in which broker or mediator agents undertake the role of task allocation. In distributed allocation, there are at least two options: (1) an acquaintance network can be used in cases where agents have representation of other agents and their capabilities or (2) a contract-net protocol (Smith, 1980) can be used to request bids from agents that are capable of proposing services that fulfill the request. The coordination of actions of agents involves a set of supplementary activities that facilitate the synchronization of multiple agents in accomplishing a complex task that cannot be accomplished by a single agent. The main reasons for coordination in agent system simulations are that (1) agents need information from other agents and/or the environment to complete/achieve their goals, (2) agents have limited resources, and (3) coordination among agents avoids redundancy in actions.

4.4.4.3 Simulation of Learning in Agent Systems

Most realistic agent systems adapt to changes in their environment to improve their performance. Consider, for instance, the simulation of the software process of an organization at level 4 of the Capability Maturity Model. By monitoring the outcomes and performance of the process, such organizations are required to improve their processes to increase production efficiency and effectiveness, as well as the quality of the artifacts produced during the process. The agent community has developed various learning tools that can be used to adapt agents to emerging unforeseen conditions. Sen and Weiss (1999) overview various learning methods such as the credit assignment approach (Holland, 1995), by which outperforming stimulus-action rules of agents are assigned credits to improve their chances for selection in future states. Other learning strategies such as reinforcement learning (Sen and Weiss, 1999) and evolving automata (Miller and Page, 2007) are common in simulating learning in agents and organizations.

4.5

Agent-Based Simulation

The use of agents in simulation is not restricted to conceptualizing and modeling scenarios of interest in a system or phenomena under study. Agent-based simulation pertains to the use of agents in the generation of model behavior. As such, the leverage of agents and intelligent agent technology in developing next-generation simulator designs and simulation infrastructure implementations constitutes the basis for agent-based simulation.

4.5.1

Autonomic Introspective Simulation

The Symbiotic Adaptive Simulation System (SAMS) presented in Chapter 2 is an obvious example (Figure 4.8) that demonstrates alternative uses of agents in a single framework. Symbiotic Simulation (S2) involves the use of simulation systems that are synchronized with the physical systems to enable mutually beneficial adaptation. In S2, simulation outputs are examined by agents and used to determine how the physical system may be optimized. Similarly, measurements from the physical system are used to validate the simulation. When uncertainty in the physical system is present, multiple what-if simulation experiments can be helpful in adjusting the physical system. However, since the number of what-if experiments that may be performed is limited by both computational and real-time constraints, the ability to conduct an efficient search of the model space is essential.

SAMS is intended as a self-organizing S2 technique appropriate for physical systems characterized by distributed, dynamic, and uncertain conditions. The salient feature of SAMS is the use of evolutionary computation in terms of a genetic algo-

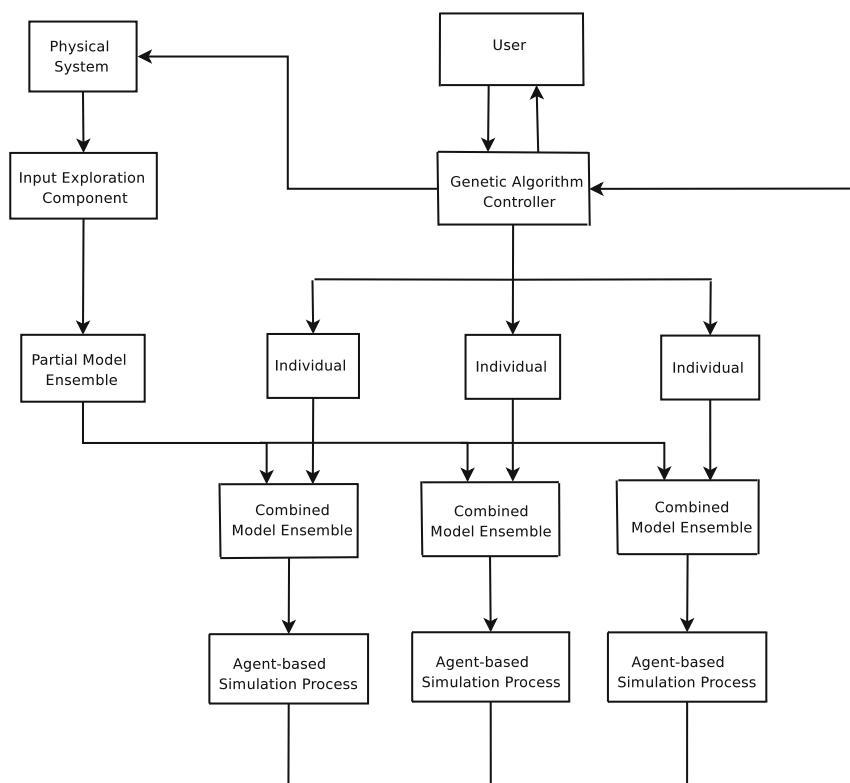


Fig. 4.8 SAMS simulator architecture.

rithm (GA) to evolve the model ensemble in response to changes in the physical system. With this feature, it is conjectured that an effective search of an uncertain model space would be possible, thus permitting synchronization with the physical system. Observations from the physical system are passed to agents within the input exploration component (IEC) responsible for conducting *agent-supported simulation* via input analysis and selecting appropriate distributions for the uncontrollable model factors. Samples from these distributions are then used to create a partial model ensemble (PME), a copy of which is integrated with each individual to form one combined model ensemble (CME) for each agent simulation process (ASP).

The genetic algorithm controller (GAC) is embedded within the simulator to facilitate *agent-based simulation* and is responsible for evolving the population of individuals that are used to form the CMEs and a number of agent simulation instances (ASIs) is mapped to one or more CPU cores. If the population used by the GAC is large, or if hardware resources are limited, multiple ASPs can run on a single core. If there is an excess of hardware, a SAMS simulator is capable of uploading models within the CME to multiple CPU cores. Outputs from the ASPs take the form of objective fitness values of the individuals that have been averaged across all of the replications specified by the CME. These are passed to the GAC, which then assigns the fitness values to the simulated individuals before continuing execution of the GA. Ideally, it should be possible for a user to interact with the GAC in real time to examine the individuals generated and possibly seed new configurations to the population.

4.5.2

Agent-Coordinated Simulator for Exploratory Multisimulation

We define multisimulation as a simulation of several aspects of reality in a study. It includes simulation with multimodels, simulation with multiaspect models, and simulation with multistage models. Simulation with multimodels allows computational experimentation with several aspects of reality; however, each aspect and the transition from one aspect to another one are considered separately. (In special cases, multimodels can be metamorphic models or evolutionary models). Simulation with multiaspect models (or multiaspect simulation) allows computational experimentation with more than one aspect of reality simultaneously. This type of multisimulation is a novel way to perceive and experiment with several aspects of reality as well as exploring conditions affecting transitions. While exploring the transitions, one can also analyze the effects of encouraging and hindering transition conditions. Simulation with multistage models allows branching of a simulation study into several simulation studies, each branch allowing one to experiment with a new model under similar or novel scenarios. Each different strategy component characterizes a distinct aspect. Multisimulation can be used to branch out multiple simulations, where each simulation uses a specific component configured with an exclusively selected strategy component. Similarly, multiple distinct stages of the problem can be qualified at a given point in time during the simulation by

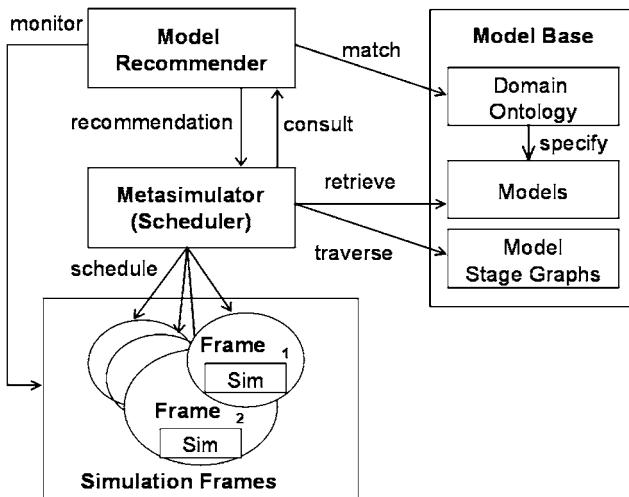


Fig. 4.9 Components of a multisimulation engine.

virtue of the evaluation of an updating constraint. In such a case, multisimulation enables branching multiple distinct simulations, each one of which generates the behavior of distinct plausible stage within the problem domain.

Multisimulation with multimodels, multisaspect models, or multistage models needs mechanisms to decide when and under what conditions to replace existing models with a successor or alternative. Staging considers branching to other simulation studies in response to a scenario or a phase change during experimentation. Graphs of model families facilitate derivation of a feasible sequence of models that can be invoked or staged. More specifically, a graph of model families is used to specify alternative staging decisions. Each node in the graph depicts a model, whereas edges denote a transition or switching from one model to another. Figure 4.9 depicts the components of the abstract architecture of a possible multisimulation engine.

A metasimulator is an agent that generates a staged composition of models by traversing the model stage graph and coordinates their simulation and staging within distinct simulation frames. Each frame simulates a distinct subset of models derived from the model stage graph. Note, however, that not all staged compositions are feasible or useful. Hence, the metasimulator needs to consult with the model recommender before model staging to determine if emergent trigger or transition condition in the simulation is consistent with the precondition of the model to be staged. More than one model in a family can qualify for staging; in such cases separate simulation frames need to be instantiated to accommodate and explore plausible scenarios. Given a collection of models (or, more generally, a family of models), a stage graph can be generated automatically by an optimistic approach that connects every available node (model) to every other node within the problem domain. The edges in a model stage graph denote plausible transitions between

models as the problem shifts from one stage to another. One can consider each model as a separate conflict management protocol (i.e., compromise over actions, compromise over outcomes, negotiation, and mediation) or a phase in the conflict process (i.e., escalation, resolution), where a phase (i.e., resolution) can constitute alternative models (i.e., mediation, negotiation, third-party intervention).

The subsets of staged models can be identified by traversing and enumerating the graph in some order (i.e., depth-first). Infeasible paths may be due to an unreachable node, or they may result from conflicts between the transition condition and precondition of the target model. Infeasible paths due to incompatible sequences of models are common. Each edge (say from n_i to n_j) indicates that there is some legitimate solution that includes n_i followed by n_j ; yet, it does not imply that every solution containing n_i followed by n_j is legitimate. As argued above, each model in a family of models is associated with a precondition. A precondition denotes the conditions required for a model to be instantiated. Hence, the feasibility of staging a successor model depends on the satisfiability of its precondition (relevance) by the condition of the transition and the postcondition of the predecessor model. As a result, not all enumerated staged sequences of model components are feasible.

Model recommendation in multisimulation can simply be considered as the exploration of the model staging space that can be computed by a reachability analysis of a graph. There are two modes for the usage: (1) offline enumeration of paths using the graph and performing a staged simulation of each model in sequence one after the other, unless a model staging operation becomes infeasible due to conflict between the transition condition and the precondition of the successor model, and (2) runtime generation of potential feasible paths as the simulation unfolds. In both cases, an online model recommender plays a key role in qualifying a successor model. The first case requires derivation of a sequence of models using a traversal algorithm. The edges relate families of models. Therefore, the actual concrete models, the preconditions of which satisfy the transition condition, need to be qualified since transition to some of these model components may be infeasible due to conflict between a candidate model and the inferred situation. Identifying such infeasible sequences is computationally intractable; otherwise, it would have been possible to determine if the conjunction of two predicates is a tautology by using a polynomial time algorithm. Experience in the component-based simulation paradigm, however, indicates that for most model components preconditions are simple. Hence, it is possible to eliminate some models that violate the transition condition. For the remaining possible transitions it is possible to select one of three strategies: (1) omit all difficult qualification conditions, (2) decide on an edge-by-edge basis which specific models of a model family to include, or (3) include all difficult edges. Omitting all difficult associations between transitions and model preconditions is conservative. This strategy excludes all infeasible models. The cost is the exclusion of some feasible edges. Hand selection of those associations between transition conditions and models facilitates inclusion of feasible models. Nonetheless, the costs involved with this level of accuracy are the potential human error and the effort needed to filter out infeasible models. Choosing to include all

difficult associations is liberal, in that it ensures inclusion of all feasible models. The cost is the inclusion of some infeasible models, hence the inclusion of some undesirable staged compositions that enforce models to be simulated even when their qualification conditions are violated. Nevertheless, it is possible to screen out such models using an online model recommender.

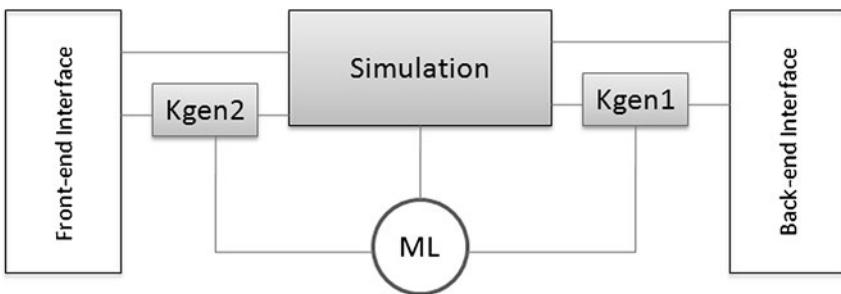
The second, more ambitious, yet flexible, approach is to delay the enumeration process until a model is qualified at runtime. Runtime generation of feasible staging using the graph of model families requires monitoring and evaluation of transition conditions as the simulation unfolds. An agent planning layer connected to the simulator would be capable of identifying, qualifying, and, if necessary, selecting and instantiating a model based on the specified preferences and options. Furthermore, in the case of an impasse or lack of knowledge about preferences among qualifying model switch strategies, a planning layer can guide exploration of alternative contexts (games) in some order. The scheduler agent follows the recommendations made by the planner agent to instantiate distinct simulation frames.

Candidate models and associated simulations are maintained by focus points. A focus point manages branch points in the simulation frame stack. Suppose that a goal instance (i.e., stage transition condition) is at the top of the stack. If only a single model qualifies for exploration, then it is pushed onto the stack. Yet if more than one model matches the condition, a simulation focus point is generated to manage newly created simulation branching (discontinuity) points. Each one of these simulation focus points has its own context. When a path is exhausted, the closest focus point selects the next available model to instantiate the simulation frame or return to the context that generated the focus point. As simulation games are explored, a network of focus points is generated. Determining which focus point should be active at any given time is the responsibility of the metascheduler. When more than one model is qualified, then the scheduler needs to decide which one to instantiate. Control rules can inform its decision. Three steps are involved in deploying a new simulation frame in such cases: matching, activation, and preference. The matching step should both syntactically and semantically satisfy the request. The activation step involves running a dynamic set of rules that further test the applicability of models with respect to contextual constraints. Finally, the preference step involves running a different set of rules to impose an activation ordering among the active frames.

4.6

Agent-Supported Simulation

Agent-supported simulation involves the use of intelligent agents and associated technologies (Luck et al., 2003) to improve simulation and gaming infrastructures or environments (1) to provide computer assistance for frontend and/or backend interface functions, (2) to process elements of a simulation study symbolically (for example, for consistency checks and built-in reliability), and (3) to provide cognitive abilities to the elements of a simulation study, such as learning or understand-



Kgen: Knowledge Generator

ML: Machine Learning

Fig. 4.10 Agent-supported simulation.

ing abilities. Figure 4.10 provides an abstract view of the elements of an agent-supported simulation system.

The knowledge processing and generation component (i.e., KGen1) can be viewed as the frontend subsystem that symbolically manipulates user or system inputs to improve the operation of a simulation and/or model specification/development environment. Simulations are goal-directed experimentation systems, and agents can play a significant role in evaluating domain-specific knowledge and objectives to select models, input data models, and proper experiment designs to support the specification, design, and composition of simulation models. A knowledge generation subsystem can also aid the simulation at runtime by filtering data and integrating relevant information to generate hypotheses for use in the simulation. The backend interface can also be augmented by another knowledge generation subsystem to make inferences about simulation output. Output analysis methods such as reliability assessment and ANOVA can be carried out by agents to suggest a proper course of actions and visualizations for decision makers. Concomitantly, the learning component may facilitate fine tuning/adapting the simulation input parameters and domain-specific knowledge based on the observed output of the simulation, as well as the state of the context that embeds the simulation system.

To illustrate the utility of intelligent agent technology in simulation development and new simulator design, we look into two studies (Yilmaz and Pasqualetti, 2005; Yilmaz, 2007) conducted by one of this chapter's authors. Specifically, first, we overview the role of mediator and facilitator agents in improving simulation integration and interoperability.

4.6.1

Agent-Mediated Interoperation of Simulations

Interoperation and dynamic composability of disparate simulations represent a longstanding challenge within the M&S community. While both issues are extensively studied, a unified and coherent strategy that facilitates achieving seamless

and transparent congruity among conceptual and realization spaces of simulations is still an elusive goal. Supporting cooperative interaction among globally available simulations requires the seamless exchange and sharing of data, information, knowledge, and models. The constant evolution and changes in these resources complicate seamless discovery, location, and retrieval of “sufficiently” relevant services for the task at hand. Furthermore, the heterogeneity underlying the syntax and semantics of these services requires intelligent facilitators and mediators for content exchange among producers and consumers that have different worldviews in defining and interpreting such services.

In Yilmaz and Pasquetti (2005), an agent-supported metalevel interoperation architecture is proposed to address these challenges. The strategy involves the introduction of an agent framework independent of the simulation infrastructure for explicit separation of interoperation protocols from the simulation environment. To this end, agent-based mediation, brokering, matchmaking, and facilitation services are suggested as critical components for interoperation and composability. The proposed approach is based on the following premises: (1) deployment of brokering protocols improves transparency and balances the workload in interoperation, (2) partial matchmaking mechanisms provide more efficient and effective model and data qualification strategies than the conventional keyword-based matching mechanism, (3) automated mediation facilities that are deployed independently of the simulation infrastructure improve transparency and runtime extensibility.

The following are the minimal requirements for an infrastructure that aims to support interoperation among simulations in a federated system (Tolk and Diallo, 2005).

- *Administration* is the process of managing the information exchange needs that exist between the services. Administration involves the overall information management process for the service architecture. Location, discovery, and retrieval of content are critical components of administration.
- *Management* involves identifying, clarifying, defining, and standardizing the meaning of content.
- *Alignment* ensures that the data to be exchanged exist in the participating systems as an information entity or that the necessary information can be derived from the available models and services published by participants.
- *Transformation* is the technical process of aggregation and/or disaggregation of the information entities of the embedding systems to match the information exchange requirements.

An agent-mediated strategy has been suggested by Yilmaz and Pasquetti (2005). Figure 4.11 depicts the elements of this strategy in terms of an agent organization that constitutes facilitator, mediator, broker, and matchmaker agents that perform the necessary management, alignment, and transformation functions.

Furthermore, the agent organization aims to decouple the simulation from the intricate details of instantiation and interoperation of a family of models to avoid explicit assumptions and facilitate seamless reconfiguration with alternative ensembles. This way, the agent organization abstracts the simulation instantiation and

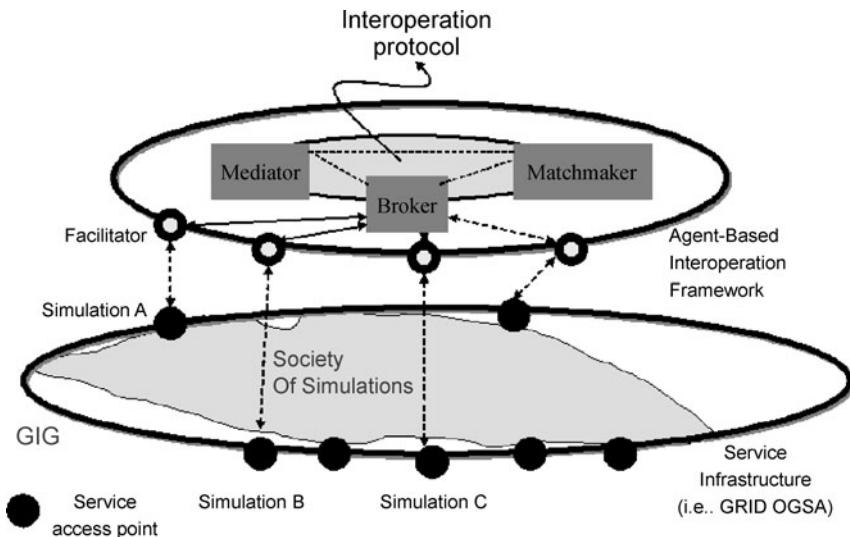


Fig. 4.11 Agent-mediated interoperation framework.

interoperation process. It helps make a simulation system independent of how its models are created, composed, and represented. The organizational domain encapsulates the knowledge about which models the simulation uses. Furthermore, the concrete organization hides the details about how simulation programs for these models are created and composed together. Therefore, the decoupling of the instantiation and interoperation processes from the simulation infrastructure gives significant flexibility in terms of what gets instantiated and exchanged, who instantiates it, how it gets created and transformed, and when.

4.6.1.1 The Facilitator Agent

The facilitator agent acts as a gateway between the simulation infrastructure and the agent organization that orchestrates the simulation interoperation. Simulations join a society of simulations by registering their facilitator with the metalevel interoperation protocol. As a controller, the facilitator agent is aware of the capabilities and needs of the simulation service that it is associated with. The requests coming from the simulation domain will be delegated to brokering, matchmaking, and mediation agents in accordance with the embedded interoperation protocol that facilitates seamless data discovery, location, retrieval, and transformation.

4.6.1.2 The Mediator Agent

The mediator agent is responsible for converting simulation content to/from a common reference model (i.e., C2IEDM). To facilitate mediation, conflicts between the assumptions and obligations of simulations need to be resolved. As discussed in Tolk and Diallo (2005), four types of conflicts are common between the desired and provided services.

- *Semantic* conflicts occur when the local schema objects must be aggregated or disaggregated but fail to match.
- *Descriptive* conflicts occur when the same concept is described in terms of synonyms, homonyms, or different attributes and values.
- *Heterogeneous* conflicts occur when concepts are described in terms of substantially different methodologies.
- *Structural* conflicts occur when concepts are defined in terms of different structures in the schemata (metadata).

The mediation agent uses a common reference model to convert between the data encoding standards. Note that the same strategy is applicable to models. That is, models can be converted from one formalism to another to facilitate the use of disparate simulations developed within distinct conceptual frameworks deployed in the infrastructure.

4.6.1.3 The Broker Agent

The interaction between content (i. e., data, model) requesting facilitators (consumers) and potential service providers (producers) are achieved via flexible mechanisms that can vary depending on the characteristics of the application domain. Brokering among content producers and consumers in a federated simulation system brings flexibility via fine-grained interoperation that takes specific constraints of individual simulations into account. For instance, two simulations that use a common ontology and vocabulary may not require mediation. In that case, the recommendation protocol would be sufficient, as the producer and the consumer can directly exchange data and services through their facilitators. On the other hand, when conflicts among simulations need to be resolved, the recruitment protocol can be used so that the broker can submit the requests to the producer along with instructions for mediation. In cases where producer and consumer need to be synchronized, the broker agent can shift to a notification protocol between two specific facilitators.

4.6.1.4 The Matchmaker Agent

For dissemination and recommendation of services, various protocols in different contexts have been studied. Matchmaking is the cooperative partnership between information providers and consumers along with the help of an intelligent facilitator (Genesereth and Singh, 1993). Using this approach information providers actively publish their capabilities and services to the matchmaker, and the consumers send requests for their desired information to the matchmaker. Matchmaking enables the providers and requesters to exchange dynamically changing information in a more effective and active manner than traditional methods. Matchmaking mechanisms have been widely used in various applications and fields where information changes rapidly, such as product development and crisis management. And yet the dynamic interoperation of such services is critical for the coherent operation of the overall system. Furthermore, a matchmaking strategy with quantitative distance metrics provides insight about the extent of alignment needed for

two (data) models to interoperate. Hence, a matchmaking agent needs to cooperate with the mediator agent within the envisioned interoperation protocol to improve the degree of automation in interoperation.

4.6.2

Agent-Supported Simulation for Decision Support

In this section, we briefly overview a case study that illustrates the use of agent technology as a support facility to enable symbiotic decision support. Decision science involves understanding cognitive decision processes, as well as methods and tools that assist decision making. Strategy problems are typically characterized by significant uncertainty. Proper simulation-based decision support methodologies that are consistent with the way experts use their experience to make decisions in field settings could improve modeling a course of actions (COA), simulating them faster than real time, and then performing COA analysis. Real-time decision making requires models and tools that allow interaction with the system of interest. A symbiotic decision support system is defined as one that interacts with the physical system in a mutually beneficial way. It is highly adaptive, not only performing “what-if” experiments to control the physical system but also accepting and re-

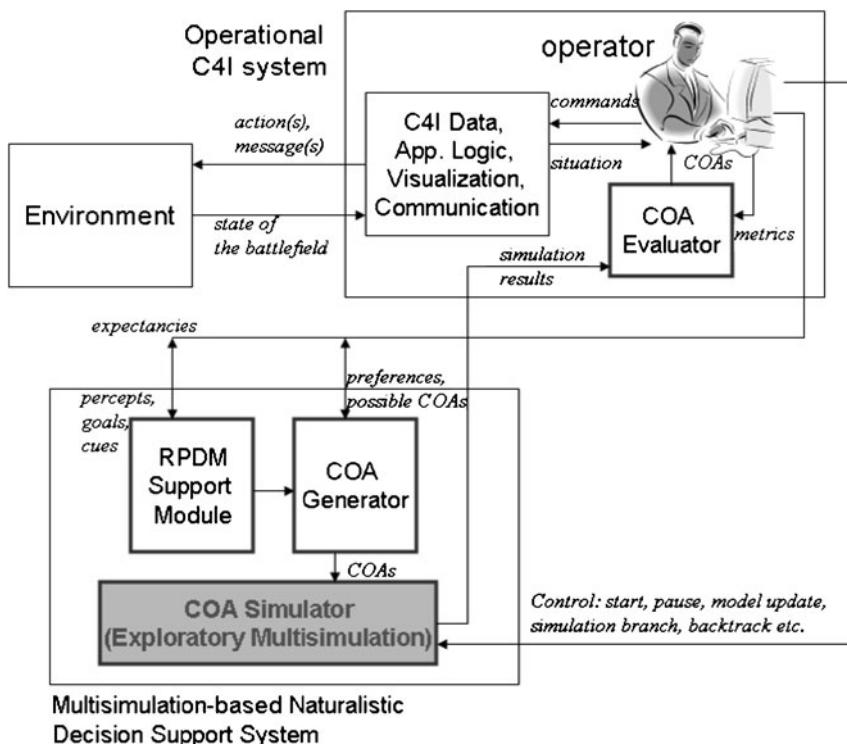
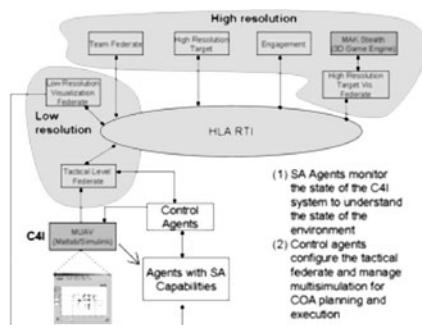


Fig. 4.12 Architecture of an agent-supported decision support system.

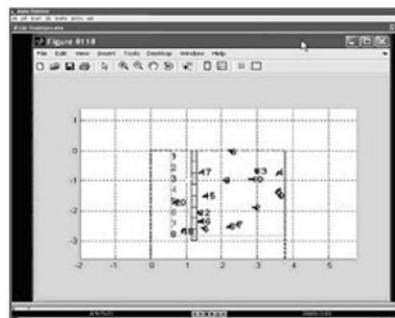
sponding to data from the physical system. Intelligent agents are used in this study to filter, perceive, and understand the state of a system to drive the simulation system to explore outcomes of potential COAs. Figure 4.12 depicts the architecture of an agent-supported simulation-based decision support system that illustrates the knowledge generation and machine learning components of the abstract view presented in Figure 4.9.

In this study, a multiresolution coordinated mission for unmanned air vehicles is used to gain experience and identify the means for the principled design of such systems. The Recognition Prime Decision Module (RPDM) and COA generator constitute the situation awareness component that involves perception, understanding, and anticipation functions. The COA simulator, which is based on multisimulation, explores the outcomes of potential COAs recommended by the control agents.

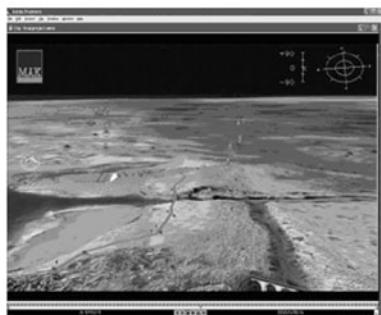
We use multi-UAV (MUAV) simulation (Figure 4.13) as an experimental framework that drives the multiresolution simulation framework that constitutes the tactical and high-resolution simulations. Tactical federate is the component with which the decision maker interacts to (1) monitor situation models generated by the system and (2) use multisimulation in decision making. The interaction between MUAV and tactical federate is handled by a separate component that



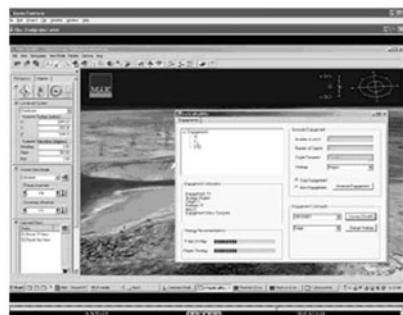
Components of the Prototype



C4I Simulation



High-resolution Simulation



Creating a New Engagement

Fig. 4.13 Situation awareness and controller agents for COA generation.

clusters the targets to identify battalions and establish teams based on an agent-based contract net protocol (Smith, 1980), which is a distributed task allocation algorithm. Note that the clustering and team formation algorithms are handled by intelligent agents that augment the decision making process of the operator. The tactical federate uses the information passed from the MUAV to instantiate and configure a team with a strategy. The strategy is identified using a Bayesian network that constitutes the anticipation element of the decision making life cycle supported by intelligent agents. The team, high-resolution, and engagement federates, along with MAK Stealth 3D Game Engine, constitute the high-resolution simulation. Situation awareness and controller agents make recommendations for updating the multisimulation. Situation awareness is defined as the perception of elements in a particular environment within time and space, the comprehension of their meaning, and the projection of their status in the near future. Situation awareness, as depicted here, provides a set of mechanisms that enable attention to cues in the environment. The understanding, diagnosis, and COA generation processes in this study are based on the following phases coordinated by agents: (1) target clustering, (2) situation model (target graph) generation for each UAV, (3) consensus model generation for collective understanding of the situation, and (4) expectancy and COA generation.

4.7 Summary

In light of this framework and the elaborated categories of the use of agents in simulation, we anticipate that the use of agents will increase as the technological context extends with emerging trends and critical drivers such as augmented cognition, semantic web, web services and service-oriented computing, grid computing, ambient intelligence, and autonomic computing become more pervasive. Simulation-based design of such systems will require seamless introduction of agents and/or agent technologies. We expect that the use of simulation for the design of agents will become a new avenue for research. Analytic and heuristic methods in specifying the deliberative architecture of agents are common. Simulation can play a significant new role in the design of agents in such a way that software agents use simulation to make decisions before acting on the environment. A nested simulation concept, in which simulations are embedded within agents, is another possibility. While the use of agents as simulation design metaphors is common, the use of simulation for agent behavior generation is rare.

As a final note, we believe that just as there is no single agent-based abstraction that is uniformly better on all types of problems, there is no one technology (e.g., evolutionary computation, artificial neural nets, symbolic machine learning, and mathematical optimization) that is sufficient to account for the complexity and difficulty of most real-world problems. Rather, we have to recognize that complex real-world systems are developed using a broader systems engineering perspective that requires seamless integration of appropriate technologies.

References

- Agha, G. and Hewitt, C. (1985) Concurrent programming using actors: Exploiting large-scale parallelism. *Proceedings of the Foundations of Software Technology and Theoretical Computer Science, Fifth Conference*, pp. 19–41.
- Bankes, S. (1993) Exploratory modeling for policy analysis. *Operations Research*, **41** (3), 435–449.
- Bates, J. (1994) The role of emotion in believable agents. *Communications of the ACM*, **37** (7), 122–125.
- Denning, P.J. (2007) Computing is a natural science. *Communications of the ACM*, **50** (7), 13–18.
- Epstein, J.M. (2006) *Generative Social Science: Studies in Agent-Based Computational Modeling*. Princeton University Press, Princeton, NJ.
- Ferber, J. (1999) *Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence*, Addison-Wesley.
- Genesereth, M. and Singh, N. (1993) A knowledge sharing approach to software interoperation. *A Knowledge Sharing Approach to Software Interoperation, Logic Group, Computer Science Department*.
- Ghassem-Aghaei, N. and Ören, T.I. (2003) *Towards Fuzzy Agents with Dynamic Personality for Human Behavior Simulation*. Proceedings of the 2003 Summer Computer Simulation Conference, pp. 3–10.
- Holland, J.H. (1995) *Hidden Order: How Adaptation Builds Complexity*, Helix Books (Addison-Wesley), New York.
- Kakehi, R. and Tokoro, M. (1993) *A Negotiation Protocol for Conflict Resolution in Multi-Agent Environments*. Proceedings of the International Conference Intelligent Cooperative Information Systems, Rotterdam, The Netherlands, pp. 185–196.
- Little, R.G. (2005) *Organizational Culture and Performance of Critical Infrastructure: Modeling and Simulation of Socio-Technological Systems*. Proceedings of the 38th Hawaii International Conference on System Sciences, Hawaii, USA.
- Luck, M., McBurney, P. and Preist, C. (2003) *Agent Technology: Enabling Next Generation Computing – A Roadmap for Agent Based Computing*, Agentlink.
- Miller, J.H. and Page, S.E. (2007) *Complex Adaptive Systems: An Introduction to Computational Models of Social Life*, Princeton University Press, Princeton, NJ.
- Muller, J.P. (1996) *The Design of Intelligent Agents: A Layered Approach*, vol. 1177 of *Lecture Notes in Artificial Intelligence*, Springer.
- Ören, T.I. and Yilmaz, L. (2004) *Behavioral Anticipation in Agent Simulation*. Proceedings of the Winter Simulation Conference, Washington D.C., pp. 801–806.
- Sen, S. and Weiss, G. (1999) Learning in multiagent systems. chapter 6 of *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, pp. 259–298.
- Shoham, Y. (1993) Agent-oriented programming. *Artificial Intelligence*, **60** (1), 51–92.
- Smith, R.G. (1980) The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Transactions on Computers*, **29** (12), 1104–1113.
- Sun, R. (2006) *Cognition and Multi-Agent Interaction: From Cognitive Modeling to Social Simulation*, Cambridge University Press.
- Tolk, A. and Diallo, S. (2005) Model-based data engineering for web services. *IEEE Internet Computing*, **9** (4), 65–70.
- Van Dyke Parunak, H. (1999) Industrial and practical applications of dai. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, pp. 377–421.
- Weiss, G. (1999) *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, MIT Press, Cambridge, MA.
- Wooldridge, M. (2002) *Introduction to Multiagent Systems*, Wiley.
- Yilmaz, L. (2007) Toward next generation simulation-based computational tools for conflict and peace studies. *Social Science Computer Review*, **25** (1), 48–60.
- Yilmaz, L. and Ören, T.I. (2005) *Agent-Directed Simulation, Special Issue of the Simulation: Transactions of the Society for Computer Simulation International*, vol. 81 (7), 3–5 (July 2005).
- Yilmaz, L. and Paspuletti, S. (2005) Toward a meta-level framework for agent-supported interoperation of defense simulations. *Journal of Defense Modeling and Simulation*, **2** (3), 161–175.

Yilmaz, L. and Phillips, J. (2007) The impact of turbulence on the effectiveness and efficiency of software development teams in small organizations. *Software Process: Improvement and Practice Journal*, 12 (3), 247–265.

Zeigler, B.P., Praehofer, H. and Kim, T.G. (2000) *Theory of Modeling and Simulation*, Academic Press.

Part Three Systems Engineering and Quality Assurance for Agent-Directed Simulation

5

Systems Engineering: Basic Concepts and Life Cycle

Steven M. Biemer and Andrew P. Sage

5.1

Introduction

The primary objective of this chapter is to provide a foundational understanding of the multifaceted and transdisciplinary field of systems engineering. This foundation is necessary as it provides the context for the specific and applied topics that follow in other chapters in this work. However, describing systems engineering is not a simple task, especially within a single chapter. This description requires understanding two interconnected ideas: the basic concepts and principles of the field today, wrapped within the lifecycle of a system's development. While some concepts are universal (such as understanding the system from the perspective of the "whole"), most concepts are better described within the various phases that most complex systems traverse as they evolve from idea to an actual, deployed engineered system.

Also, systems engineering focuses on the engineering of large-scale, complex systems (Sage, 1992). Simple, or small, systems need not be burdened with all of the formalism of systems engineering processes. Unfortunately, most systems developed today by government and private corporations do not fall into the simple category. Whether it is a new jet fighter system, a business system for a large corporation, a complex computer simulation, or a new public transportation vehicle, the complexity of new systems has grown to the point of needing systems engineering discipline to ensure success (Sage and Biemer, 2007).

This chapter is organized into four basic parts. The first part describes the goals of exploring agent-based systems engineering and its related fields and introduces why a chapter on classic systems engineering is needed in a book such as this. The second part describes the definition, attributes, and need for classical systems engineering. Although we alluded to this need above, this section provides details on where the systems engineering discipline assists in the system development effort. The third part of the chapter defines a generic life cycle model applicable to almost all system development efforts. The various phases within this life cycle model will be explored, and specific systems engineering principles, methods,

techniques, and tools will be provided. The last part identifies and describes four basic concepts that are applicable at the intersection of systems engineering and system life cycle.

5.2

Agent-Based Systems Engineering

Agent-based modeling within software engineering has been developed and used for well over a decade (Wooldridge, 1997). Other chapters in this work describe in detail the structure and attributes of agents within the modeling and simulation environment. Agent-based systems engineering (ABSE), however, is a relatively new concept, though it represents a logical extension of software engineering applications.

At the heart of ABSE is the development of a decentralized and distributed information system, requiring large-scale control and coordination (Crespi and Cybenko, 2005). These types of systems contain multiple elements that exhibit independent, or at least semi-independent, behaviors that enable accomplishment of their separate goals. It is appropriate to represent these elements as autonomous or semiautonomous agents operating in a dynamically changing environment. Using this representation, we can hypothesize that desired and stable global behaviors will evolve or emerge from the collection of individual elements.

Of course, there are downsides to the agent-based approach (Jennings, 2000). One of those downsides is the ability to predict overall system behavior. So, if predicting the behavior of the total system is problematic, then how do we use this approach in developing complex systems? This is where the concept of ABSE provides opportunities.

ABSE aims to combine the fields of systems engineering with artificial intelligence with the goal of exploring systems modeling and performance, and tying these abilities to their scientific foundations. Classical systems engineering focuses primarily on the operational and physical domains, while artificial intelligence deals with human perception, decision making, and actions and behaviors. This combination may lead to a powerful new subdiscipline within both systems engineering and artificial intelligence, allowing us to adequately characterize and predict emergence behaviors of complex systems, leading to innovative methods of developing these types of systems. Thus, we need to introduce the foundational principles and practices of systems engineering before we discuss their application to agent-directed simulation and, ultimately, agent-directed systems engineering.

5.3

Systems Engineering Definition and Attributes

First and foremost, systems engineering is a transdisciplinary management technology (Sage, 2002). If we parse this sentence, we see three components to sys-

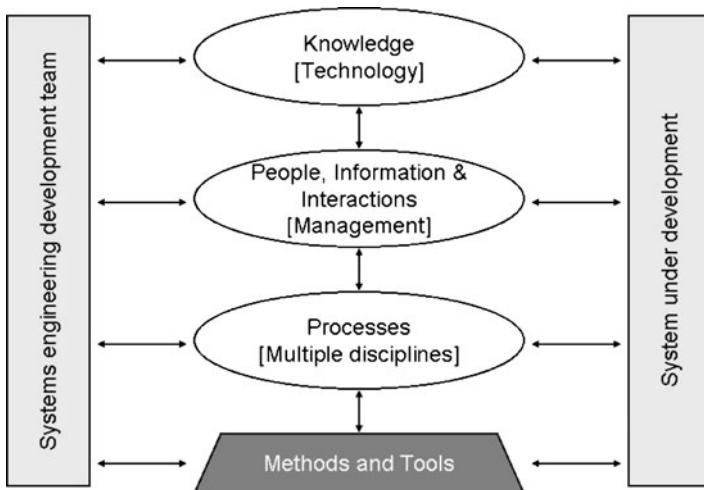


Fig. 5.1 Conceptual illustration of the three components of systems engineering.

tems engineering: technology, management, and multiple disciplines integrated together. Technology involves the manipulation, extension, and dissemination of scientific knowledge for some betterment of human activity. Therefore, systems engineering must include, at its basic core, the organization and dissemination of knowledge. Second, management involves the internal and external interactions of an organization. Thus, systems engineering involves the creation, organization, and workings of groups of people, and the interactions and information that flow between them. Third, systems engineering brings together all of the disciplines necessary for the development, manufacture, and deployment of a system. These multiple disciplines must be integrated into one or more structured processes. Thus, three terms can summarize the field of systems engineering: knowledge, people, and processes. How these three entities are brought together is what distinguishes this field from basic management (including program management).

Although not specifically mentioned in the above paradigm, methods and tools become the foundational mechanisms for integrating and implementing the three facets of systems engineering. The combination of the three components and of the underlying tools can be depicted in the following figure, taken from Sage (2002) and expanded to include all of the components discussed above.

5.3.1

Knowledge

The last half of the 20th century and the first part of the 21st century have seen explosive growth in technology and knowledge. This rapid growth has led to new fields and systems (e.g., communications, space travel, entertainment, finance), as well as the increased complexity of those systems. With this new complexity comes the requirement for new procedures, materials, interactions, and tools. Technology

associated with these new fields continues to advance at a very rapid pace. Thus, one of the key components of systems engineering is to stay ahead of the new technologies and the knowledge required to understand them (Kossiakoff and Sweet, 2002).

It is useful to distinguish between knowledge, information, and data (Sage and Rouse, 2002). Information is data arranged in meaningful patterns such that it is potentially useful for decision making. While knowledge can be defined as something that is believed, true, that works, and is reliable. With these definitions, we can argue that systems engineering manages both knowledge and information. However, managing knowledge (especially with respect to technology) is different than managing information (which is usually associated with management of organizations). Managing knowledge involves the development and application of technologies to appropriate problems (technical, social, organizational, programmatic, etc.), while managing information is creating, collecting, storing, organizing, and disseminating information to the right people who need it. Fundamentally, the systems engineer does both. As Sage and Rouse (1999) assert, systems engineers are brokers of information and knowledge leading to the definition, development, and deployment of systems of all types.

With higher complexity and newer technologies comes risk, in all of its various forms. Identifying, understanding, managing, and, ultimately, mitigating risk are key concepts in the field of systems engineering. Complexity, technology, and knowledge need not be feared by the engineering community, as long as the principles and practice of systems engineering continue to evolve with them.

5.3.2

People and Information Management

The systems engineer forms a partnership with the program manager in developing a new system. As Figure 5.1 illustrates, this partnership forms the foundation of a team of engineers, specialists, managers, and manufacturers who will actually develop the system. This team interacts with a host of stakeholders, vendors, suppliers, subcontractors, government officials, company officials, and others to perform the appropriate liaison work necessary for guiding a new system through the development process.

As we mentioned above, the systems engineer brings people, information, and organizations together to interact in appropriate ways to make decisions and apply their expertise. A useful construct for thinking about these interactions, involving the people and organizations that must interact, and the information that must be collected, processed, and passed from one entity to another, is to examine the six interrogatives: who, what, where, when, why, and how. Zachman (1987) uses the six interrogatives as a basis for his architectural framework. Using Sage's and Rouse's adaptation, we can form the two-dimensional framework in Figure 5.2. Sage and Rouse (2002) have categorized the six interrogatives into information and knowledge interrogatives to distinguish between those that relate to information and those that relate to knowledge.

	Information Interrogatives				Knowledge Interrogatives	
	Entities (What)	Time (When)	Locations (Where)	People (Who)	Functions/Activities (How)	Purpose/Motivation (Why)
Stakeholders	Policy Makers					
	Planners					
	Enterprise Owners					
	Systems Engineers/Architects					
	Builders					
	Impacted Publics					

Fig. 5.2 Stakeholder/interrogative framework.

Each cell within the above matrix represents a separate consideration when developing a system. Each stakeholder does not necessarily need information and knowledge across the interrogatives. For instance, policymakers may not need information on the entities (what) but would be very interested in the purpose and motivation for the systems. Conversely, builders may not be interested in the purpose so much as the entities. Thus, the systems engineer must keep the proper perspective in mind when organizing the system team.

Thus we see that, in their management capacity, the systems engineer and the program manager select the team, organize and track the effort, gather and define the information flow, and facilitate the interactions among and with the team. Utilizing modern management techniques from experience and learning, the systems engineer will ensure that his team operates like a system itself in order to accomplish the intended development effort.

5.3.3

Processes

Kossiakoff and Sweet (2002) define a major system development project as a veritable Tower of Babel. In this tower, specialists from different fields and disciplines converge to develop and produce a successful new system. Each specialist has his or her own language, experience, and knowledge bases. Of course, these are peculiar to the specialist's field. The systems engineer, more than anyone (including the program manager), is responsible for bringing these specialists together and bridging the language gap.

As Kossiakoff and Sweet point out, however, the depth of interdisciplinary knowledge that is required to interact with each specialist is not the same depth necessary to work effectively in the specialist's field. Thus, the systems engineer need not be

Classic Software Waterfall Life Cycle (also called Linear Sequential Model)

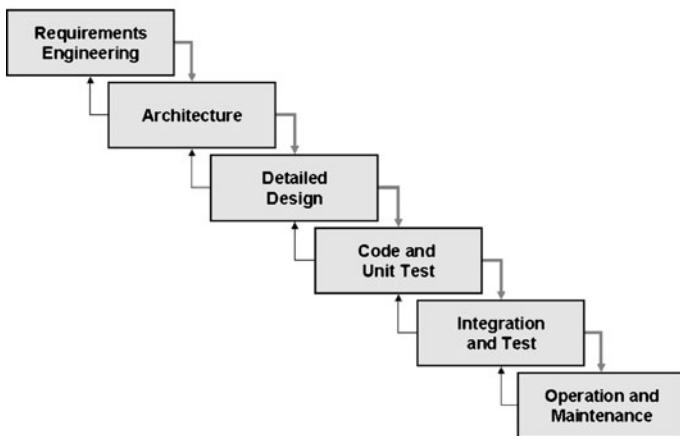


Fig. 5.3 Classic software waterfall process model.

Systems Engineering “Vee” Process

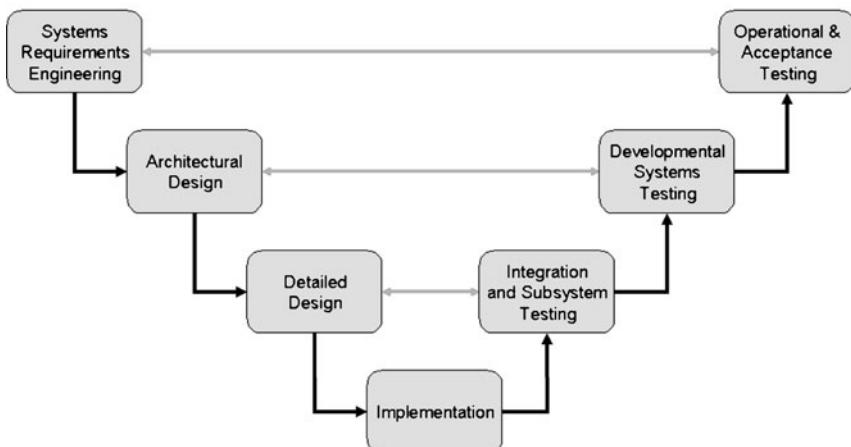


Fig. 5.4 Systems engineering “Vee” process model.

an expert in every field; he or she must only be able to communicate with each specialist and bring them all together within a common language framework. Part of this language framework is a common (or at least tailorable) process. And while many standards exist with respect to processes (discussed below), the systems engineer must develop and promulgate a single, tailored process for the team.

An additional dimension that sometimes confuses the discussion on systems engineering processes is the fact that academia generally uses simplified processes to facilitate education. Thus, it is common to see the classical software waterfall pro-

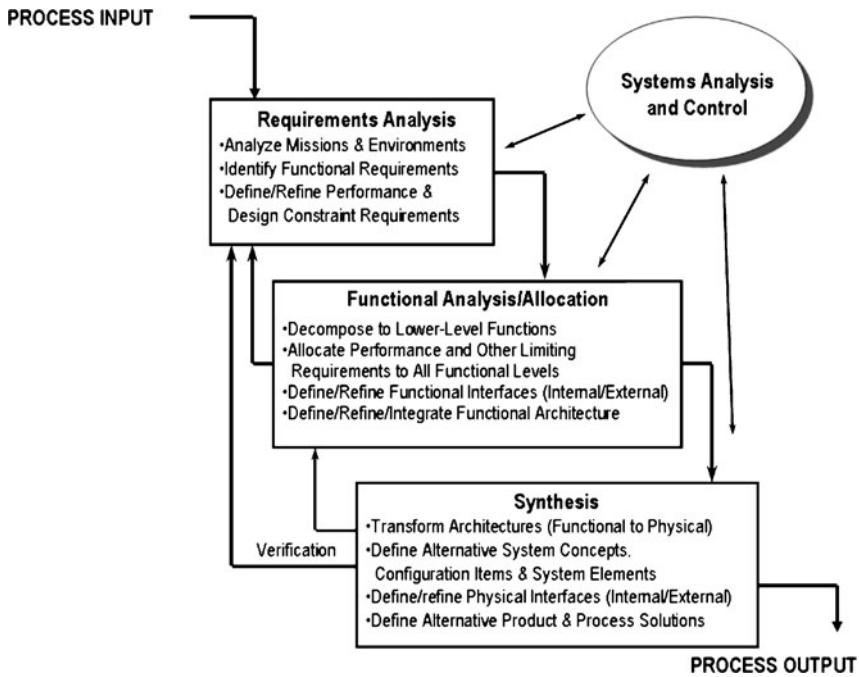


Fig. 5.5 MIL-STD-499B systems engineering process.

cess or the traditional systems engineering Vee process model used in instruction (shown below in Figures 5.3 and 5.4). Neither process is common in the practitioner's domain; nor are these process models recognized in the current standards. However, they can be effectively used to teach systems engineering principles. Additionally, they contain the basic phases that need to occur in some fashion and in some order. Thus, the systems engineering instructor can focus on these phases individually within this simple model.

Regardless of which process model is used for illustration, or discussion, the systems engineer creates a unique process for his or her particular development effort. And this process must be consistent with the acquisition plan for the system, the management philosophy of the organization, the business model used within the enterprise, and the cost, schedule, and performance constraints of the program.

Sage and Biemer (2007) summarize the three processes that are described in the current set of standards, and one that is still used, although the standard itself has been discontinued. This standard, MIL-STD-499B (MIL-STD-499B, 1991), is still followed within many US DoD communities. Figure 5.5 depicts this process. The process consists of four basic activities: requirements analysis, functional analysis and allocation, synthesis, and system analysis and control. The first three are generally sequential, although feedback is prevalent.

The processes within current standards are the IEEE-1220 (IEEE, 1999), the EIA-STD-632 (Martin, 2000), and the ISO-IEC-IEEE-STD-15288 (IEEE, 2005). The first,

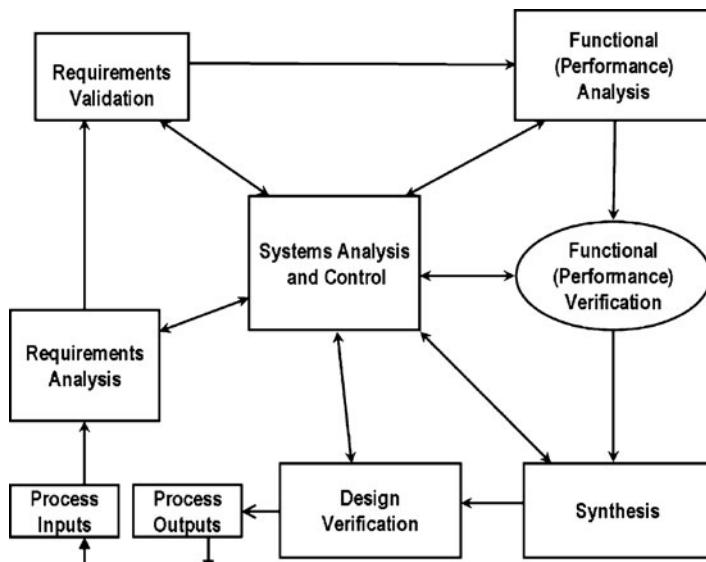


Fig. 5.6 IEEE-1220 systems engineering process.

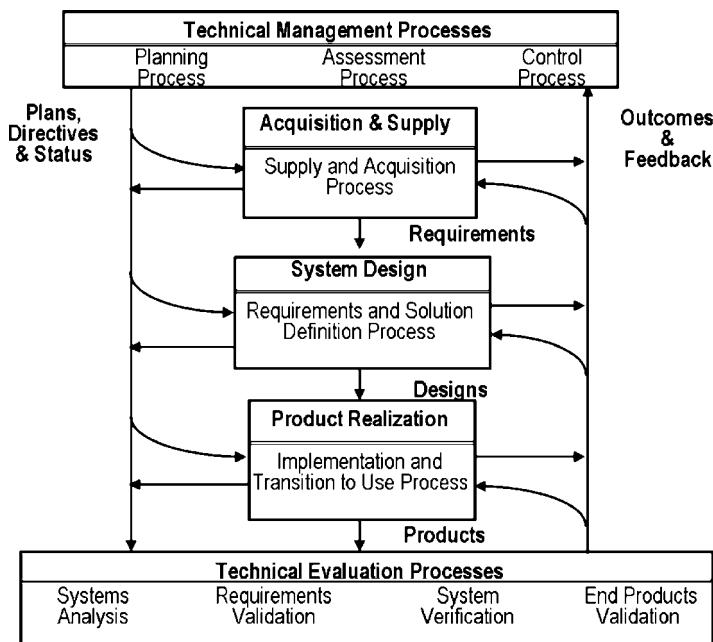


Fig. 5.7 EIA-STD-632 systems engineering process.

and perhaps the simplest, process is the IEEE-1220 (IEEE, 1999), depicted in Figure 5.6. Besides the main control activity in the middle of the graph, the general flow of activities is clockwise, starting from the bottom left, beginning with "Pro-

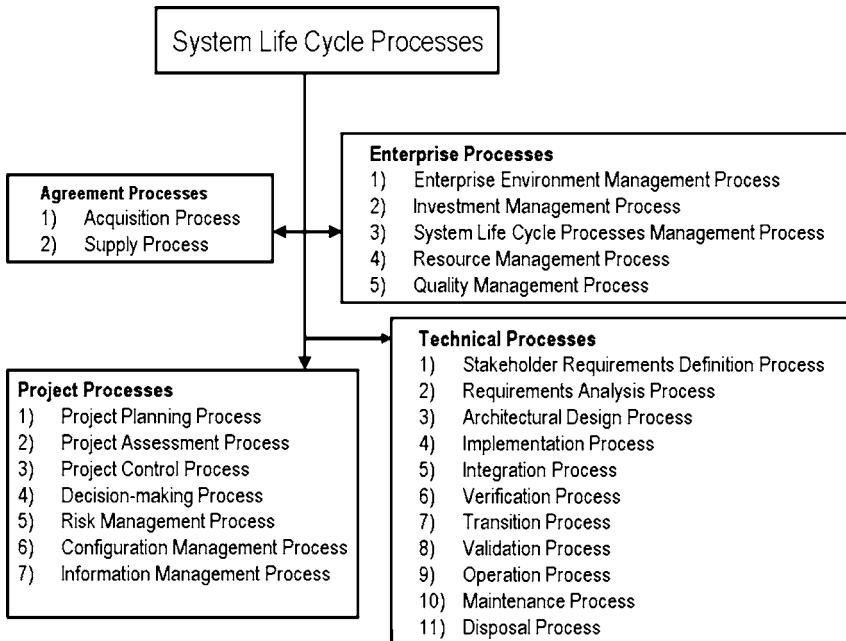


Fig. 5.8 ISO-IEC-IEEE-STD-15288 systems engineering process.

cess Inputs". It is interesting and important to note the similarities with the related military standard. The second systems engineering process is depicted in Figure 5.7. EIA-632 (Martin, 2000) expands the basic activities included in the above standards and includes additional evaluations in verifying and validating designs, both conceptual and real. Additionally, the standard includes postdevelopment activities such as deployment and the transition to use. Feedback loops are included throughout to indicate the iterative nature of the process.

A basic feature of EIA-632 is that the standard identifies multiple processes that, when integrated together, provide the systems engineer with an overall methodology for developing a system (Sage and Biemer, 2007).

Finally, we present the ISO (IEEE, 2005) standard in Figure 5.8. Unlike the others presented, the ISO standard is not a process in the traditional sense. The standard describes many activities, each one a separate process in itself. It is the systems engineer who determines the activities based on the needs of the system development effort and links them together in a master process. Thus, every project is different from an overarching process perspective. This standard is, indeed, the epitome of flexibility. Of course, the systems engineer must be skilled to identify and link the appropriate activities together, knowing what do execute in sequence and what should be accomplished in parallel.

5.3.4

Methods and Tools

We separate methods and tools from the three basic concepts of systems engineering so as to distinguish the building blocks that are used (and continually evolving and improving) from the basic components of systems engineering. Most methods and tools are not unique to the field of systems engineering but in fact arise in some other engineering or scientific discipline. These methods are adopted by, and adapted to, systems engineering.

One key tool in the systems engineering toolbox is the model and the dynamic representation of the model known as simulation. Law (2006) defines a model simply as a representation of a system, and a simulation numerically exercises a model for a given set of inputs. And while a specific form of modeling and simulation is the topic of this book, all forms of models and their counterpart simulations can be, and are, used within systems engineering for multiple purposes.

Beyond simulation, systems engineers have a variety of mathematical and logical techniques and methods for calculating the performance, effectiveness, and efficiency of alternative conceptual and actual system designs. These methods can be categorized to fit specific applications or aspects of a system design. Some examples of these categories, by no means an exhaustive listing, are given below (Reilly, 1993):

- Decision making techniques (e.g., analytical hierarchy process);
- Risk analysis (e.g., fault tree analysis equations);
- Project planning (e.g., critical path method);
- Reliability (e.g., probability of failure);
- Optimization (e.g., linear programming);
- Economic analysis (e.g., cost estimation equations).

And finally, the systems engineer has a plethora of diagramming, graphing, and drawing models that can be used to identify, evaluate, and communicate system designs. These also can be categorized, but they number quite extensively. Thus, listing them here would be inappropriate. Many of these graphically based tools have been grouped into languages or frameworks. For example, the Unified Modeling Language, known as UML, identifies and defines many diagramming techniques to represent designs and architectures within the object-oriented construct (Booch et al., 1999). Other languages and frameworks exist as well.

Thus, we have a multitude of methods, tools, and techniques available. And while some have gained popularity over the years with a particular generation of systems engineers, together they represent the ever-growing toolbox to generate, evaluate, and communicate the system design.

5.3.5

The Need for Systems Engineering

We come now to the question, why systems engineering? For those managers without experience in complex systems development, this becomes a common question when attempting to institute systems engineering processes and principles into a new effort. After all, this is going to cost the program money. And the bottom line needs to improve in order to demonstrate a return on investment (ROI) in systems engineering. So our question returns, why systems engineering?

The answer is simple to state, but much harder to demonstrate. First, however, we need to issue a caveat before answering the question. Applying systems engineering to the development effort yields a ROI only for complex systems. Designing a new toaster or hair dryer may not yield benefits from systems engineering; developing a new fighter aircraft would. We also realize that at this point, this statement is simply conjecture without evidence. What makes supplying evidence so difficult is that a side-by-side comparison is needed to “prove” ROI. And while we don’t have such an example, we do have many examples of systems development efforts that did not use systems engineering and failed (as defined by meeting users’ requirements). Blanchard and Fabrycky (2006) shared their experience of evaluating past systems development efforts, which led them to conclude that most problems were a direct result of not applying a disciplined, top-down “systems approach”. Furthermore, they described three benefits that had been realized when using a systems engineering process in a complex systems development program:

1. Reduction in system design, development, production, operation, and support costs;
2. Reduction in system acquisition time (largely from reductions in fixes that traditionally need to be implemented late in the program); and
3. More visibility and a reduction in design risks.

The degree to which these reductions are realized depends, of course, on the specific nature of the program. But in general, the more complex the system, the higher the benefit from applying systems engineering techniques has been.

So not only do we now have an answer to the question why systems engineering?, we also have at least one metric to evaluate success: does the system meet user requirements?

5.4

The System Life Cycle

Many people fail to understand that the systems engineering process (or processes, as the numerous processes presented above demonstrate) and the system life cycle are different. In general, the systems engineering process chosen or adapted is repeated for each of the major phases of the system life cycle. Of course, different

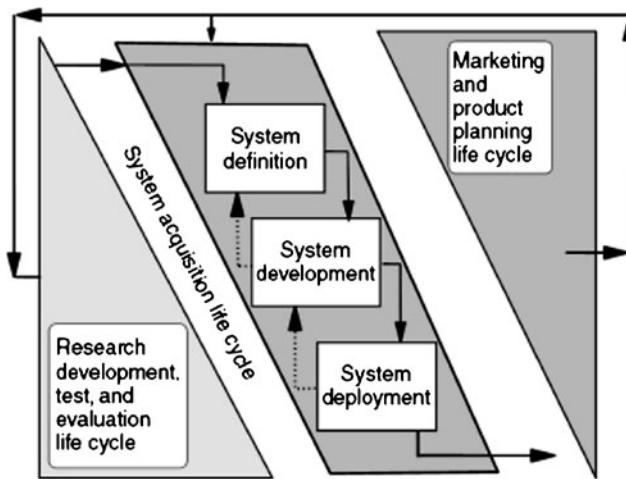


Fig. 5.9 Three primary systems engineering life cycles.

steps in the systems engineering process may be emphasized during each phase of the life cycle.

Kossiakoff and Sweet (2002) introduce the system life cycle with a general three-phased approach that is common to all life cycle models currently in use. It is comprised of concept development, followed by engineering development, followed by postdevelopment.

Concept development is the initial stage of formulation and definition of a system concept perceived to best satisfy a valid need. Engineering development covers the translation of the system concept into a validated physical system design meeting the operational, cost, and schedule requirements. Postdevelopment includes the production, deployment, operation, and support of the system throughout its useful life.

Sage (1992, 2002) also presents a system acquisition life cycle model that mirrors the Kossiakoff and Sweet model Kossiakoff and Sweet (2002) and brackets the three stages with the research, development, test, and evaluation life cycle and the marketing and product planning life cycle. This is illustrated in Figure 5.9 (Sage, 1992, 2002).

The decomposition of these basic three stages into additional phases is dependent on the problem domain. Government agencies tend to define a unique life cycle model by its system development activities. All major system development is constrained to follow the basic model, with some exceptions sanctioned by each organization. Perhaps the best example is the US Department of Defense (DoD). The official life cycle model is illustrated in Figure 5.10.

The triangles represent milestones in which major program decisions are made. In essence, the system must “pass” these milestone points to continue on to the next phase. Once a system has passed milestone C, low rate production may commence. At Initial Operational Capability (IOC), a decision is made to transition to

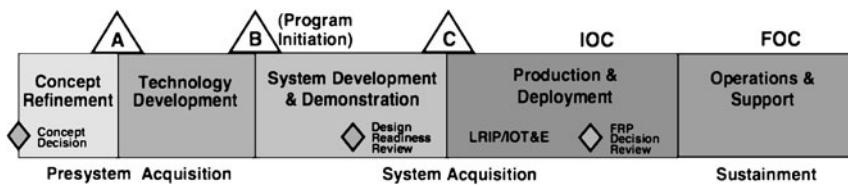


Fig. 5.10 2007 US DoD life cycle model.

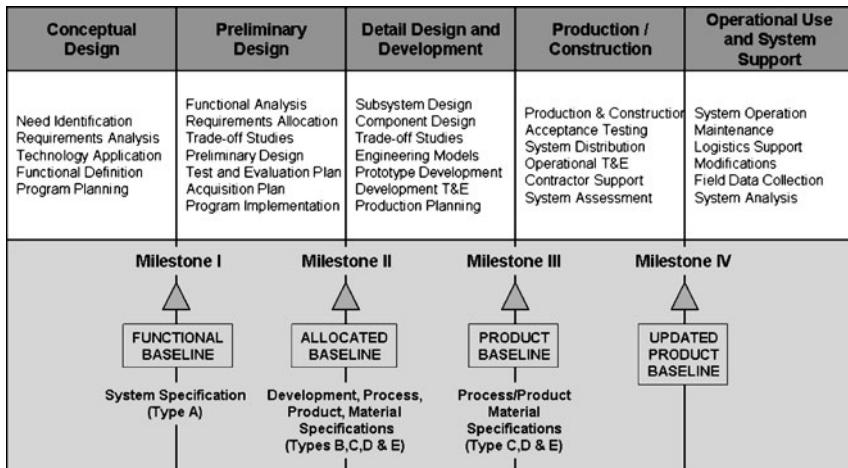


Fig. 5.11 Blanchard and Fabrycky's system acquisition process activities.

full rate production (FRP). After the productio n phase has completed, the military declares Full Operational Capability (FOC).

This life cycle model is also used outside of DoD, albeit with some tailoring. Other life cycle models exist, but many are based on the DoD model. For example, ISO/IEC 15288 (IEEE, 2005) defines five stages: concept, development, production, utilization, and support. It is not difficult to map these five stages into the five stages of the DoD model.

Blanchard and Fabrycky (2006) present an expanded life cycle model using milestones to designate major decision points. Attached with each milestone is a baseline system configuration – placed under formal configuration control with managed changes. Also attached with each milestone are a set of specifications, with their letter designation (e.g., Type A, Type B). Figure 5.11 is a consolidated and synthesized version of the original figure appearing in the reference. No attempt was made to change the meaning of the figure.

We will use Blanchard and Fabrycky's version as an explanation tool to discuss the major activities for each of these five phases.

5.4.1

Conceptual Design (Requirements Analysis)

While the conceptual design phase has many activities, the heart of this stage of system development is the identification, elicitation, analysis, and validation of requirements. Requirements come in many flavors, especially early in the life cycle. Therefore, we need to differentiate some of the terms that are found in the literature and the community. Figure 5.12 is one representation of the relationships between four types of system requirements.

In many cases, the initial set of requirements are articulated at a high level of detail and are usually referred to as needs, to distinguish them from formal requirements documents that will follow. These needs tend to represent an organizational perspective. In other words, this articulation usually represents what the organization wants to accomplish with the new system.

From an initial articulation of the purpose, or objectives, of the system, we typically will see two additional documents developed, although they may be combined into a single document. Regardless of packaging, the operational and functional requirements must be defined. These two sets of requirements represent two additional perspectives: the users and the systems. The operational requirements are worded to describe what the operator or user is able to do with the system. The functional requirements are worded to describe what activities the system can perform. Obviously, there may be overlap among the two categories.

The three requirements documents are then combined with two additional perspectives, the program manager's perspective on the acquisition process and the

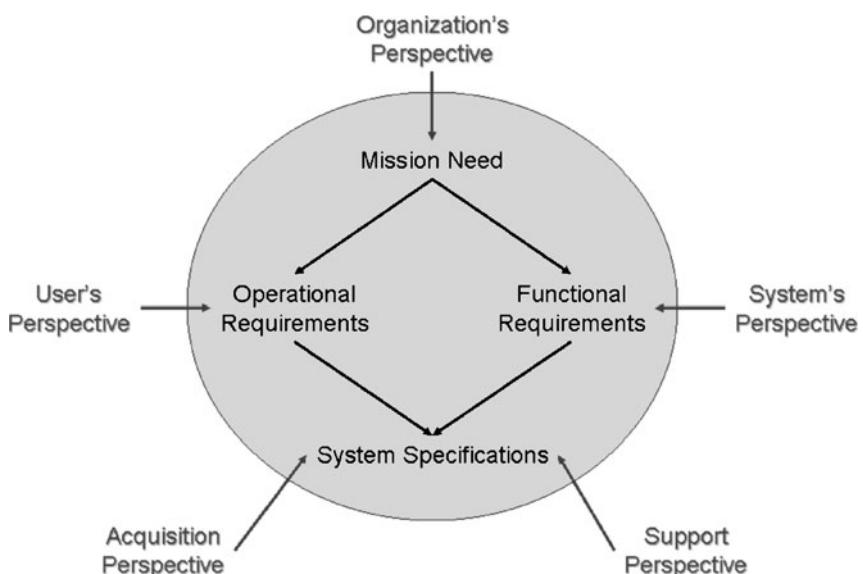


Fig. 5.12 One representation of system requirements.

support manager's perspective on the operations and maintenance process. All of this is then synthesized into a formal system specification. This synthesizing is known as requirements analysis. This specification is a major part of what is known as the functional baseline.

5.4.2

Preliminary Design (Systems Architecting)

With the introduction of the term systems architecting (Rechtin, 1991), many of the activities within the preliminary design phase have been expanded and given their own subdiscipline within systems engineering. At its essence, systems architecting is the process of designing a new system (Maier and Rechtin, 2002). Its primary goals are to define the system from three basic perspectives: an operational perspective, a functional perspective, and a physical perspective. Notice how these three perspectives parallel the distribution and evolution of requirements in Figure 5.16. In fact, system architecting usually begins after a draft of these various requirements is developed. Additionally, systems architecting is used to refine these requirements.

Specifically, embedded within these three perspectives are six views, outlined in Table 5.1 and depicted in Figure 5.13 (Maier and Rechtin, 2002).

Table 5.1 Major system of architectural views

Perspective or view	Description
Purpose/Objective	What the client wants
Form	What the system is
Behavior or functional	What the system does
Performance objectives or requirements	How effectively the system does it
Data	The information retained in the system
Managerial	System construction and management

Defining these six views represents the major activities of systems architecting. Each view may consist of multiple models (e.g., diagrams, figures, charts, matrices). Once the views are drafted, they are checked for consistency and synthesized into a complete system concept description.

5.4.3

Detailed Design and Development

In the third phase of the system life cycle, the focus is shifted from the system as a whole to the subsystems and components that will ultimately be integrated into a workable system. It is within the detailed design phase where many specialty disciplines of engineering and manufacturing come together. This process in which

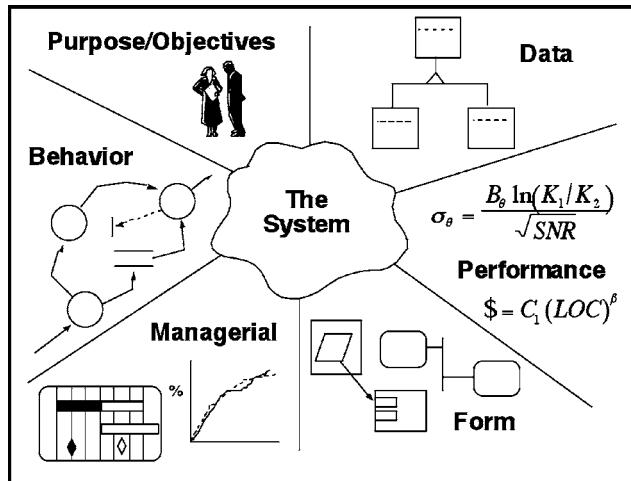


Fig. 5.13 The system as a collection of six views.

Concurrent Engineering					
Activity		Concept Development	Design Development	Design Validation	Production Development
Marketing Product Planning					
Engineering	Feasibility				
	Production Design				
Testing	New Technology				
	Main Program				
Manufacturing	Feasibility/ Tolerancing				
	Tool Studies				
	Tooling				

Fig. 5.14 Concurrent engineering.

all product life cycle activities are concurrently considered in the design phase is known as concurrent engineering and shown in Figure 5.14 (Hartley, 1998). Here, each discipline is brought to the design team early in the process to influence the design.

Within this structure and process, many tools and techniques are applied to developing and verifying the design of the many system components. One major tool used in the design of system components is the trade-off study. Alternative designs, or design approaches, are evaluated using this technique. Many times, developing working prototypes of components, and even the system itself, can be useful in evaluating alternative implementations. Modeling and simulation play a very large role when the cost of developing and producing prototypes is prohibitive, or at least restrictive. Creating virtual prototypes via models and computer simulations allows many alternative concepts to be evaluated before a single design is chosen.

This phase also comprises the creation of plans for production and manufacturing, system deployment, and test and evaluation plans. Most of these plans have

typically been drafted in previous phases; however, they must be finalized and approved before the program can transition to production.

5.4.4

Production and Construction

This phase consists of the actual manufacture of system components, their integration into aggregated subsystems, and testing of the integrated product as a complete system. This is where the rubber meets the road. The components need to meet their specifications and fit together into subsystems, which in turn will fit together into a working system. The integration strategy and plan must be developed before integration starts and is usually drafted in conceptual form during the systems architecting process (when system decomposition occurs).

As components are integrated together, the developer tests the integrated pieces until the entire system is assembled. This final level of testing performed by the developer is generally known as developmental testing or final integration testing.

Once the developer is satisfied with the product, it is provided to the user or customer for what is typically known as acceptance testing. In the military, this activity is known as operational testing, to denote the idea that the customer will validate the system within an operational (versus development) environment using “real-life” scenarios.

Finally, the production and delivery of the system includes many support elements. A complex system will include various support vehicles, parts, maintenance, and tools to operate in an operational environment. For example, the U.S. Air Force’s F-16 Falcon jet fighter includes much more than the aircraft. Figure 5.15 shows the actual aircraft at the top and all of the support equipment necessary to keep one aircraft flying. All of these elements are produced, integrated, tested, and delivered to the customer/user.



Fig. 5.15 F-16 jet fighter support elements.

5.4.5

Operational Use and System Support

The last phase is the longest and most expensive phase in the systems life cycle. This phase includes the actual operation of the system and everything needed to keep the system operational in the evolving environment. Activities such as maintenance, logistics support, and improvement modifications are included over the life span. This phase ends when the system is formally retired from operational service and is disposed of.

5.5

Key Concepts of Systems Engineering

While systems engineering is rich in concepts, ideas, practices, and techniques, we will focus our discussion on four concepts that are key to successfully applying the systems engineering processes to a complex development effort. These simple concepts can sum up a large portion of the discipline.

5.5.1

Integrating Perspectives into the Whole

The first principle of systems, and arguably the most important, is the perspective of the systems engineer. It is the job of the systems engineer to maintain the perspective of the whole system throughout the life cycle process. Very few people will maintain the focus on the whole since their job is typically within a specialty area. Moreover, the program manager will typically focus on the programmatic “whole”, leaving the technical whole to the systems engineer.

Kossiakoff and Sweet (2002) state that “it is the responsibility of the systems engineer to guide the development so that each of the components receives the proper balance of attention and resources, while achieving the capabilities that are optional for the best overall system behavior.” This responsibility represents the technical “honest broker” that must decide between competing demands, often compromising among key system elements.

Ensuring the success of the system over the success of any particular decision or element will require the systems engineer to continually focus on the right balance. Kossiakoff defines balance as “ensuring that no system attribute is allowed to grow at the expense of an equally important or more important attribute.” This truth is magnified when critical attributes are independent, and only through a deep understanding of the system need and its objectives will the systems engineer strike the proper balance between competing views, as illustrated in Figure 5.16.

The goal of the systems engineer is to develop a system that can satisfy the customers needs by applying a disciplined systems engineering process throughout the system development life cycle. Thus the best “balanced” system, among the competing desires and requirements of each of the specialties, is achieved.

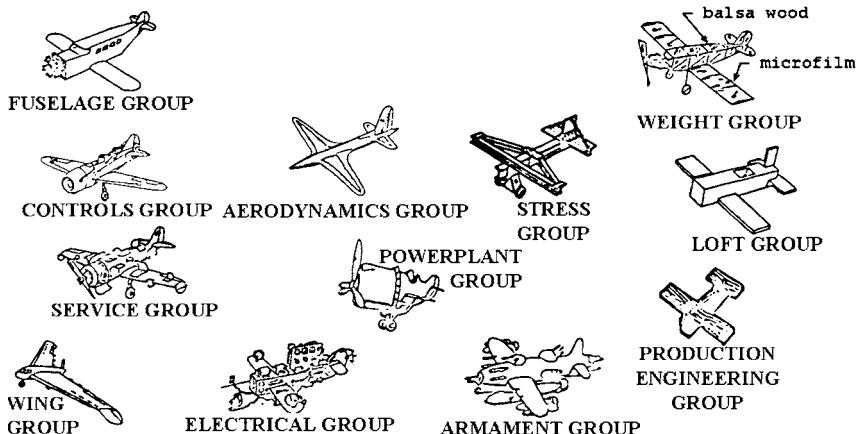


Fig. 5.16 Different perspectives of an aircraft.

5.5.2

Risk Management

This chapter began by stating that systems engineering was largely a transdisciplinary management function with people, knowledge, and processes at its core. As implied in the previous sections, the challenge is to integrate a heterogeneous collection of people and knowledge from diverse disciplines under a single set of processes to accomplish a successful system development. The systems engineer draws upon a combination of technical and people skills to accomplish this goal.

A critical aspect of managing people, knowledge, and processes is the inherent risks associated with developing complex systems. Since developing new systems involves acquiring and applying knowledge about advanced, but not fully developed, components, the process inherently involves risks of several types: technical, programmatic, and operational, to name a few (Kossiakoff and Sweet, 2002).

Risk arises from two broad sources: uncertainty in decision making and the difficulty in integrating multiple disciplines (Sage and Rouse, 1999). Since complexity requires the application of new knowledge without a deterministic understanding of the outcome, we must come to grips with potential adverse outcomes of our development decisions. Additionally, combining different disciplines may lead to anticipated and unanticipated consequences. Therefore, the systems engineer is given responsibility for managing at least a subset of risks (e.g., technical and operational), usually in collaboration with the program manager.

Lowrance (1976) defined risk simply as the combination of probability and severity of adverse effects. This definition is still applicable today, though the methods used in risk management have improved significantly. Risk is typically displayed in a risk matrix, such as the one depicted in Figure 5.17. Multiple ways of defining the levels of likelihood and consequences have been proposed and are available in the literature.

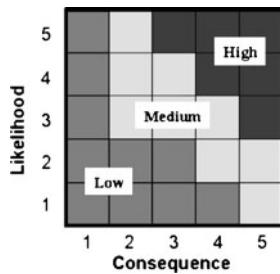


Fig. 5.17 Sample risk matrix.

D. Cooper and Walker (2005) established a general process for managing risks by defining seven steps:

1. Establish the context – develop a structure for the tasks to follow;
2. Identify the risk – determine what might happen to jeopardize the program, and understand how these effects might occur;
3. Analyze the risk – determine the likelihood and consequences of each risk;
4. Evaluate the risk – determine the significance of each risk;
5. Treat the risk – develop a plan to deal with each risk;
6. Monitor and review – continually review the progress made in treating the risks, and repeat steps 2 through 5; and
7. Communicate and consult – collaborate with stakeholders to achieve acceptance.

By exercising this process continually through the system life cycle, the systems engineer can ensure that uncertain or unknown events and effects can be managed.

5.5.3

Decisions and Trade Studies (the Strength of Alternatives)

One of the key concepts that will confront the systems engineer throughout the system life cycle is the art of decision making. Keeping the perspectives of the whole and managing risks will continually generate decision situations. Therefore, the art of decision making is crucial to keeping the development project moving forward toward the ultimate goal of a successful system.

There is a plethora of works on the art (and science) of decision making. A favorite of ours is the work of Clemen and Reilly (2001). However, many other equally admirable books and journal articles exist. The problem is not with the quantity and quality of decision making theories and techniques. The problem lies with placing decision making theory into the context of the formal systems engineering methodology. The answer to this problem lies with what has become known as the trade study (also called trade-off study). We will discuss decisions within this context.

Understanding and making trade-offs among alternatives leading to a decision is not a new concept; formalizing it for the discipline of systems engineering is. In our experience the strength of decisions lies not in the method chosen, but in the set of alternatives explored. Choosing a proper number and type of alternative

solutions becomes the key that leads to useful, informed, and accurate decisions that properly balance the competing demands of the decisions stakeholders. Why is this? In choosing a sufficient set of alternatives, the systems engineer is accomplishing several basic attributes that are inherent in good decision making:

- Exploring the entire trade space, including understanding the performance, technology, and cost boundaries;
- Creating innovative solutions that might not normally be examined;
- Examining varying combinations of system elements and their capabilities that might lead to a superior solution; and
- Identifying the key attributes that are important to the stakeholders.

When a proper set of alternatives is developed for assessment, a formal trade study process will naturally lead to good decisions.

Again, there are several sources of trade study processes; however, Kossiakoff and Sweet (2002) present a particularly simple and effective process that we now describe:

1. Define the objectives of the trade study. Determine the type and number of decisions needed. What is the end-state of this process? What will we know when we are done?
2. Identify the alternatives, as discussed above.
3. Formulate selection criteria to be used in the evaluation. Selection criteria must be capable of differentiating between alternatives, independent of each other, understood by the evaluators, and measurable.
4. Weight the criteria. This is a mechanism by which we can correlate the evaluation criteria with the importance of the objectives.
5. Determine the values of the criteria for each alternative. Many methods for determining these values exist, including measurements, modeling, estimating, simulation, and engineering judgment. Regardless of the method, consistency among the criteria needs to be maintained.
6. Determine the total score for each alternative. Many methods exist that allow one to combine individual criteria values. Some examples are: multiattribute utility theory, Analytical Hierarchy Process, Quality Functional Deployment, mathematical combination, and simulation.
7. Examine and (when possible) eliminate sensitivities. Understand how sensitive the results of your evaluation are to changes in the alternatives, criteria, weights, and combination technique. Adjust appropriately.
8. Select the preferred alternative.

Following a structured approach, such as that of Kossiakoff and Sweet, will lead to a documented, logical, and repeatable decision process.

5.5.4

Modeling and Evaluating the System

As the project moves through the development life cycle (described below), keeping the perspective of the whole requires a method for determining the overall performance of the system vis-a-vis its critical performance parameters and their minimum values (sometimes referred to as threshold values). Early in the system life cycle, when there is not yet a physical system or system components, the systems engineer must be able to estimate the performance of the system concepts (or alternative concepts). As the project moves through the life cycle process, more definitive elements, including actual physical components, will be available for testing and measurement. Ultimately, a full-up system will become available for use to demonstrate actual performance within a variety of test and operational environments.

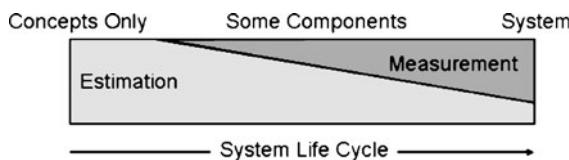


Fig. 5.18 Determining system performance across the system life cycle.

Before we have a working system, though, we must estimate the performance of our concepts and designs. If the environment, the design, and the operational usage are well understood, it may be possible to use empirical data from surrogates. If the physics and mathematics are well understood, closed-form equations may be used to estimate performance. However, with most complex systems, appropriate surrogates are not available, nor is the physics understood sufficiently to develop accurate solutions. It is in these situations that many systems engineers turn to modeling and simulation techniques.

Models and simulations can be developed and used at different levels of fidelity and environments. A common distinction between levels of modeling and analysis is illustrated in Figure 5.19. The term “operational cycle” is dependent on the type of system being developed. For a military system, this might mean a single battle. For a communications system, this might mean a period of time that represents a cyclic sequence of events. Regardless, the level of fidelity increases as one goes from top to bottom of the figure. Determining the level at which to estimate the performance of the system depends on two things: the level of understanding and design of the system (usually tied to the phase of the life cycle) and the objectives of the system.

To illustrate this pyramid, consider an example system, like a military jet fighter. At the subsystem level, its individual subsystems are modeled with great fidelity – the engine, the control surfaces, the weapons systems, the sensors, and the life support systems. The time frame examined would typically be in seconds or minutes. At the single system level, the fighter aircraft as a total system is modeled –

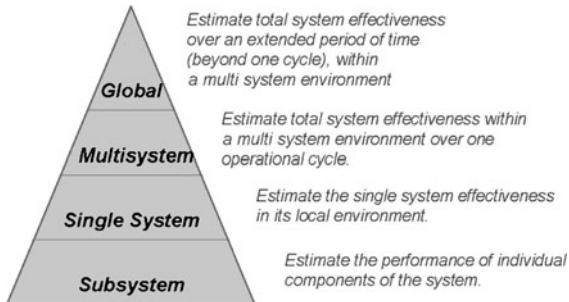


Fig. 5.19 Levels of modeling, simulation, and analysis.

a single jet fighter operating within an operational environment with a mission to accomplish. The six degrees of freedom (x , y , z , yaw, pitch, and roll) could all be represented in great detail. The time frame might be minutes to hours, and the operational cycle at this level could be a single sortie. At the multisystem level, a squadron of jet fighters can be modeled within a single mission environment. Each fighter may not need to be represented in all six degrees of freedom; three degrees may be sufficient for the problem being evaluated. Typically, the time frame would be hours or days, and the operational cycle might be a single maintenance cycle spanning several sorties. Finally, at the global level, a wing of fighters (consisting of multiple squadrons) could be represented, including the maintenance and planning functions in addition to the actual flying missions. The time frame at this level could be weeks or months, and the operational cycle could cover the single deployment of the wing (typically six months).

The definition and division of each level is largely user determined. The point is that modeling and simulation can vary with the scope and extent of the problem examined. For systems engineering, all levels could and typically would be of concern to the customer. Thus, the management team must be ready to perform analysis to determine system performance within all of these contexts.

5.6 Summary

Many books introducing systems engineering start with a description of the system life cycle model and the systems engineering process. This approach provides readers with a very structured context in which to learn systems engineering – and is largely process focused. We decided to focus on the definition and foundations of systems engineering and introduce the reader to the essence of the discipline before launching into the structured life cycle models.

Thus, we divided this chapter into three basic parts. The first part defined systems engineering and described its four foundations: development and use of knowledge, management of people and information, implementation of processes,

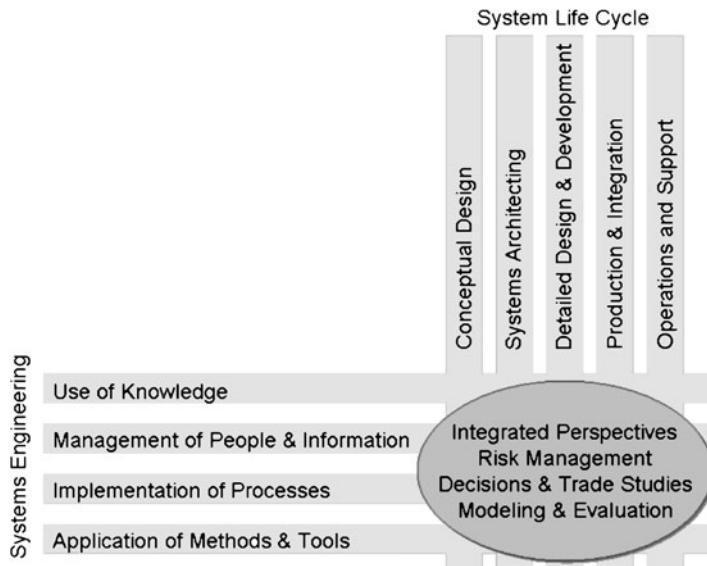


Fig. 5.20 Intersection of systems engineering and the system life cycle

and the application of various methods and tools. These foundations represent the core of what systems engineering is: a transdisciplinary management technology.

The second part defined the system life cycle. It is important to understand the difference between systems engineering (especially its processes) and the system life cycle process. The life cycle is essentially the sequence of development activities defined by the developing organization. It tends to be rigid, because of the complexity of the acquisition process and the oversight needed for large, complex, expensive systems. The last part focused on four crucial concepts within systems engineering: integrating perspectives into a single systems view, managing risk, performing trade studies to support decisions, and using modeling to evaluate system attributes and performance. These concepts were chosen, not to represent the total of systems engineering, but because they exist throughout all aspects of systems engineering (knowledge, people and information, processes, and methods and tools) and all stages and phases of the system life cycle. The interaction of these three parts is depicted in Figure 5.20. Systems engineering intersects the system life cycle, and foundational to this intersection are the four key concepts noted here.

References

- Blanchard, B. and Fabrycky, W. (2006) *Systems Engineering and Analysis*, Prentice Hall, 4th edition, New Jersey.
- Booch, G., Rumbaugh, J. and Jacobson, I. (1999) *The Unified Modeling Language Reference Manual*, Addison-Wesley, New York.
- Clemen, R. and Reilly, T. (2001) *Making Hard Decisions with Decision Tools Suite*, South-Western College, California, updated edition.
- Crespi, V. and Cybenko, G. (2005) *Agent-Based Systems Engineering and Intelligent Vehicles*

- and Road Systems*, DARPA Technical Report AFRL-IF-RS-TR-2005-366.
- Raymond, G., Cooper, D., Grey, S. and Walker, P. (2005) *Project Risk Management Guidelines: Managing Risk in Large Projects and Complex Procurements*, Wiley, New Jersey.
- Hartley, J. (1998) *Concurrent Engineering: Shortening Lead Times, Raising Quality, and Lowering Costs*, Productivity Press, New York.
- IEEE (1999) *IEEE-1220: IEEE Standard for Application and Management of Systems Engineering Processes*, IEEE, New York, NY.
- IEEE (2005) *IEEE 15288, Adoption of ISO/IEC 15288: 2002 Systems Engineering System Life Cycle Processes*, IEEE, New York, NY.
- Jennings, N. (2000) On agent-based software engineering. *Artificial Intelligence*, **117**, 277–296.
- Kossiakoff, A. and Sweet, W. (2002) *Systems Engineering Principles and Practices*, Wiley, New Jersey.
- Law, A. (2006) *Simulation Modeling and Analysis*, 4th edition, McGraw-Hill, New York.
- Lowrance, W. (1976) *Of Acceptable Risk: Science and the Determination of Safety*, W. Kaufmann Publishers, Los Altos, CA.
- Maier, M. and Rechtin, E. (2002) *The Art of Systems Architecting*, 2nd edition, CRC Press, New York.
- Martin, J. (2000) Processes for engineering a system: An overview of the ansi/eia 632 standard and its heritage. *Systems Engineering*, **3** (1), 1–26.
- MIL-STD-499B (1991) *MIL-STD-499B: Engineering Management Standards*, DOD.
- Rechtin, E. (1991) *Systems Architecting*, Prentice Hall, New Jersey.
- Reilly, N. (1993) *Successful Systems Engineering for Engineers and Managers*, Van Nostrand Reinhold, New York.
- Sage, A. (2002) Systems engineering in *Handbook of Dynamic System Modeling*, (ed. P.A. Fishwick), CRC Press, New York, pp. 4.1–4.10.
- Sage, A. (1992) *Systems Engineering*, Wiley, NJ.
- Sage, A. and Biemer, S. (2007) Systems family processes for architecting, design and integration. *IEEE Systems Journal*, **1** (1), 5–16.
- Sage, A. and Rouse, W. (2002) Systems engineering and systems management. in *Standard Handbook of Electronic Engineering*, (eds D. Christiansen, C.K. Alexander and D.K. Jurgen), McGrawHill, New York, pp. 2.1–2.23.
- Sage, A. and Rouse, W. (2009) *Handbook of Systems Engineering and Management*, Wiley, NJ, 2nd edition (in press).
- Wooldridge, M. (1997) *Agent-Based Software Engineering*, Mitsubishi Electric Digital Library Group, London, UK.
- Zachman, J. (1987) *Zachman Architecture Framework*, IBM Technical Report.

6

Quality Assurance of Simulation Studies of Complex Networked Agent Systems

Osman Balci, William F. Ormsby, and Levent Yilmaz

6.1

Introduction

Simulation modeling is a process that entails the development of a model of a system to conduct experiments for the purpose of understanding the behavior of the system and evaluating its alternative operation strategies. Agent simulation, in particular, makes it possible to model complex situations and synthetic worlds whose overall structures emerge from interactions between individuals, that is, to cause structures on the macro level to emerge from models on the micro level, thus breaking the level of barrier in classical modeling (Barsley, 1988; Kaufmann, 1996; Spriet and Vansteenkiste, 1982, 1993). However, unless developed models are demonstrated to be valid and credible, the predictions resulting from simulation experiments will carry the high risk of dissemination of inaccurate knowledge.

Quality is a critically important issue in almost every discipline and is sometimes referred to as Quality is Job 1. Whether we manufacture a product, employ processes, or provide services, quality often becomes a major goal. Achieving that goal is the challenge. Many associations have been established worldwide for quality, for example, American Society for Quality (<http://www.asq.org/>), Australian Organization for Quality (<http://www.aoq.asn.au/>), European Organization for Quality (<http://www.eoq.org/>), and Society for Software Quality (<http://www.ssq.org/>). Manufacturing companies have quality control departments, business and government organizations have Total Quality Management (TQM) programs, and software development companies have Software Quality Assurance (SQA) departments to be able to meet the quality challenge (Balci, 2004).

Modeling and simulation (M&S) of network-centric open agent systems poses significant technical challenges. A network-centric system is a system of systems aligning and integrating other systems and agents such as computers, databases, mobile devices, people (users), processes, satellites, sensors into a globally networked distributed complex system. Characteristics of a network-centric system are described using a layered architecture. Challenges for M&S of network-centric agent systems are presented. The paper focuses on the quality assessment chal-

lenge and advocates the use of a quality model with four perspectives: product, process, project, and people. A hierarchy of quality indicators is presented for network-centric agent system M&S. An approach is described for conducting collaborative assessment of M&S quality using the quality indicators.

Model validation and verification (V&V) plays a key role in mitigating quality risks pertaining to accuracy of simulations. Validation is defined as demonstrating that a computerized model satisfies the simulation objectives and requirements with sufficient accuracy within its domain of applicability (Banks et al., 1987). A model is considered valid under a set of experimental conditions if the models response accuracy is within an acceptable range for its intended purpose. Validation often deals with the question: Are we building the right model? (Balci, 1987). Model verification, on the other hand, deals with building the model right.

M&S applications are mostly made up of software or are software based. Software is inherently complex and very difficult to engineer. Under the current technology, we are incapable of developing a reasonably large and complex software product and guaranteeing its 100% accuracy. Accuracy is considered just one of many quality characteristics of an M&S application and is judged by conducting V&V. As advocated in Balci et al. (2002b), we can increase our confidence in the accuracy of large-scale and complex M&S applications by employing a quality-centered assessment approach.

The purpose of this chapter is to present such a quality-centered assessment approach for large-scale open complex agent systems. Section 6.2 elaborates on the characteristics of such systems. Section 6.3 discusses quality assurance challenges pertaining to the simulation of large-scale open agent systems. Section 6.4 describes a layered architecture to represent such systems. Section 6.5 lists a number of technical challenges for M&S applications of complex agent systems. Section 6.6 presents an M&S quality assessment approach based on four perspectives: product, process, project, and people. Concluding remarks are given in Section 6.7.

6.2

Characteristics of Open Agent Systems

Open systems are systems consisting of components that are heterogenous, autonomous, and dynamic. A critical aspect of such systems is the significance of interaction and the ensuing challenges and complexity due to dynamic patterns of interaction. Agent abstraction provides a useful metaphor to examine characteristics and quality assurance, as agency is primarily based on the notions of autonomy, heterogeneity, and dynamics.

- Autonomy can be defined as the ability of agents to operate without the direct intervention of humans or others and have some degree of control over their actions. However, complete autonomy is likely to result in undesirable consequences in terms of performance and optimal behavior. Hence, constraints on autonomy are imposed using *protocols* and various conflict avoidance, resolution, and management mechanisms.

- Heterogeneity refers to the independence of components within the system. The independence can be reflected in terms of discrepancies in representation, implementation, and data exchange standards between the agent components that constitute the system. Heterogeneity of the information models requires proper mediation and interoperation mechanisms to resolve conflicts and to facilitate seamless exchange and interpretation of data. Some degree of commonality between agents is needed for successful integration and operation. Ontologies are used to define conceptual domain models so that assumptions and obligations of heterogeneous components can be mapped.
- Dynamism corresponds to the capability to dynamically reconfigure and update a system with minimal centralized control. Agents have the capabilities to perceive their environments to adapt and tailor their behavior to improve the performance of the system to attain optimal behavior.
- Communication is a critical notion that enables the interaction of agents within the system. Interaction involves fundamental means of describing the mechanisms that allow agents to exchange data to accomplish their tasks and achieve their goals. Cooperation, for instance, is a general form of interaction studied in the design and implementation of agent systems, and it involves explicit specification of collaboration, coordination, and conflict resolution. Communication is the basis for interactions, and it is expressed in terms of intermediary mediators such as signals or messages. While communication at the abstract level may involve specific linguistic constructs (e.g., speech acts), exchange of data between agents in a complex and open agent system over distributed nodes is ultimately implemented in terms of physical means such as sending data packets over a network.
- Protocols enable the study of communication. In simple terms, a protocol defines when and how an agent may communicate with other agents. Rigid specifications are often preferred in conventional systems to attain desirable results; yet the result is the lack of flexibility and constraints in the autonomy of agents. Hence, new decentralized protocols that minimize the impact on autonomy are emerging in the context of agent systems.

6.3

Issues in the Quality Assurance of Agent Simulations

The characteristics examined above raise specific issues for the quality assessment of agent simulation models. The conceptual components of simulations incorporate networks and processes embedded in the phenomena under study. Metamodeling can be used to define the structural aspects of the organizational network in terms of a knowledge base that defines the rules of interaction among agents that constitute the system. One can also interpret the rule set as the grammar defining the style of an organization. The similarity between the configuration style of the actual organization and its computational counterpart can be established by

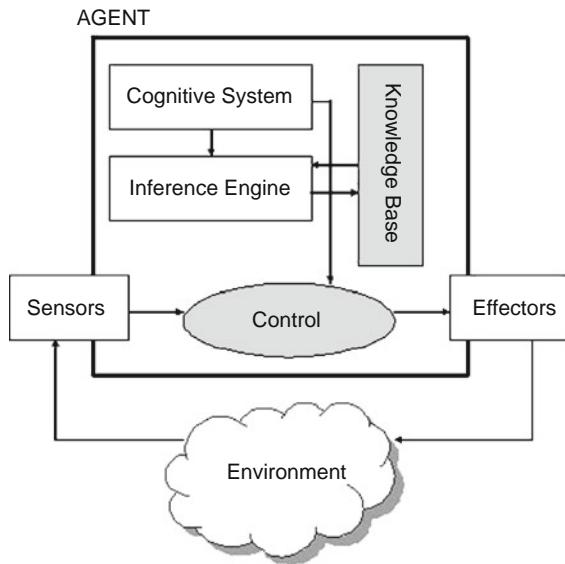


Fig. 6.1 A microarchitecture for agents.

using graph similarity methods. Note, however, the major quality assurance problems associated with such conceptual models is beyond the structural completeness, consistency, and correctness. There is also a lack of methodological support for evaluating agent knowledge bases. Research on the quality assurance of such models tends to be fragmentary in nature and unclear in scope.

Figure 6.1 illustrates the components of a generic agent. The agent communicates with its environment through sensors and effectors often by receiving and sending messages. The control subsystem includes the state-dependent protocol that regulates agent behavior. The protocol represents modes (i.e., states) and associated interaction patterns (i.e., sequences of actions).

The cognitive system that enables perception interacts with the embedded knowledge base and inference engine to constantly monitor and affect the mental state of the agent. The Belief–Desire–Intention (BDI) framework is one such theory that specifies the cognitive behavior of an agent using epistemic logics. Given the above common architecture of cognitive agents, the quality assurance issues can be delineated as follows. First, the knowledge base and inference engine that facilitate the formalization of the relations need to be consistent, coherent, and complete. Second, the theories underlying the personal beliefs, motivations, and commitments need to be implemented by the cognitive subsystem correctly. This requires application of proper evaluation methods and specification methods of cognitive functions to facilitate analysis. The reactive and proactive protocols embedded in the control system need to be tested to determine correct implementation as well. Finally, a verified (e.g., accuracy dimension of quality) multiagent simulation needs to be evaluated against the theory or data. Whereas such architectures may

give rise to interesting phenomena, and sometimes even mimic empirical data, the underlying micro- and macro-level processes may be far from realistic. Hence, establishing the accuracy of simulations requires measuring the extent to which the model accurately characterizes the dynamic processes embedded within the system. Statistical validity measures exist for quantitative model validation, but such metrics often examine the input/output and response surface differences to facilitate drawing inferences with regard to the model's validity. On the other hand, there is a lack of quantitative metric suites that help determine the quality of models across various dimensions. This chapter introduces a strategy to rectify this problem.

6.4

Large-Scale Open Complex Systems – The Network-Centric System Metaphor

A network-centric system (NCS) (also known as network-centric operations) is a system that aligns and integrates other systems and autonomous/heterogeneous agents such as battlefields, communities of interest (COIs), computers, databases, mobile devices, organizational entities, people (users), processes, satellites, sensors, globally networked software, and a distributed complex system. A layered architectural view of an NCS is depicted in Figure 6.2.

The universe of discourse consists of many systems such as those listed in Figure 6.2. Example COIs include Command and Control COI, Finance COI, Intelligence COI, Logistics COI, Personnel COI, and other Institutional and Expedient COIs. Computers include cluster computers, desktops, handhelds, laptops, servers, and supercomputers. The processes include the following: (a) enterprise management processes (e.g., requirements generation process; planning, programming, budgeting, and execution process; acquisition process; readiness process; policy management process; agency management processes; inspector general processes; operational test and evaluation processes) and (b) business processes (e.g., installations and environment, human resources, strategic planning and budget, accounting and finance, logistics, acquisition, technical infrastructure).

The transport layer includes telecommunications networks such as communications, mobile ad hoc networking, multicast networking, wireless networks, wireless sensor networks, and wireline networks (Figure 6.2). One emerging communications technology is Voice over Internet Protocol (VoIP) for transmitting voice, fax, telephone calls, and other information over a data network like the Internet. VoIP enables carrying both voice and data over one line. Worldwide Interoperability for Microwave Access (WiMAX) is another emerging broadband wireless access technology.

The protocol layer consists of many technologies including Common Open Policy Service (COPS), Directory Enabled Networking (DEN), Heterogeneity Aware Peer-to-Peer (P2P), HyperText Transfer Protocol (HTTP), Internet Protocol Security Policy (IPSP), Internet Protocol Version 6 (IPv6), Network Data Management Protocol (NDMP), Simple Object Access Protocol (SOAP), Transmission Control

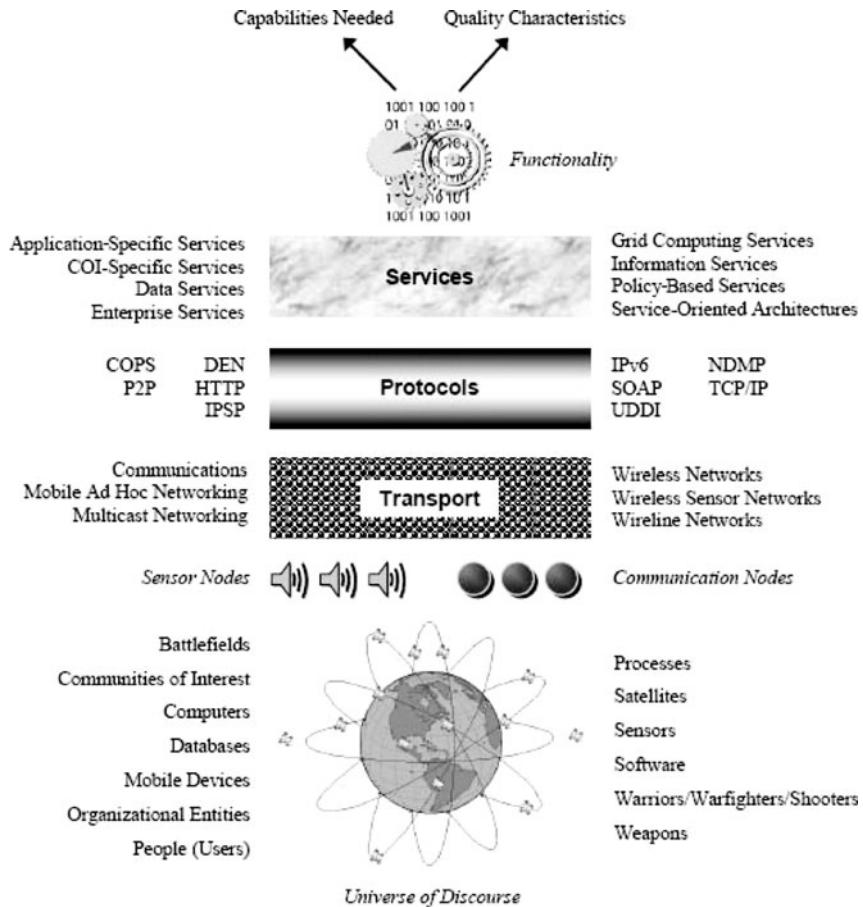


Fig. 6.2 Layered architectural view of a network-centric agent system.

Protocol/Internet Protocol (TCP/IP), and Universal Description, Discovery, and Integration (UDDI). The service layer includes the services listed in Figure 6.2. Application-specific services include tracking, identification, data mining, sensor management, and threat analysis. Enterprise services include messaging (e.g., chatting, instant messaging, e-mail), mediation, registry, discovery, storage, and security. Information services include information assurance, information exchange, information metadata, information modeling, information processing, information security, information transfer, database services, models, pedigrees, and metrics. Policy-based services include Class of Service (CoS), Common Information Model (CIM), Common Open Policy Service (COPS), Directory Enabled Networking (DEN), Policy Core Information Model (PCIM), Quality of Service (QoS), and Service Level Agreement (SLA). Service-Oriented Architectures (SOAs) include Common Object Request Broker Architecture (CORBA), Distributed Com-

ponent Object Model (DCOM), Utility Computing/On-Demand Computing, and web services. The capabilities needed from an NCS include the following:

- Common, consistent knowledge;
- Distributed, collaborative planning and execution;
- Dynamically managed, interoperable, high-capacity connectivity;
- Enterprisewide integrated information;
- Time-sensitive decision making;
- Communications and networking;
 - High-altitude airborne communications relay and router;
 - Multiband, multibeam antennas;
 - Maritime optical communications;
 - Dynamic, seamless, mobile inter-/intranetworking;
 - Cognitive networks;
 - Integrated autonomous network management;
 - End-to-end QoS/CoS-enabled networks.
- Intelligence, surveillance, and reconnaissance (ISR)
 - Automated and autonomous sensor networking and management;
 - Horizon Extension Surveillance System (HESS);
 - Surveillance Warfighter Array of Reconfigurable Modules (SWARM);
 - Automated management and control of intelligence and cryptologic assets.
- Common Operational and Tactical Picture (COTP)
 - Compose, manage, and distribute the COTP;
 - Decision support for dynamic target engagement (using fusion engines and intelligent agents).
- Information assurance
 - Assured authenticity of data and information;
 - Collaboration among multiple security domains;
 - Development and sharing COTP across multiple security domains;
 - Trusted authentication mechanisms.

The capabilities provided are subject to quality assessment by using indicators such as dependability, interoperability, maintainability, scalability, and survivability. The dependability indicator is assessed by availability, reliability, safety, and security.

6.5

M&S Challenges for Large-Scale Open Complex Systems

M&S can be used for open complex systems for many objectives including the following:

1. Guide the military personnel, commanders, subordinate commanders, stakeholders, developers, and others in the acquisition of the system.
2. Evaluate system designs to carry out a specified mission or function against specific criteria.

3. Assess how well a set of capabilities can be provided by a particular design of the system.
4. Evaluate several proposed operating policies or procedures in architecting the system.
5. Predict the effects of emerging new technologies (e.g., VoIP, WiMAX) on system performance.
6. Assess the interoperability of the system elements (e.g., agents).
7. Simulate the input that goes into the system under operational test and evaluation.
8. Facilitate the elicitation of stakeholder needs, identification of new threats, determination of new capabilities needed, and generation of new requirements for the system.
9. Provide significant economic benefits through the reuse of already developed models and simulations.
10. Help reduce the system engineering and integration risks.
11. Conducting an M&S project under any one of the objectives stated above poses an onerous task with many technical challenges, some of which are described below.

Open system architectures are typically represented under different views including the following:

- The operational view represents the tasks and activities, operational elements, and information flows required to accomplish or support a military operation.
- The system view is a description, including graphics, of systems and interconnections providing for or supporting warfighting functions.
- The technical view represents the minimal set of rules governing the arrangement, interaction, and interdependence of system parts or elements, whose purpose is to ensure that a conformant system satisfies a specified set of requirements.
- All views show the scope, purpose, intended users, environment, and analytical findings.

Each view provides a different perspective for analysis. Engineering a simulation model for each view as well as for an integrative view of all views is an extremely complex task. The computational power required for such a simulation model dictates the need to use distributed simulation on a supercomputer with thousands of processors. Incorporating different levels of granularity in model representation poses another serious technical challenge. It is often desirable to simulate system architecture at different levels of granularity, for example, soldier level, tank level, battalion level, or combat level. It is extremely challenging to build a model at a finer level of granularity and to enable model execution at a higher level of granularity. So a large-scale open system is a system of systems. Each system (e.g., a coarse agent) has its own architecture and exists within its own unique problem domain such as combat systems, computer network systems, sensor systems,

weapon systems, and wireless communication systems. Engineering a simulation model of such a system of systems requires technical domain expertise crossing the boundaries of many disciplines. Another technical challenge is substantiating that the simulation model has sufficient quality so that it can be certified for a set of intended uses. The quality assessment challenge is addressed in the next section.

6.6

Quality Assessment of Simulations of Large-Scale Open Systems

Assessing the quality of M&S application open systems poses serious technical and managerial challenges for engineers, analysts, and managers. M&S application quality is the degree to which the M&S application possesses a desired set of characteristics. Quality assessment is situation dependent and the desired set of characteristics changes from one M&S application to another. M&S application quality is not assessed to conclude with a binary decision, where 1 implies perfect quality and 0 implies totally imperfect quality. M&S application quality must be judged as a degree on a scale from 0 to 100. Figure 6.3 shows notional relationships among M&S application quality, utility, and cost during the development life cycle.

While the M&S application is being developed, its quality and utility improve and its development cost rises. The M&S application utility continues to increase as its quality continues to improve, but levels off after a point. The M&S application development costs continue to rise to provide better quality and utility. However, after a point, further development costs do not significantly improve the utility. The intersection point of the utility and cost curves and their shapes are notional and are expected to change from one M&S application development to another. M&S

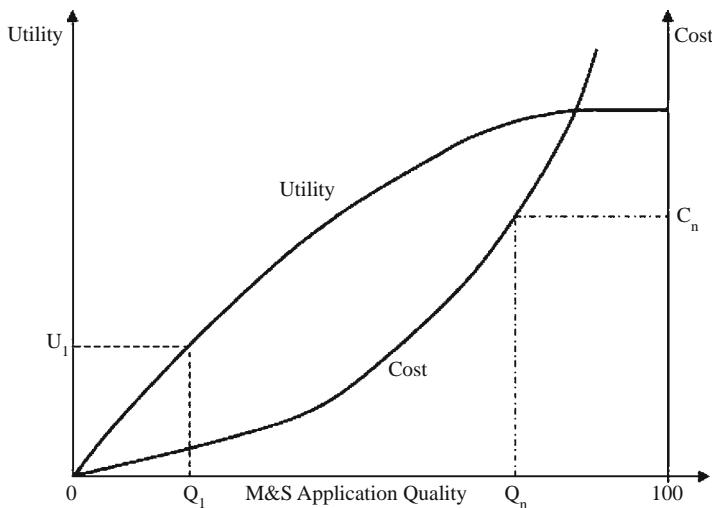


Fig. 6.3 M&S application quality, cost, and utility.

Nominal Score	Numerical Score	Description
Excellent	[90 .. 100]	M&S application exceeds the requirements for a particular quality indicator (e.g., accuracy) under a given set of intended uses.
Very Good	[80 .. 89.99]	M&S application meets all of the requirements for a particular quality indicator under a given set of intended uses.
Satisfactory	[70 .. 79.99]	M&S application satisfies most of the requirements for a particular quality indicator under a given set of intended uses.
Ordinary	[60 .. 69.99]	M&S application meets some of the requirements for a particular quality indicator under a given set of intended uses.
Marginal	[50 .. 59.99]	M&S application fails to meet most of the requirements for a particular quality indicator under a given set of intended uses.
Deficient	[40 .. 49.99]	M&S application is deficient in meeting the requirements for a particular quality indicator under a given set of intended uses.
Unsatisfactory	[25 .. 39.99]	M&S application is unacceptable with respect to a particular quality indicator under a given set of intended uses.
Superficial	[0 .. 24.99]	M&S application is useless with respect to a particular quality indicator under a given set of intended uses.

Fig. 6.4 Nominal score set for M&S application quality.

application quality should be assessed using a nominal score that corresponds to a numerical interval score. An example nominal score set is given in Figure 6.4.

An M&S application can be certified for a particular quality characteristic (indicator) under a given set of intended uses. Example quality indicators include accuracy (which is assessed by conducting verification and validation), interoperability, fidelity, dependability, performance, supportability, and usability. The intended uses must be well defined (Balci and Ormsby, 2000).

The certification decision cannot be made with 100% certainty due to many factors including the complexity of the problem domain, lack of data, reliance on human judgment, and lack of qualified subject matter experts. Therefore, the certification decision should be made with a confidence level similar to that in statistics. In statistics, a $100 \times (1 - \alpha)$ confidence interval is constructed to estimate the unknown population mean of a random variable as $[a, b]$, where α is the significance level, $1 - \alpha$ is the confidence level, a is the lower limit, and b is the upper limit of the interval. For $\alpha = 0.05$, the confidence interval is interpreted as we are 95% confident that the true value of the population mean is contained within the interval $[a, b]$. Similarly, a certification decision should be stated as we are highly confident that the M&S application possesses sufficient accuracy for the prescribed set of intended uses. The confidence level should be stated as a nominal value such as extremely, highly, or slightly. M&S application quality assessment is considered to be a confidence-building activity. The more comprehensive and detailed the assessment is, the more confidently the certification decision can be reached. Four major perspectives or four Ps influence the quality, as depicted in Figure 6.5. The quality assessment can be approached from any one of the four Ps, but a combination of all four will provide the best balance and result in a much higher level of confidence in making a certification decision.

M&S application quality should be assessed hand in hand with the model development by way of assessing a particular development life cycle stage's



Fig. 6.5 Four perspectives of quality.

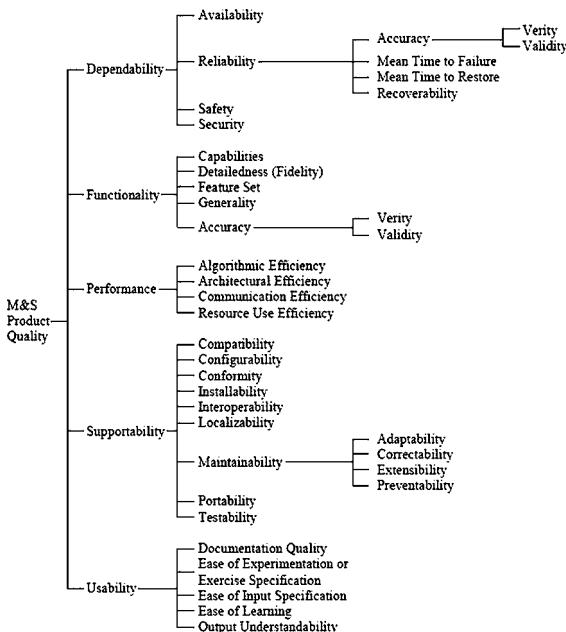


Fig. 6.6 Assessing product quality.

- output work product (or artifact),
- process used in creating the output work product,
- quality of the people employed, and
- project characteristics (e.g., capability maturity, documentation, planning, risk management).

Figure 6.6 presents a hierarchy of indicators for assessing M&S application quality from the product perspective. This is a sample hierarchy showing the higher-level indicators, which should be decomposed further depending on the characteristics of a particular NCMS M&S application (Balci, 2004).

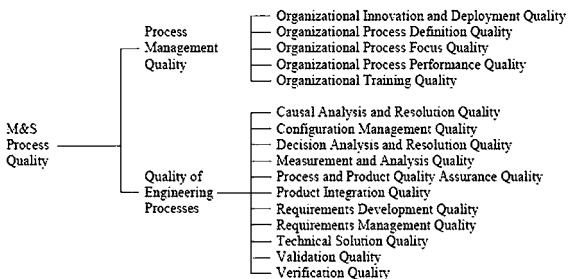


Fig. 6.7 Assessing process quality.

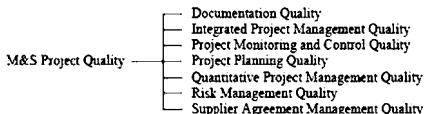


Fig. 6.8 Assessing project quality.

Figure 6.7 presents a hierarchy of indicators for assessing M&S application quality from the process perspective. The indicators chosen represent the process-related areas of the Capability Maturity Model Integration (CMMI) for Systems Engineering and Software Engineering (CMMI-SE/SW) (SEI, 2001b). These indicators need to be further decomposed to provide a desired level of measurement. Other process quality indicators can be added to this hierarchy depending on the characteristics of a particular M&S application.

Figure 6.8 presents a hierarchy of indicators for assessing M&S application quality from the project perspective. The indicators chosen represent the project-related areas of the CMMI-SE/SW (SEI, 2001b). These indicators need to be further decomposed to provide a desired level of measurement. Other project quality indicators can be added to this hierarchy depending on the characteristics of a particular M&S application.

Figure 6.9 presents a hierarchy of indicators for assessing M&S application quality from the people perspective. The indicators chosen represent the process areas of the People Capability Maturity Model (P-CMM) (SEI, 2001a). These indicators need to be further decomposed to provide a desired level of measurement. Other people quality indicators can be added to this hierarchy depending on the characteristics of a particular M&S application.

Collectively, M&S application quality assessment based on the four Ps generates a large number of quantitative and qualitative indicators under a wide and deep hierarchy. Assessment of the indicators mandates subject matter expert (SME) evaluation and requires the integration of disparate evaluations. Planning and managing such measurements and evaluations require a unifying methodology with computer-aided assistance and should not be performed in an ad hoc manner. Such a methodology has been developed by Balci (2001) and computer-aided assistance has been provided by the Evaluation Environment (EE) software system (Orca, 2005).



Fig. 6.9 Assessing people quality.

EE is a web-based client/server distributed software system structured based on the Java 2 Platform, Enterprise Edition (J2EE) industry-standard architecture. It enables collaborative evaluations by engineers, analysts, and SMEs who are geographically dispersed. EE uses 128-bit encrypted Secure Sockets Layer (SSL) technology to provide secure communication between the EE server and the EE user. EE runs under the IBM WebSphere Application Server, IBM DB2 Universal Database, and IBM HTTP Server. It uses open technology standards such as XML, XSLT, SVG, DHTML, and PDF. EE has its own XML markup language called EEML for project data import/export, archive/restore, and automatic report generation in PDF. Based on areas of needed expertise, SMEs are employed for constructing a hierarchy of indicators, relative criticality weighting of the indicators, and assigning scores for the leaf indicators. Considering the functional and non-functional requirements specified for the NCS M&S application, a hierarchy of indicators is created for assessing its quality. Figure 6.10 shows an example EE project with a hierarchy of indicators presented above.

The indicator hierarchy created forms an acyclic graph since an indicator may have more than one parent. The hierarchy should be examined to determine if it possesses sufficient comprehensiveness and depth. Relative criticality weighting of indicators and SMEs are performed by using the Analytic Hierarchy Process (AHP) (Balci et al., 2002a). AHP is commonly used in multicriteria decision making and consists of the following three steps: (1) perform pairwise comparisons, (2) assess consistency of pairwise judgments, and (3) compute the relative weights. SMEs are assigned to assess certain leaf indicators in the hierarchy. Since EE is a web-based software system, SMEs can be geographically dispersed. An SME assigns a nominal score (e.g., good) corresponding to an interval score (e.g., [80, 89]) for each leaf indicator. The leaf indicator scores are automatically aggregated throughout the entire hierarchy. EE automatically generates a report in PDF format. Further details of how to conduct an EE project are provided by Balci et al. (2002a).

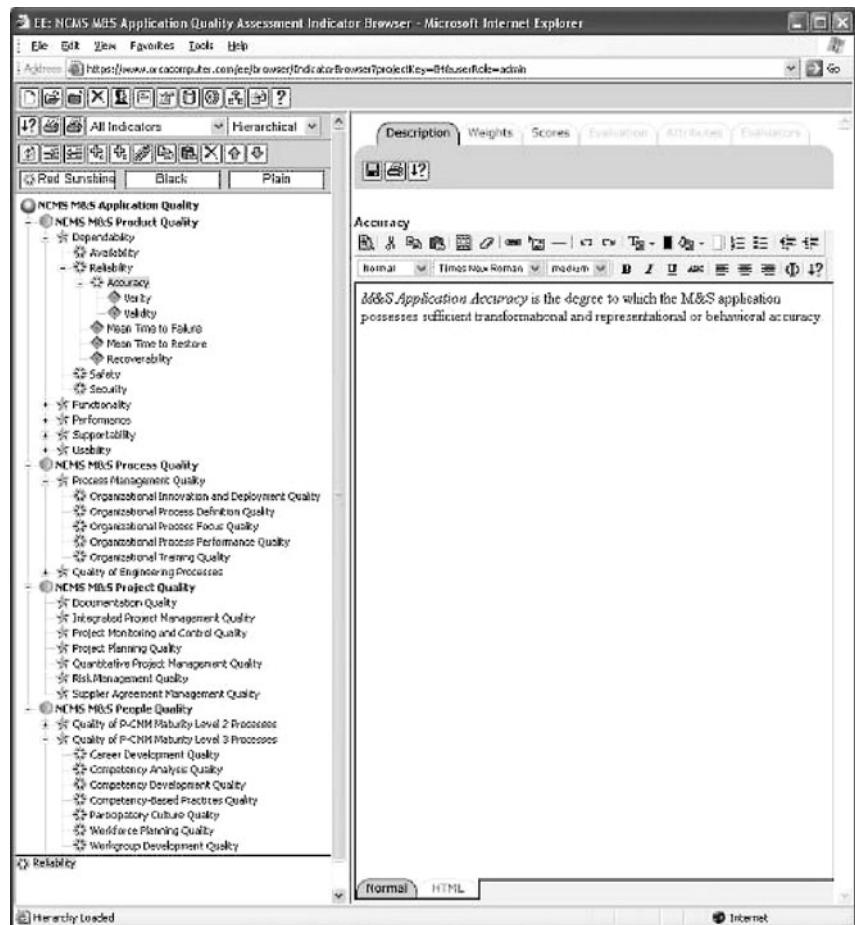


Fig. 6.10 Collaborative assessment of M&S quality.

6.7

Conclusions

Advances in modeling paradigms, methodologies, and technologies have had a considerable impact on how quality assessment is perceived. These advances are partly influenced by the changing nature of the problems addressed by the modeling community. Recently, agent simulation has attracted widespread interest in the computing and systems engineering communities. Development and quality assessment of M&S applications for such complex open system architectures and designs pose significant technical and managerial challenges. This paper outlines a strategy for addressing the quality assessment issue.

References

- Balci, O. (1987) *Credibility and Assessment of Simulation Results: The State of the Art*. Proceedings of the Conference on Methodology and Validation, pp. 6–9.
- Balci, O. (2001) A methodology for certification of modeling and simulation applications. *ACM Transactions on Modeling and Computer Simulation*, 11 (4), 352–377.
- Balci, O. (2004) *Quality Assessment, Verification, and Validation of Modeling and Simulation Applications*. Proceedings of the 2004 Winter Simulation Conference, pp. 122–129.
- Balci, O. and Ormsby, W.F. (2000) *Well-Defined Intended Uses: An Explicit Requirement for Accreditation of Modeling and Simulation Applications*. Proceedings of the 2000 Winter Simulation Conference, pp. 849–854.
- Balci, O., Adams, R.J., Myers, D.S. and Nance, R.E. (2002a) *A Collaborative Evaluation Environment for Credibility Assessment of Modeling and Simulation Applications*. Proceedings of the 2002 Winter Simulation Conference, pp. 214–220.
- Balci, O., Nance, R.E., Arthur, J.D. and Ormsby, W.F. (2002b) *Expanding our Horizons in Verification, Validation and Accreditation Research and Practice*. Proceedings of the 2002 Winter Simulation Conference, pp. 653–663.
- Banks, J., Gerstein, S. and Searles, S.P. (1987) *Modeling Processes, Validation, and Verification of Complex Simulations: A Survey*. Proceedings of the Conference on Methodology and Validation, pp. 13–18.
- Barsley, F.M. (1988) *The Science of Fractal Images*, Springer-Verlag, New York.
- Kaufmann, S. (1996) *At Home in the Universe: The Search for Laws of Self-Organization and Complexity*, Oxford University Press, London, England.
- Orca (2005) Evaluation environment. *Orca Computer – Technical Report*, <https://www.orcacomputer.com/ee/>.
- SEI (2001a) People capability maturity model (p-cmm) version 2.0. Software Engineering Institute – <http://www.sei.cmu.edu/cmm-p/>.
- SEI (2001b) Capability maturity model integration (CMMI) for systems engineering and software engineering (CMMI-SE/SW) version 1.1. Software Engineering Institute – <http://www.sei.cmu.edu/cmmi/>.
- Spriet, J.A. and Vansteenkiste, C.G. (1982) *Computer-Aided Modelling and Simulation*, Academic Press, London, England.
- Spriet, J.A. and Vansteenkiste, C.G. (1993) *Complexity: The Emerging Science at the Edge of Order and Chaos*, Touchstone Books.

7

Failure Avoidance in Agent-directed Simulation: Beyond Conventional v&v and qa

Tuncer I. Ören and Levent Yilmaz

“The significant problems we face cannot be solved at the same level of thinking we were at when we created them.” (Albert Einstein)

7.1

Introduction

Simulation has hundreds of application areas (Ören, 2009a); it is multifaceted (Zeigler, 1984) and, as elaborated in Chapter 1 (Ören, 2009b), can be perceived from several different perspectives such as (1) purpose of use, (2) problem to be solved, (3) connectivity of operations of real and simulation systems, (4) types of knowledge processing, and (5) philosophy of science. According to the purpose of use, two major categories of use of simulation are for (1) performing *experiments* and (2) providing *experience* for entertainment or for three types of training. As a term, simulation also denotes, according to its original nontechnical usage, imitation (fake or pretense) or a representation (i. e., not real, virtual).

7.1.1

The Need for a Fresh Look

In use of simulation to perform experiments, simulation is goal-directed experimentation with dynamic models and is used for several types of decision support, understanding, and for education (Ören, 2007a). Types of usages of simulation for decision support are listed in Chapter 1. In *training*, simulation is the use of a representation of a system to gain/enhance competence through experience under controlled conditions. Training can be achieved (1) to gain/enhance *motor skills* by using virtual equipment as is the case in simulators and virtual simulators (i. e., virtual simulation), (2) to gain/enhance decision making and *communication skills* by gaming simulation (i. e., constructive simulation), or (3) to gain/enhance *operational skills* by a mixture of real system and simulation (i. e., live simulation). With

the advent of agent-based modeling, the synergy of software agents and modeling and simulation (M&S) has gained importance.

Agents are autonomous software modules with perception and social ability to perform goal-directed knowledge processing, over time, on behalf of humans or other agents in software and physical environments. The knowledge processing abilities of agents include reasoning, motivation, planning, and decision making. Additional abilities of agents are needed to make them intelligent, humanlike, and trustworthy. Abilities to make agents intelligent include anticipation, understanding, learning, and communication in natural language. Abilities to make agents more trustworthy as well as assuring the sustainability of agent societies include being rational, responsible, and accountable. These lead to rationality, skillfulness and morality (e.g., ethical agent, moral agent) (Ghasem-Aghaei and Ören, 2003).

Abilities to make agents humanlike include the representation of personality, emotions, and culture. *Agent-directed simulation* is the generic term to denote synergistic use of simulation and software agents. It consists of two types of simulation: simulation for agents and agents for simulation.

The term *agent simulation* denotes simulation of systems modeled as software agents. Sometimes, when the other two possibilities of the synergy of software agents and simulation are not taken into account, the term *agent-based simulation* is also used to denote agent simulation. Similar terminological problems exist in other areas of simulation. For example, the term *computer simulation* denotes both simulation of computer systems as well as simulation where the operations are performed by a computer; for the latter, a proper terminology would be *computerized simulation*. Similarly, the term *Monte Carlo simulation* is used to denote two completely different concepts such as stochastic simulation as well as the use of random variates to solve deterministic problems such as computation of multidimensional integrals.

The second category, that is, *agents for simulation*, consists of two types of simulation: agent-supported simulation and agent-based simulation. Agent-supported simulation is the use of agents as a support facility to provide computer assistance in problem solving or enhancing cognitive capabilities of support functions especially for frontend and backend human–system interfaces. Agent support can be used in conventional simulation as well as in agent simulation. *Agent-based simulation* focuses on the use of agents for the generation of model behavior in a simulation study (ADS, 2008). This possibility of use of software agents dates back the early days when “demons” were used for behavior generation (Javor, 2006). Execution-time model management can be achieved by software agents in an agent-based simulation study.

To date, very little has been published about the problems with agent-based modeling and agent-directed simulation; one exception is Jones (2007). Furthermore, possible problems of the synergy of systems engineering and agent-directed simulation are not yet well documented, even though for complex problems such a syn-

ergy is highly desirable. Thus this chapter has been written to point out additional sources of failures in order to take necessary precautions to minimize associated risks. As we see in later sections, validation and verification (V&V) as well as quality assurance (QA) techniques are used extensively; however, there are cases where a fresh look is needed to avoid failures. We call this paradigm the failure avoidance (FA) paradigm for M&S. The FA paradigm can be applied to (1) conventional M&S, (2) to all three types of agent-directed simulation, (3) to systems engineering, (4) to agent-directed simulation systems engineering, and (5) to agent-directed simulation used for systems engineering.

7.1.2

Basic Terms

Definitions of some basic terms follow:

Failure: An event that does not accomplish its intended purpose.

1. The condition or fact of not achieving the desired end or ends.
2. Nonperformance of what is requested or expected (AHD, 2008).

Mistake:

1. An error or fault resulting from defective judgment, deficient knowledge, or carelessness.
2. A misconception or misunderstanding (AHD, 2008).

Error:

1. An act, assertion, or belief that unintentionally deviates from what is correct, right, or true.
2. The condition of having incorrect or false knowledge.
3. The act or an instance of deviating from an accepted code of behavior.
4. Mistake.
5. *Mathematics* – The difference between a computed or measured value and a true or theoretically correct value (AHD, 2008).

Fault: Something that impairs or detracts from physical perfection; a defect (AHD, 2008).

Defect: A serious functional or structural shortcoming (AHD, 2008).

Deficiency: The quality or condition of being deficient; incompleteness or inadequacy (AHD, 2008).

Flaw: An often small but always fundamental weakness (AHD, 2008).

Shortcoming: A deficiency; a flaw (AHD, 2008).

Sophism: A deliberately invalid argument displaying ingenuity in reasoning in the hope of deceiving someone.

Paralogism: Mistakenly invalid argument: in logic, an invalid argument that is unintentional or that has gone unnoticed.

7.2

What Can Go Wrong

Any failure in the use of M&S would have significant repercussions both in the application area and in the professionalism of M&S. This is due to the increasing importance of M&S.

7.2.1

Increasing Importance of M&S

Simulation continues to be used in more and more sophisticated application areas (Ören, 2009a). Several studies report either trends or normative views for growth areas of M&S (Yilmaz et al., 2008). Within the European Commission, simulation is also used to study innovation as the driving force of an economic development strategy. Simulation is crucial in developing viable solutions in almost all the problems cited in a recent book on the technological and scientific challenges of the 21st century (Bourgeois and Grou, 2007). These challenging technological problems cited include environmental problems, information technology, transportation, energy, health, and space as a new frontier. The proper and reliable use of simulation is imperative.

7.2.2

Contributions of Simulation to Failure Avoidance

Two relationships exist between FA and simulation. Simulation can be used successfully for FA in several fields, and failure should be avoided in simulation. The online Risks Digest of the Committee on Computers and Public Policy of the ACM (Risks, 2008), when searched with the keyword “simulation”, gives references related to simulation and failures (Risks-SIM, 2008). Of particular interest are contributions of simulation to FA and the need for FA in simulation studies. In this section, we focus on contributions of simulation in FA in several fields.

Model V&V is a current topic of great interest to both government and industry. In response to a ban on the production of new strategic weapons and nuclear testing, the Department of Energy (DOE) initiated the Science-Based Stockpile Stewardship Program (SSP). An objective of the SSP is to maintain a high level of confidence in the safety, reliability, and performance of the existing nuclear weapons stockpile in the absence of nuclear testing. This objective has challenged the national laboratories to develop high-confidence tools and methods that can be used to provide credible models needed for stockpile certification via numerical simulation (Schaller, 2004).

Neumann (1993) reports in Risks Digest (vol. 19, issue 91, August 1998, available at <http://catless.ncl.ac.uk/Risks/19.91.html#subj1>):

The Lockheed-Martin Titan IV that began self-destructing at 20000 ft only 40 s after liftoff from Cape Canaveral carried a top-

secret satellite (code-named Vortex) for the US National Reconnaissance Office. It was destroyed on ground command two seconds later. The Air Force gave no information on the cause. This was the last launch for this Titan IV model; future launches are already scheduled to use an improved model. [Source: Reuters item, 13 August 1998; PGN Abstracting]

Only two failures out of 25 launches is reportedly thought to be a reasonably good record, although this loss is expensive – \$ 300 M for the Titan, and between \$ 800 M and \$ 1 B for the satellite. Associated Press noted that a previous Titan IV failure occurred from Vandenberg AFB in August 1993 (ignoring the Titan IV that blew up on the test stand on 1 April 1991 – see [(RISKS-12.09)] – as a result of a problem that seemingly could have been caught in simulation).

T. Lima reports in Risks Digest (vol. 24, issue 70, 29 May 2007; available at <http://catless.ncl.ac.uk/Risks/24.70.html>) based on the following story (also available on the Strategy Page: <http://www.strategypage.com/htm/w/htmurph/articles/20070529.aspx>):

Last month [April 2007], a Swedish Gripen fighter crashed when the pilot suddenly ejected. The pilot insisted that he had not activated the ejection system. After intense investigation, and lots of computer simulation of flight systems, investigators concluded that the pilots [sic] account of events was accurate. Turns out that if enough g-force is applied to the aircraft, the pilot ejection system automatically activates. This leaves the aircraft without a pilot, right after it has performed a stressful maneuver (to produce the high g-force). This sort of thing is increasingly common with modern weapons systems. That [sic] because these systems are increasingly more complex systems of systems, where it has become impossible to forecast all of the possible unpleasant, and unwanted, events that could occur under certain situations.

T. Panton reports in Risks Digest (vol. 24, issue 75, 25 July 2007; available at <http://catless.ncl.ac.uk/Risks/24.75.html>):

I'm a fan of well designed simulations. In a former life I was involved in the testing of a control system for a chemical plant. We created a faithful simulation of the plant, then arranged for our simulator to output voltages that mimicked the sensors that were in the real plant. We then plugged these outputs into the control system and went through a series of tests. The results were totally unexpected. It failed, in some cases the simulated plant responded too slowly. We assumed that the problem was the simulation or the interfaces. After much study we concluded it wasn't. The control system was at fault, and in a subtle way, the control blocks covering the most time critical loops had been spread over multiple processors and the inter-processor communication was introducing a significant delay. The manufacturer 're-optimized' the loops and the problem was fixed. Used appropriately simulations (or stimulations ?) can tell you things you couldn't easily find any other way, so should be in the toolbox of any serious tester.

7.2.3

Need for Failure Avoidance in Simulation Studies

Some serious failures of M&S were reported by Neumann (1993). The following are from ACM's online forum on risks.

S. Malloy reports in Risks Digest (vol. 7, issue 65, 15 October 1988; available at: <http://catless.ncl.ac.uk/Risks/7.65.html#subj2>):

In fact, one of the games designed by Simulations Publications, Inc. (SPI) before they were bought out by TSR was ordered by the Army. Firefight was intended as a simulation for warfare in Europe, to teach tactics to infantry and armor commanders. Within a number of simplifying abstractions, it modeled the weapons systems available to a unit commander in Germany. SPI later made this game available as part of their regular line. It soon became apparent that the game was not only useful for teaching tactics, it was also a device to build confidence and improve morale – the way the rules and weapons systems data were set up, it was almost impossible for a Soviet player to pull anything better than a draw out of the game. The game mechanics were biased so that an American player could win by using the 'right' tactics ('right' in the Army sense – the approved Army tactics for a given situation), rather than encouraging the players to come up with their own tactics.

L. Fernandes reports in Risks Digest (vol. 18, issue 66, 12 December 1996; available at: <http://catless.ncl.ac.uk/Risks/18.66.html#subj5>):

The January 1997 issue of "Dr. Dobbs Journal" has an article in which the author reports that his software simulation of the automatic baggage handling system of the Denver airport mimicked the real-life situation. In his conclusion, he notes that the consultants did perform a similar simulation and had recommended against the installation of the system currently in place. The city, however, overruled the consultant's report (the contractors who were building the system never did see the report) and gave the go-ahead.

Similarly, simulation studies done during the summer of 2004 – a year before the Hurricane Katrina disaster in 2005 in New Orleans – was reported to be ignored by the decision makers (Longley).

R.A. Zeh reports in Risks Digest (vol. 23, issue 60, 27 November 2004; available at: <http://catless.ncl.ac.uk/Risks/23.60.html>):

I live in River Forest, IL, a Chicago suburb. The November 10th, 2004 edition of our local paper, The Wednesday Journal, contained coverage of a development review board meeting for some new construction. The architect for the construction had done a computerized "Shadow Study" to determine how the new construction would impact the area. The study simulated conditions on Jun 20 and Dec 20. Here is the section relevant to RISKS: When Nimesh

said that long shadows were present at 6 p.m. in the 20 Dec simulation, DRB chairman Frank Martin look [sic] at him for a second, then said “It’s dark at 6 p.m. in December”. “The software’s not perfect,” replied Nimesh after a moment’s hesitation. I would suggest that the user needs a little work too. Checking your answers for reasonableness is always a good practice, even if you aren’t using a computer.

NASA’s DART spacecraft smashes into satellite (April 2005) – P.G. Neumann (based on other sources) reported in Risks Digest (vol. 24, issue 29, 26 May 2006; available at: <http://catless.ncl.ac.uk/Risks/24.29.html>):

NASA’s 800-pound Demonstration for Autonomous Rendezvous Technology (DART) spacecraft was supposed to circle a defunct orbiting Pentagon satellite. A report released on 15 May 2006 indicates that DART moved to within 300 ft of the satellite 472 mi above Earth, and then lost control – crashing into the satellite. The report says the collision was based on faulty navigational data from the main sensor that caused DART to believe that it was backing away from its target rather than approaching.

P.B. Ladkin reported in Risks Digest (vol. 24, issue 32, 14 June 2006, available at: <http://catless.ncl.ac.uk/Risks/24.32.html>):

... The bias in the Surrey GPS receiver was known, but the software fix for it had never been implemented by the DART team, and the bias was not reflected in the software model simulating the GPS receiver in preflight testing, so this simulation failed to elicit the reset problem, says the report (Morning, p37–8).

Examples of negative consequences of simulation if not used properly (in training and education) [from Ören (2002a)] are:

- Training to enhance motor and operational skills (and associated decision making skills)
- Ill-prepared operators (civilian as well as military) for regular operating conditions;
- Ill-prepared operators (civilian as well as military) for rare emerging conditions;
- Recommending unfit personnel for jobs requiring high dexterity;
- False sense of achievement.
- Training to enhance decision making and operational skills
- Ill-prepared decision makers (civilian as well as military) for regular operating conditions;
- Ill-prepared decision makers (civilian as well as military) for rare emerging conditions;
- Dehumanization of decisions.
- Education
 - Missed opportunity to better learn the subject matter;
 - Misinformation.

Examples of negative consequences of simulation if not used properly (in areas other than training and education) [from Ören (2002a)]

- Evaluating alternative courses of action
 - Missed opportunity to gain insight into the subject matter;
 - Insufficient or incorrect advice;
 - Interpretation of results influenced by desired (political) outcome;
 - Models used beyond their scope of applicability;
 - Not enough evidence to evaluate results;
 - Acquisition;
 - Acquisition of equipment/services not fully fit for the purpose.
- Engineering design
 - Equipment malfunctions;
 - Unreliable structures;
 - Neglect of engineering knowledge and regulations;
 - Numerical inaccuracies;
 - Incomplete models.
- Prototyping
 - Recall of (hundreds of) thousands of defective units;
 - Deficient representation of novel technologies.
- Diagnosis
 - False alarms;
 - Inability to detect “faint” alarms;
 - Proof of concept;
 - Burden to future generations (for example, in simulation of safe disposal of nuclear fuel waste for tens of thousands of years);
 - Unwarranted extrapolation of present knowledge.
- Understanding
 - Missed opportunity to have proper understanding;
 - M&S used as “proof of concept” rather than as “investigation of concept”.

A framework based on the above examples to be developed to have more comprehensive categories and more details for each category may be very useful.

7.2.4

Some Sources of Failure in M&S

Several simulationists have pointed out common mistakes in M&S studies (Jain, 1991; Huang, 1994; Sadiku and Ilyas, 1994; Menasce, 2002; Brodsky, 2003; Claypool, 2004; Jasperneite, 2008). Göktürk (2008) provides a tripartite analysis of common mistakes in simulation under the following headings: common mistakes in modeling, in conducting the experiment, and in project management. Since computerization is an important phase, a group to enumerate common mistakes in computerization is well worth having. A list of common mistakes in user expectations also deserves a separate category. The following nonexhaustive five-part list is

representative of the common mistakes in M&S projects and includes also views of the cited authors.

- Common mistakes in modeling include:
 - Incorrect assumptions may be used;
 - Inappropriate level of detail;
 - Ignoring important parameters;
 - Improper use of units or their conversions (when units are changed, the values should also be converted);
 - Wrong choice of criteria to be used in decomposing the system;
 - Invalid models (use of models not representing reality correctly with respect to the goal of the study);
 - Lack of adequate validation of the model.
- Common mistakes in experimentation include:
 - Improperly handled initial conditions;
 - Transient phase not taken into consideration;
 - Runtime not long enough;
 - Improper use of random number distributions;
 - Poor random number generators (safer to use a well-known generator);
 - Improper selection of the seeds of the random number generators.
- Common mistakes in computerization include:
 - Unverified models (computerization errors exist);
 - Inappropriate language (e. g., use of programming language instead of specification language);
 - Inappropriate behavior generation algorithms (e. g., integration of a stiff system with a regular integration algorithm);
 - Random number generators not reliable.
- Common mistakes in project management include:
 - Incomplete mix of essential skills (needed: project leader, M&S specialists, software engineers, domain expertise);
 - Study goal not specified clearly;
 - Lack of understanding of the problem (project team members as well as users need to have a common understanding of the study's goal);
 - Time estimates for different phases of the project are not accurate;
 - Cost estimates for different phases of the project are not accurate;
 - Inappropriate documentation;
 - Inability to manage the development of a large complex computer program.
- Common mistakes in the expectations of users include:
 - Inadequate level of user participation;
 - Simulation-based learning must be “fun” or “gamelike” to be effective. (Great content is the foundation of a great learning system. Great simulation-based design will never compensate for nor overcome poor content.) E-learning simulation training is best for addressing certain training objectives; other training methods may be better suited for other objectives. For that reason, it is often best to use a blended learning model

that uses the best mix of methods for the training task at hand (Brodsky, 2003);

- The results are generated by computer; hence they would be correct. There is no need for reasonableness checks of the results;
- Ignoring the results of simulation study.

Deliberations and elaborations on the list and taking the necessary precautions may be very helpful in the FA of M&S projects. Some of the necessary actions may fall under the V&V activities, some others in QA activities, and the rest of the activities may be useful to avoid failures of M&S projects that might have been completely certified from V&V and QA perspectives. Often common failures are elaborated from the point of view of conventional M&S. Within the scope of this book, possible failures regarding agent-directed simulation and systems engineering need also to be elaborated. This is done in sections titled “Failure Avoidance for Agent-Directed Simulation” and “Failure Avoidance for Systems Engineering”. Majone and Quade (1980) provide an in-depth analysis of pitfalls of systems analysis studies. Several pitfalls are also applicable for simulation studies. Hall (1977) covers errors in experimentation. Over 150 types of errors are listed in Table 7.1.

7.3

Assessment for M&S

Three groups of concepts are involved in an assessment paradigm: types of assessment, criteria for assessment, and elements of M&S to be assessed. This section is based on Ören (1984), where over 900 types of M&S-related assessments are reported. The elements of the last two groups do not need to be orthogonal (Sheng et al., 1993). An element, such as the goal of the study, can appear in both groups of elements of M&S to be assessed as well as criteria for assessment since it can be assessed with respect to some criteria and can be used as a criterion in the assessment of some other elements.

7.3.1

Types of Assessment

Assessment studies fall under two major categories, that is, descriptive and normative. Descriptive assessments include syntactic, morphological, and semantic assessment. Normative assessments include pragmatic and ethical assessments.

The *descriptive assessment* of an element of a simulation study is its evaluation with respect to the value-free rules used to represent it. It consists of syntactical, morphological, and semantic assessments. The syntactic assessment of an element of a simulation study is its evaluation with respect to the rules of representing the symbols or the expressions constructed from these symbols. The morphological assessment of an element of a simulation study is its evaluation with respect to the norms to represent relevant forms or structures. Both problem-dependent

Table 7.1 A list of (over 150) types of errors.

absolute error	ethical error	programming error
acceptance error	experimental error	projection error
accidental error	experimentation error	propagated error
accumulation error	extrapolation error	proportional error
acknowledge error	fatal error	quadratic error
algorithm error	fixed error	random error
algorithmic error	fractional error	read error
ambiguity error	frequency error	reasoning error
analysis error	gain error	rejection error
angular error	global error	relative error
approximation error	global integration error	representation error
ascertainment error	global relative error	requirement error
assumption error	hardware error	residual error
attribution error	heuristic error	resolution error
balance error	human error	rounding error
balanced error	hypothesis error	round-off error
bearing error	inadvertent error	sampling error
bias error	inherited error	scientific error
biased error	input quantization	error semantic error
bit error	inscription error	sensitivity error
calculation error	instrument error	sensor error
calibration error	instrumentation error	sequence error
chaotic error	integration error	simplification error
classification error	interpolation error	simulation error
clerical error	irrecoverable error	single error
computational error	judgment error	software design error
computer error	language error	software error
computerization error	linearization error	solution error
conceptual error	loading error	specification error
consistency error	local error	stable error
constraint error	local integration error	standard error
convergence error	logical error	static error
copying error	machine error	substitution error
correlated error	measurement error	syntactic error
cultural bias error	method error	syntactical error
cumulative error	model error	syntax error
damping error	modeling error	systematic error
data error	moral error	transcription error
decision error	nonsampling error	transmission error
deductive error	observation error	truncation error
definition error	observational error	type I error
design error	offset error	type II error
detected error	omission error	type III error
diagnostic error	overestimation error	typical error
digitization error	parameter error	unacknowledged error
discretization error	parameterization error	unbiased error
disk error	parity error	uncorrelated error
dumping error	perception error	unification error
dynamic error	persistent error	usage error
environment error	phenomenological error	user error
estimation error	program error	

and methodology-based elements can be subject to the scrutiny of morphological assessment. The semantic assessment of an element of a simulation study is its evaluation with respect to the meaning attached to it.

The *normative assessment* of an element of a simulation study is its evaluation with respect to some norms of a value system that can be pragmatic or ethical. The pragmatic assessment of an element of a simulation study is its evaluation with respect to practical results such as implementability, usability, clarity, comprehensibility, sensitivity, or cost effectiveness. An elaborate list can be found in Balci *et al.* (Chapter 7). In product quality, for example, dependability, functionality, performance, supportability, usability, and their subgroups are listed.

The *ethical assessment* of an element of a simulation study is its evaluation with respect to a set of moral codes. A code of professional ethics for simulationists of the Society for Modeling and Simulation International (SCS) (Ören *et al.*, 2002) has already been adopted by several M&S groups including the NATO Modeling and Simulation Group and the Modeling and Simulation Professional Certification Commission (M&SPCC, 2008). The rationale for a code of professional ethics for simulationists is also developed by Ören (2002b). Ethical assessment is an important aspect of FA in M&S.

7.3.2

Criteria for Assessment

Criteria which can be used for the assessment of several elements of a simulation study are listed in Table 7.2.

7.3.3

Elements of M&S to be Studied

The basic elements of a simulation study that may be assessed for acceptability, rejection, or in some cases for improvement are listed in Table 7.3.

7.4

Need for Multiparadigm Approach for Successful M&S Projects

New paradigms can replace or enrich the previous paradigm(s). In the case of M&S projects where some paradigms are necessary but not sufficient, V&V, QA and FA paradigms together provide much richer and more powerful possibilities to ensure quality and to avoid failures. For this reason, a multiparadigm approach is desirable for successful M&S projects. In the quantification of success, we should also take into consideration that if the metric is wrong, the precision of measurement is irrelevant.

Table 7.2 Criteria for assessment.

Ethical considerations
Pragmatic considerations
Problem solving (paradigm, time, cost)
Goal of the study
Resources (available, to be procured, developed)
Real system <ul style="list-style-type: none"> • (existing, to be engineered) • (structure, behavior)
Technical system specification
Model <ul style="list-style-type: none"> • parametric model • parameter values (point values, vector values, distribution functions) (range of acceptable values)
Experimental conditions
Simulation <ul style="list-style-type: none"> • (run, study) • (transition period, steady-state period)
Computer (time, cost)
Computer program (time, cost) for <ul style="list-style-type: none"> • problem specification • translation/compilation • maintenance • execution
Norms of <ul style="list-style-type: none"> • simulation methodology • modeling methodology • design of experiments • agent technology • software engineering • systems engineering

7.4.1

V&V Paradigm for Successful M&S Projects

The definitions of validation and verification – adopted from AIAA (2008) – are as follows:

Validation: The process of determining the degree to which a model is an accurate representation of the real world from the perspective of the intended uses of the model. Verification: The process of determining that a model implementation accurately represents the developer's conceptual description of the model and the solution to the model (Schaller, 2004).

Table 7.3 Elements of M&S to be assessed.

People
<ul style="list-style-type: none"> • system analyst/simulationist • programmer
Problem solving paradigm
<ul style="list-style-type: none"> • decision process
Background knowledge
<ul style="list-style-type: none"> • problem (or domain) dependent knowledge • methodology-based knowledge <ul style="list-style-type: none"> • knowledge of simulation modeling experimentation (design of experiments) data analysis result analysis • knowledge of support areas
Goal of the study
Restrictions (and their trade-offs)
Experimental conditions
Real system (its environment, structure, and behavior)
Model (its environment, structure, and behavior)
Model behavior
<ul style="list-style-type: none"> • generation • processing (collection, compression, display, interpretation)
Computer
Computer environment/language

A popular short expression is:

Validation often deals with the question Are we building the right model? Model verification, on the other hand, deals with building the model right. (Balci, 1987).

The vital importance of validation and verification has been recognized from the very early days of M&S. The Technical Committee on Model Credibility of SCS published the first list of terminology for model credibility in 1979 (Schlesinger, 1979). A recent document states, “The development of guidelines and procedures for conducting a model V&V program are currently being defined by a broad spectrum of researchers. This report reviews the concepts involved in such a program” (Schaller, 2004).

The following selection from the report is important:

Verification and validation are processes that collect evidence of a model’s correctness or accuracy for a specific scenario; thus, V&V cannot prove that a model is correct and accurate for all possible conditions and applications, but, rather, it can provide evi-

dence that a model is sufficiently accurate. Therefore, the V&V process is completed when sufficiency is reached (Schaller, 2004).

Hence, a simulation model certified to be valid and verified for a study may not hold these attributes for another study. Bibliographies of V&V have been prepared by Balci and Sargent (1984) and by Oberkampf (1998). The archives of Foundations 2002 (Foundations, 2002) contains recent articles and rich references on V&V. Pace (2004) reviewed the seven V&V challenge areas for M&S identified by Foundations '02 and has presented the author's impression of progress during the 2 years since the workshop. These areas are: (1) qualitative assessment, (2) use of formal assessment processes, (3) inference, (5) coping with adaptation, (6) aggregation, and (7) human involvement/representation. Among other valuable contributions, Scholten (2008) has a good review and coverage of references on V&V from a broad problem-solving perspective. Yilmaz (2006) elaborates on validation and verification of social processes within agent-based computational organization models.

7.4.2

QA Paradigm for Successful M&S Projects

The chapter, in this book, written by Balci et al. (2009) and titled "Quality Assurance of Simulation Studies of Complex Networked Agent Systems" covers in detail the QA paradigm for M&S. Without mentioning a paradigm shift, Balci et al. (2009) expose the fact that while V&V is important, it represents only a limited aspect of the quality risk in M&S; hence they justify additional need for a QA paradigm.

Model Validation & Verification plays a key role in mitigating quality risks pertaining to accuracy of simulations. Validation is defined as demonstrating that a computerized model satisfies the simulation objectives and requirements with sufficient accuracy within its domain of applicability. A model is considered valid under a set of experimental conditions, if the model's response accuracy is within acceptable range for its intended purpose. Validation often deals with the question Are we building the right model? Model verification, on the other hand, deals with building the model right.

Balci underlined the importance of simulation QA in other articles as well. For example, in 1998 he wrote:

To increase significantly the probability of success in conducting a simulation study, an organization must have a department or group called simulation quality assurance (SQA). The SQA group is responsible for total quality management and works closely with the simulation project managers in planning, preparing, and administering quality assurance activities throughout the simulation study. The SQA is a managerial approach that is critically essential for the success of a simulation study. Ören (1981) presents concepts, criteria, and paradigms that can be used in establishing an SQA program within an organization Balci (1998).

Several SQA terms are defined by Balci (2004). Different authors have concentrated on the QA paradigm both in conventional M&S and for the synergy of artificial intelligence and M&S. For example, Ören (1986, 1987) elaborated on QA paradigms for artificial intelligence in M&S. In 2008, a newsletter, titled “Simulation Quality Assurance”, was introduced for performance simulation of buildings (aecsimqa, 2008). The quality principles presented for the ergonomics of human-computer interfaces of M&S software by Ören and Yilmaz (2005) can be tailored for agent-based computer assistance in the realization of human-computer interfaces in agent-supported simulation. Balci et al. (2009) introduce the four Ps (i. e., product, process, people, and project) approach to simulation QA and provide in-depth and useful information. Focusing on the process of M&S may also lead to built-in quality.

7.4.3

Failure Avoidance Paradigm for Successful M&S Projects

In this section, we focus on FA in simulation. The maturity of simulation is achieved through developments, improvements, and, especially, through several paradigm shifts. As presented by Thomas Kuhn in his seminal book *The Structure of Scientific Revolutions*, three stages are necessary for a paradigm shift to occur. In the first stage, a paradigm becomes dominant. In the second stage, limitations or problems with the dominant paradigm are observed or better anticipated. In the third stage, a new paradigm that would surpass these limitations and problems is proposed and after some time lag becomes the new dominant paradigm.

The V&V paradigm has been the dominant paradigm for the success of M&S projects. However, as justifiably shown by Balci et al. (2009), the V&V paradigm is a subset of the QA paradigm; meaning that the V&V paradigm is necessary but not sufficient for the success of M&S projects. The QA paradigm, as presented by Balci et al. (2009), is indeed very comprehensive in scope. However, an agent-directed simulation project can still fail (Ören, 2007b). To be able to see and thereafter avoid these possible failures, conception of the problem from another perspective, namely a paradigm shift, is needed. This new paradigm to avoid several categories of failures in agent-directed simulation and its synergy with systems engineering can be called the FA paradigm. In the synergy of agent-directed simulation and systems engineering there are two major categories of sources of failure. They are sources of failure inherent in software agents and sources of failure inherent in systems engineering. Sources of failure for any type of simulation study are listed in Table 7.4. Terms related to sources of failure are given in Table 7.5.

7.4.4

Lessons Learned and Best Practices for Successful M&S Projects

It is well documented that people learn better from their mistakes. However, those who cannot learn from the mistakes (and even from the positive experiences) of others are doomed to repeat those mistakes. It is much more clever to learn from

Table 7.4 Sources of failure of M&S.

In M&S, sources of failures can be:

Project management
 Goal of the study
 Instrumentation
 Data collection
 Assumptions (explicit and/or implicit) in specifications of models, experiments (scenarios, experimental frames), and (model, experimental frame) pairs
 Modeling (conceptual models)
 Scenarios (experimental conditions) (realism and applicability of scenarios, consistency of joint scenarios in federations and federations of federations)
 Design of experiments
 Experimentation (behavior generation)
 Computerization of (models, experiments, runtime libraries and infrastructure)
 Computation (numerical computing, soft computing)
 Logic (fallacies in logic (paralogisms, sophisms))
 Artificial intelligence (rule-based expert systems, software agents (trustworthy agents, moral agents))
 M&S infrastructure (including runtime facilities)
 Documentation (inconsistent, erroneous, non existing)
 Communication (between stakeholders)
 Recommendations of the simulation study
 Not or late implementing the recommendations

Table 7.5 Terms related to sources of failure.

blunder	fault	mistake
corruption	flaw	malfunction
defect	fraud	omission
deficiency	illusion	paralogism
deviation	imperfection	pitfall
error	incorrect	risk
failure	misconception	sophism
fallacy	misfunction	wrong

the mistakes of others and benefit from their positive experiences. For this reason, lessons learned and best practices are very valuable. An early and still totally valid normative view on how to conduct simulation studies is titled the “Don’ts of Mathematical Modeling” (Golomb, 1970). Several publications promote lessons learned in different aspects of M&S studies. For example, Hofmann (2004) presents lessons learned about criteria for decomposing systems into components in military M&S. Harper and Zacharias (2004) report lessons learned in integrating cognitive models in large-scale simulation environments. Zabek (2007) writes about lessons learned from the design and execution of a federation for joint experimentation.

Collection and analyses of lessons learned studies in V&V, QA, and FA may be useful in benefiting from the shared experiences of many simulationists. For ex-

ample, “Verification and Validation without Independence: A Recipe for Failure” by Arthur and Nance (2000) is very important for pointing out an important factor. Another detailed study reports lessons learned from bank loan simulations (BLS, 2008). Considering the banking crisis in 2008, the value of using appropriate criteria becomes very important.

7.5

Failure Avoidance for Agent-Based Modeling

Very recently, studies on FA of some aspects of agent-directed simulation studies have started to be reported. They focus mainly on the V&V aspect, as was the case with conventional M&S studies. In the years to come, we may expect to have substantial research and publications in this field. Some of these publications are listed below:

2005:

- Shillingford et al. (2005) – Verification and validation of an agent-based simulation model;
- Xiang et al. (2005) – Verification and validation of agent-based scientific simulation models.

2006:

- Kennedy (2006) – Verification and validation of agent-based and equation-based simulations and bioinformatics computing;
- Yilmaz (2006) – Validation and verification of social processes within agent-based computational organization models.

2007:

- Ören (2007b) – Reliability in agent-directed simulation;
- Tolk (2007) – Constraints for V&V of agent based simulation: First results – A system-of-systems engineering perspective.

2008:

- Moya et al. (2008) – Visualization and rule validation in human-behavior representation;
- Schoenharl and Madey (2008) – Evaluation of measurement techniques for the validation of agent-based simulations against streaming data.

7.5.1

Failure Avoidance in Rule-Based Systems

Reasoning can be implemented as rule-based systems for agents. However, since the early days of artificial intelligence, the reliability of expert systems has been an issue (Hollnagel, 1989) and rule-based systems may involve several categories of logical errors (Gupta, 1995; O'Leary and O'Keefe, 1989; Stachowitz and Chang, 2008).

In addition to the validation and verification issues (Balci, 2004) as well as QA issues of conventional simulation (Balci et al., 2009), logical errors of rule-based systems need to be analyzed in all three types of agent-directed simulation. Hollnagel (1989) classifies expert systems failures as follows:

- Overconfidence in the inference engine;
- Incompleteness of symptom set, of conclusion set, or of rules;
- Erroneous knowledge (mistake, transcription errors, or biases);
- Misleading priorities;
- Misleading dialog.

Logical errors that may occur in rule-based systems can be related to consistency or completeness. Some types of consistency errors are listed below:

- Inconsistent rules
 - Inconsistency under generalization (ISA), synonymy, referential identity, incompatibility;
 - Indirect inconsistency;
- Redundant rules (superfluous rules);
- Conflicting rules;
- Subsumed rules (duplication);
- Circular rules;
- Recursive rules and indirect recursion;
- Ambiguous rules;
- Irrelevant rules;
- Conflicting rules (conflict under synonymy, referential, incompatibility, indirect conflict).

Some types of completeness errors are listed below:

- Missing rule (e.g., missed cases for safety-critical systems unreliable rule base);
- Unreferenced attributes;
- Illegal attributes;
- Unreachable (facts, conclusions);
- Dead end (literals, conclusions).

Some examples of erroneous inference rules are given in Table 7.6.

Table 7.6 Examples of erroneous inference rules.

Type of erroneous	Rules examples
Ambiguous rules	if a and b then x if a and b then y
Contradictory rules (conflicting rules)	Contradictory antecedent if a then x if c then x Contradictory consequent if a then x if a then y
Dead-end rules	(Unreachable conclusions) if a then x (where x is not a terminal condition and there are no antecedents containing x)
Inconsistent rules	if a then x if a then $\neg x$
Redundant rules	if a and b then x if b and a then x Another example if a and b then x and y if a and b then x (redundant)
Rules with incompatible premises	if a and b then x (where a and b are incompatible)
Rules with irrelevant literals	if a and b then x if a and $\neg b$ then c Equivalent rule is: if a then x
Rules with unreferenced attributes	if a and b then c (where b is not used)

7.5.2

Failure Avoidance in Autonomous Systems

An interpretation of agent autonomy is: “agents operate without the direct intervention of humans or others, and have some kind of control over their actions and internal state” (Castelfranchi, 1995). Agents must interact with other agents, and this brings another dimension to their autonomy: “agents can act autonomously to some degree, and they are part of a community in which mutual influence occurs” (Hayes, 1999). Based on their interactions with each other, agents must act in such a way as to contribute to the achievement of the overall mission.

A fundamental reliability issue in agent simulation is trust and ethical behavior. Based on the interaction of agents with humans as well as with each other, the assurance of their rationality and ethical behavior is very important. Along this line, several studies exist on moral agents (Ören, 2002c,d). “Cooperation is becoming an

important paradigm for both civilian and military applications. Holonic systems are excellent candidates to conceive, model, control, and manage dynamically organizing cooperative systems. A holonic system is composed of autonomous entities (called holons) that can deliberately reduce their autonomy, when the need arises, to collectively achieve a goal. A holonic agent is a multiagent system where each agent (called a holon) acts with deliberately reduced autonomy to assure harmony in its cooperation in order to collectively achieve a common goal" (Ören, 2001a).

7.5.3

Failure Avoidance in Agents with Personality, Emotions, and Cultural Background

Agents with personality and/or agents with emotions or cultural representations need to have linguistic descriptive variables that necessitate fuzzy variables. As studied by Hollnagel (1989), it is necessary to tackle the problem that "Inference engines are not very well adapted to the treatment of uncertainty. One reason is that inference in rule-based systems follows the *modus ponens* style (affirming the antecedent, realized by forward chaining), but a more fundamental reason is that inference engines process rules separately so that, in the presence of uncertainty, rules play the role of uncertain sources of information".

Another fundamental problem in using agents to model human behavior is human nature itself. Bonabeau elaborates on this issue:

A human organization is often subject to operational risk. Consider financial institutions. Operational risk arises from the potential that inadequate information systems, operational problems, breaches in internal controls, fraud, or unforeseen catastrophes will result in unexpected losses. According to the Basle [sic] Committee on Banking, operational risk involves breakdowns in internal controls and corporate governance that can lead to financial losses through error, fraud, or failure to perform in a timely manner or cause the interests of the bank to be compromised in some other way, for example, by its dealers, lending officers, or other staff exceeding their authority or conducting business in an unethical or risky manner. It is increasingly viewed as the most important risk that banks face (Bonabeau, 2008).

Hence, by not taking into consideration some aspects of human nature, a simulation study, whether conventional or agent-directed, may not predict some aspect of human involvement. However, within the framework of FA, these issues are very natural to take into account. As posited by Bonabeau:

In other words, ABM (agent-based modeling) is not only a simulation tool; it is a naturally structured repository for self-assessment and ideas for redesigning the organization. . . . ABM is perfect not just for operational risk in financial institution but for modeling risk in general. Modeling risk in an organization using ABM is THE right approach to modeling risk because most often risk is a property of the actors in the organization: risk events impact people's activities, not processes (Bonabeau, 2008).

Table 7.7 Externally generated inputs.

Mode of input	Type of input
Passive acceptance of exogenous input (imposed or forced input)	<p>Type of access to input: coupling, argument passing, knowledge in a common area, message passing.</p> <p>Nature of input:</p> <ul style="list-style-type: none"> • Data (facts) • Forced Events • Sensation (converted sensory data: from analog to digital; single or multisensor: sensor fusion) • External goals (imposed goals)
Active perception of exogenous input (perceived input)	<ul style="list-style-type: none"> • Perception (interpreted, sensory data, detected events, and perceived goals) <ul style="list-style-type: none"> • includes: decoding, selection (filtering), recognition, regulation • Evaluated inputs <ul style="list-style-type: none"> • evaluation of inputs (acceptability) • evaluation of source of inputs (reliability, credibility)

7.5.4

Failure Avoidance in Inputs

Inputs can be generated either externally or internally (Ören, 2001b).

7.5.4.1 Externally Generated Inputs

Externally generated inputs (i.e., exogenous inputs) can be two types: passively accepted or actively perceived inputs. *Passively accepted exogenous inputs* are imposed or forced inputs. As outlined in Table 7.7, they include data, facts, external events, converted sensory data (or sensations), and external (or imposed) goals.

Even though software agents are widely used, possibilities are still open for advancements in sensation and perception. Senses for agents do not need to be limited to human senses. Often, agent perception means an awareness of things through the physical senses, especially sight. However, many other possibilities exist for advanced agents. Table 7.8 summarizes some types of sensing/perception for software agents. The table can easily be enlarged by considering many other types of sensors in several types of applications (Sohraby et al., 2005).

Sensors are related to transducers. A transducer converts an input energy into another type of energy or provides information about the input energy. With this second functionality, transducers are sensors that can also be used as measurement instruments. However, sensors can also be used as input devices for software agents.

Table 7.8 Types of sensations.

Type of stimulus	Type of perception
light	<ul style="list-style-type: none"> vision (visual perception): visible light vision, ultraviolet vision, infrared vision microspectrophotometry
sound	hearing (auditory sensing): audible/infrasonic/ultrasonic sound (medical ultrasonography, fathometry)
chemical	(gas sensing/detection): smell (smoke/CO ₂ /humidity sensor) (solid or fluid sensing): taste, microanalysis
heat	heat sensing
magnetism	magnetism sensing: geomagnetism/thermomagnetism sensing, electrical field sensing
touch	sensing surface characteristics
motion	acceleration sensing
vibration	vibration sensing: seismic sensor

Some types of measurement errors related to sensors are (adopted from Wikipedia sensor):

1. The sensor is insensitive to the measured property.
2. The sensor is sensitive to any other property.
3. The sensor influences the measured property.

Some other types of errors are offset error or bias (“if the output signal is not zero when the measured property is zero”), hysteresis, approximation error (digitization error), and systematic error (or random error).

Active perceptions of exogenous inputs (Table 7.7) include perception (interpreted, sensory data, detected events, and perceived goals), and evaluated inputs. Sensation and perception can be differentiated as follows: Perception is interpreted sensory data and detected events and necessitates, in addition to passive sensation, selection (filtering), recognition, regulation, and interpretation.

Some additional sources of errors for perception are decoding, selection (filtering), recognition, regulation, and interpretation errors. Perception is widely accepted to be an essential characteristic of software agents. For example, Wooldridge and Jennings (1995) state: “An agent is specified in terms of two components: perception and action”.

7.5.4.2 Internally Generated Inputs

Internally generated inputs (i. e., endogenous inputs) are summarized in Table 7.9. They can be two types: actively perceived endogenous inputs and deliberate endogenous inputs. The active perception of endogenous inputs is based on introspection and consists of perceived internal facts, events, or the realization of their lack. Ac-

Table 7.9 Internally generated (i.e., endogenous) inputs.

Mode of input	Type of input
Active perception of endogenous input	Introspection (perceived internal facts, events; or realization of lack of them)
Deliberate endogenous input	<ul style="list-style-type: none"> • Anticipated facts and/or events (anticipatory systems) • Internally generated questions • Internally generated hypotheses by: <ul style="list-style-type: none"> • Expectation-driven reasoning (Forward reasoning) (Bottom-up reasoning) (Data-driven reasoning) • Model-driven reasoning • Internal goals (<i>internally generated goals</i>)

tive deliberate endogenous inputs consist of anticipated facts and/or events and internally generated questions, hypotheses, and goals.

7.6

Failure Avoidance for Systems Engineering

Systems engineering (SE) involves the application of engineering and management practices to transform user needs into a system specification and realization that most efficiently meets the need. It provides a very important approach to the study of complicated and complex systems. The synergy of SE and M&S is already an accepted approach to the simulation studies of large and complex systems (MITRE, 2001; NDIA-M&S, 2008; Hollenbach, 2003).

The synergy of SE and agent-directed simulation is also starting to be recognized as an important area (Ören and Yilmaz, 2006; Yilmaz and Ören, 2007). In these two previous articles we also provided several references. In an earlier article, the technical foundations for QA of SE activities for safety assessment (of nuclear fuel waste simulation) were also presented (Ören and Elzas, 1987). In this book, the focus is on the synergy of SE and agent-directed simulation; hence FA in SE is another important aspect for this synergy along with the FA for agent-based modeling.

Wymore (1999) elaborates on system-theoretic portents of predisposition to system failure. He clarifies each of the seven phases of the system life cycle, that is, (1) requirements development, (2) system design, (3) detailed design, (4) system development, (5) system test, (6) operations, and (7) retirement/replacement. Understanding them may be useful to avoid failures in SE in general. Tailoring them to the contribution of SE to simulation projects may be useful in avoiding fail-

ures in this particular area. Scholten, in his broad-based dissertation, focuses and contributes on better modeling practices from an ontological perspective on multidisciplinary, model-based problem solving (Scholten, 2008). Recently a graduate course was developed in a Canadian university on QA for SE (Bentehar, 2008).

7.7 Conclusion

Simulation, agent-directed simulation, and its synergy with SE, similar to many other advanced and sophisticated artifacts, are prone to several categories of failure. Hence, to get the full benefits of simulation, without undesirable error-prone side effects, one needs to consider the sources of failure in M&S and have them under control. For this reason, V&V and more comprehensively QA studies are done in M&S. However, FA for agent-directed simulation as well as for SE should also be considered, especially in advanced M&S systems benefiting from the synergy of agent-directed simulation and systems engineering.

References

- aecsimqa. Simulation Quality Assurance Newsletter (for building performance simulation). <http://www.aecsimqa.net/en/>. Last date of access: 6 July 2009.
- ADS. Agent-directed Simulation Symposium. <http://www.eng.auburn.edu/SCS-TM/ADS2008.htm>. Last date of access: 6 July 2009.
- AIAA. American Institute of Aeronautics and Astronautics. Guide for the Verification and Validation of Computational Fluid Dynamics Simulations. *Technical Report, AIAA-G-077-1998*, Reston, VA, 1998.
- J.D. Arthur and R.E. Nance. Verification and Validation without Independence: A Recipe for Failure. *Proceedings of the 2000 Winter Simulation Conference*, pp. 859–865, 2000.
- O. Balci and R.G. Sargent. A Bibliography on the Credibility Assessment and Validation of Simulation and Mathematical Models. *Simuletter*, vol. 15, no. 3, pp. 15–27, 1984.
- O. Balci. Credibility and Assessment of Simulation Results: The State of the Art. *Proceedings of the Conference on Methodology and Validation*, pp. 6–9, 1987.
- O. Balci. Verification, Validation, and Testing. In J. Banks, Editor, *Handbook of Simulation*, pp. 122–129. Engineering & Management Press, 1998.
- O. Balci. Quality Assessment, Verification, and Validation of Modeling and Simulation Applications. *Proceedings of the 2004 Winter Simulation Conference*, pp. 122–129. 2004.
- O. Balci, W.F. Ormsby, and L. Yilmaz. Quality Assurance of Simulation Studies of Complex Networked Agent Systems. In Yilmaz, L. and T.I. Ören (eds.) *Agent-Directed Simulation and Systems Engineering*, Wiley Series in Systems Engineering and Management, Wiley, Berlin, 2009.
- J. Bentehar. Quality Assurance for Systems Engineering (Graduate Course INSE 6280, syllabus). <http://users.encs.concordia.ca/bentehar/inse6280.html>, Concordia University, Concordia Institute for Information Systems Engineering. Last date of access: 6 July 2009.
- BLS. Bank Loan Simulation Lessons Learned. <http://www.rateset.com/Services/BLSlessons.html>. 2008.
- E. Bonabeau. Agent-based Modeling: Methods and Techniques for Simulating Human Systems. *PNAS (Proceedings of the National Academy of Sciences of the USA)*, vol. 99, no. 3, pp. 7280–7287. 2002

- P. Bourgeois and P. Grou (eds.) *Les Grands Défis Technologiques et Scientifiques au XXIe siècle*, Ellipses, Paris, France. 2007
- M.W. Brodsky. *Making Sense of Self-Service E-Learning Simulations: Fun or Functional?*? (First appeared in the January 2003 edition of E-Learning Magazine Online and is reprinted with their permission). 2003.
- C. Castelfranchi. Guarantees for autonomy in cognitive agent architecture. In Wooldridge, M. and Jennings, N.R., (eds), *Intelligent Agents: Theories, Architectures, and Languages*, (LNAI Volume 890), pp. 56–70. Springer-Verlag: Heidelberg, Germany. 1995.
- M. Claypool. *Lecture Notes: Modeling and Performance Evaluation of Network and Computer Systems*. <http://web.cs.wpi.edu/~claypool/courses/533-S04/slides/simulation.pdf>. 2004.
- Foundations 02. Foundations 2002 Workshop on Verification and Validation of Models and Simulations. Archive Site: <http://www.cs.clemson.edu/~found04/Foundations02/>, 2002.
- N. Ghasem-Aghaei and T.I. Ören. Towards Fuzzy Agents with Dynamic Personality for Human Behavior Simulation. *Proceedings of the 2003 Summer Computer Simulation Conference*, pp. 3–10, Montreal, PQ, Canada, July 20–24, 2003.
- S.W. Golomb. Don't's of Mathematical Modeling. *Simulation*, vol. 14, no. 4 (April). pp. 197–198. 1970.
- E. Göktürk. Engineering Simulation and Emulation Systems. *Lecture Notes*. <http://www.identra.com.tr/en/university-relations/08Fall-BoUn-CMPE58L-simulation-and-emulation-engineering/slidesets/week02.pdf>, 2008.
- U. Gupta. *Validating and Verifying Knowledge-Based Systems*, IEEE Computer Society Press. 1995.
- C.W. Hall. *Errors in Experimentation*, Matrix Publishers, Champaign, Illinois. 1977.
- K.A. Harper and G.L. Zacharias. Common Problems and Helpful Hints to Solve them: Lessons Learned in Integrating Cognitive Models in Large-scale Simulation Environments. *Proceedings of the 2004 Winter Simulation Conference*, pp. 891–897. 2004.
- C.C. Hayes. Agents in a Nutshell – A Very Brief Introduction. *IEEE Transactions on Knowledge and Data Engineering*, vol. 11, no. 1 (Jan/Feb), pp. 127–132. 1999.
- M.A. Hofmann. Criteria for Decomposing Systems Into Components in Modeling and Simulation: Lessons Learned with Military Simulations. *Simulation*, vol. 80, no. 7–8, pp. 357–365. 2004.
- J. Hollenbach. *M&S Committee: Mission, Background and Objectives*. Technical Report – NDIA Systems Engineering Division. 2003.
- E. Hollnagel. *The Reliability of Expert Systems*. Ellis Horwood Ltd., Chichester, UK. Halstead press: a division of John-Wiley & Sons, New York, Chichester, Brisbane, Toronto. 1989.
- B.K. Huang. *Computer Simulation Analysis of Biological and Agricultural Systems*. CRC Press. 1994.
- <http://education.yahoo.com/reference/dictionary/entry/tool>. Last date of access: 6 July 2009.
- R. Jain. *The Art of Computer Systems Performance Analysis*. Wiley, New York. 1991.
- R. Jain. *Introduction to Simulation – Lecture Notes*. <http://www.raj Jain.com/cse567-06/ftp/k24its.pdf>. 2004.
- J. Jasperneite. *Lecture notes on: Performance Evaluation of Communication Networks using Simulation Techniques – Introduction*. <http://www.raj Jain.com/cse567-06/ftp/k24its.pdf>. 2008.
- A. Javor. Model Identification Using Intelligent Agents. In: A. Javor, (ed.) *Studies in Simulation Undertaken at the McLeod Institute of Simulation Sciences Hungarian Center*. BME Department of Information and Knowledge Management. Budapest University of technology and Economics, Budapest, Hungary 2006.
- G.T. Jones. Agent-based Modeling: Use with Necessary Caution. *American Journal of Public Health*, vol. 96, no. 11 (Nov.), pp. 2055–2060. 2007.
- R.C. Kennedy. Verification and Validation of Agent-based and Equation-based Simulations and Bioinformatics Computing: Identifying Transposable Elements in the Aedes Aegypti Genome. *Masters Thesis*. <http://www.cse.nd.edu/~nom/Papers/RyanCKennedythesisfinal.pdf>. 2006. Last date of access: 6 July 2009.
- R. Longley. *FEMA's Pam Simulation Foretold Katrina Disaster – Preparedness action plans*

- not implemented in time.* <http://usgovinfo.about.com/od/defenseandsecurity/a/femapam.htm>. 2008. Last date of access: 6 July 2009.
- G. Majone and J. Quade. *Pitfalls of Analysis*. John Wiley and Sons, Chichester, England. 1980.
- D.A. Menasce. *Performance Modeling – Part III – Simulation*. Lecture Notes: <http://www.cs.gmu.edu/~menasce/cs700/files/PerformanceModeling-Simulation.pdf>. 2002. Last date of access: 6 July 2009.
- MITRE. Distributed Simulation Systems Engineering: How to Play War. <http://www.mitre.org/news/digest/archives/2001/howtoplaywar.html>. 2001. Last date of access: 6 July 2009.
- L.J. Moya, F.D. MacKenzie, and Q.A.H. Nguyen. Visualization and rule validation in human-behavior representation. *Simulation & Gaming*, vol. 39, no. 1, pp. 101–117. 2008.
- M&SPCC. The Modeling and Simulation Professional Certification Commission. <http://www.simprofessional.org/>. Last date of access: 6 July 2009.
- NDIA-M&S Com. Modelling and Simulation Committee of the Systems Engineering Division of NDIA (National Defence Industrial Association). http://www.ndia.org/Divisions/Divisions/SystemEngineering/Pages/Modeling_and_Simulation_Committee.aspx. Last date of access: 6 July 2009.
- P.G. Neumann. Modeling and Simulation. *Communications of the ACM*, vol. 36, no. 4, pp. 124. 1993.
- W.L. Oberkampf. Bibliography for Verification and Validation in Computational Simulation. *Sandia Report SAND98-2041*. <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.30.3359>. 1998. Last date of access: 6 July 2009.
- D.O'Leary and R. O'Keefe. *Verifying and Validating Expert Systems. Tutorial:MP4*. IJCAI-89. 1989.
- T.I. Ören. *Concepts and Criteria to Assess Acceptability of Simulation Studies: A Frame of Reference*. CACM, 24 (4), 180–189. 1981.
- T.I. Ören. Quality Assurance in Modelling and Simulation: A Taxonomy. *Simulation and Model-Based Methodologies: An Integrative View*, T.I. Ören, B.P. Zeigler, M.S. Elzas (eds.). Springer-Verlag, Heidelberg, Germany, pp. 477–517. 1984.
- T.I. Ören. Artificial Intelligence in Quality Assurance of Simulation Studies. *Modelling and Simulation Methodology in the Artificial Intelligence Era*, M.S. Elzas, T.I. Ören, B.P. Zeigler (eds.), North-Holland, Amsterdam, pp. 267–278. 1986.
- T.I. Ören. Quality Assurance Paradigms for Artificial Intelligence in Modelling and Simulation. *Simulation*, vol. 48, no. 4 (April), pp. 149–151. 1987a.
- T.I. Ören and M.S. Elzas. Technical Foundations for Quality Assurance of Systems Engineering Activities for Safety Assessment. *Waste Management '87*, R.G. Post (ed.), *Proceedings of the Waste Management Conference*, Tucson, Arizona, pp. 275–281. March 1–5, 1987b.
- T.I. Ören. Advances in Computer and Information Sciences: From Abacus to Holonic Agents. *Special Issue on Artificial Intelligence of Elektrik (Turkish Journal of Electrical Engineering and Computer Sciences)*, published by TUBITAK – Turkish Science and Technical Council), vol. 9. no. 1, pp. 63–70. 2001a.
- T.I. Ören. Software Agents for Experimental Design in Advanced Simulation Environments. In: S.M. Ermakov, Yu. N. Kashtanov, and V. Melas (eds.), *Proceedings of the 4th St. Petersburg Workshop on Simulation*, pp. 89–95. 2001b.
- T.I. Ören. Growing Importance of Modelling and Simulation: Professional and Ethical Implications. *Proceedings of the Asian Simulation Conference/the 5th International Conference on System Simulation and Scientific Computing*, Cheng, Zongji et al., eds., Shanghai, China. International Academic Publishers/Beijing World Publishing Corp. Vol. 1, pp. 22–26. 2002a.
- T.I. Ören. Rationale for a Code of Professional Ethics for Simulationists. *Proceedings of the 2002 Summer Computer Simulation Conference*, pp. 428–433. 2002b.
- T.I. Ören. Ethics as a Basis for Sustainable Civilized Behavior for Humans and Software Agents. *Acta Systemica*, vol. 2, no. 1, pp. 1–5. Also published in the Proceedings of the InterSymp 2002 – The 14th International Conference on Systems Research, Informatics and Cybernetics of the IIAS,

- July 29–August 3, Baden-Baden, Germany. 2002c.
- T.I. Ören. In Search of a Value System for Sustainable Civilization: Ethics and Quality. *Special Focus Symposium (within InterSymp 2002) on: Cognitive, Emotive and Ethical Aspects of Decision Making and Human Actions*, August 1–2, Baden-Baden, Germany, pp. 69–72. 2002d.
- T.I. Ören and L. Yilmaz. Quality Principles for the Ergonomics of Human–Computer Interfaces of Modeling and Simulation Software. *Proceedings of SIMCHI'05 – 2005 International Conference on Human–Computer Interface Advances for Modeling and Simulation*, New Orleans, LA, pp. 5–11. January 23–25, 2005.
- T.I. Ören and L. Yilmaz. Synergy of Systems Engineering and Modeling and Simulation. *Proceedings of the 2006 International Conference on Modeling and Simulation – Methodology, Tools, Software Applications (M&S MTS),* pp. 10–17, Calgary, Alberta, Canada. July 31–August 2, 2006.
- T.I. Ören. The Importance of a Comprehensive and Integrative View of Modeling and Simulation. *Proceedings of the 2007 Summer Computer Simulation Conference*, San Diego, CA, USA, July 15–18 2007a.
- T.I. Ören. Reliability in Agent-Directed Simulation. *Proceedings of the EMSS 2007 – 19th European Modelling and Simulation Symposium*, Bergeggi, Italy, pp. 78–86. October 4–6, 2007b.
- T.I. Ören. Uses of Simulation. *Principles of Modeling and Simulation: A Multidisciplinary Approach*, by John A. Sokolowski and Catherine M. Banks (eds.). John Wiley and Sons, Inc. New Jersey. 2009a.
- T.I. Ören. Modeling and Simulation: A Comprehensive and Integrative View. In L. Yilmaz and T.I. Ören (eds.). *Agent-Directed Simulation and Systems Engineering*, Wiley Series in Systems Engineering and Management, Wiley-Berlin, Germany. 2009b.
- T.I. Ören, M.S. Elzas, I. Smith and L.G. Birta. A Code of Professional Ethics for Simulationists. *Proceedings of the 2002 Summer Computer Simulation Conference*, pp. 434–435. 2002.
- D.K. Pace Modeling and Simulation Verification and Validation Challenges. *John Hopkins APL Technical Digest*, vol. 25, no. 2, pp. 163–172. 2004.
- A. C. M. Risks. Forum on Risks to the Public in Computers and Related Systems. *ACM Committee on Computers and Public Policy*, <http://catless.ncl.ac.uk/> Risks 2008. Last date of access: 6 July 2009.
- A.C.M. Risks. *References on Simulation for Failure Avoidance and Need for Failure Avoidance in Simulation*, <http://catless.ncl.ac.uk/php/risks/search.php?query=simulation> 2008. Last date of access: 6 July 2009.
- M.N.O. Sadiku and M. Ilyas. *Simulation of Local Area Networks*. Wiley. 1994.
- C. Schaller. *Concepts of Model Verification and Validation Report LA-14167-MS*. Los Alamos National Lab. 2004.
- S. Schlesinger. Terminology for Model Credibility. *Simulation* vol. 32, no. 3, pp. 103–104. 1979.
- T.W. Schoenharl and G. Madey. Evaluation of Measurement Techniques for the Validation of Agent-Based Simulations Against Streaming Data. *International Conference on Computational Science*, Krakow. <http://www.nd.edu/~dddas/Papers/51030006.pdf>. 2008. Last date of access: 6 July 2009.
- H. Scholten. *Better Modelling Practices: an Ontological Perspective on Multidisciplinary, Model-based Problem Solving*, PhD thesis, Wageningen University, Wageningen, The Netherlands. <http://library.wur.nl/wda/dissertations/dis4562.pdf>. 2008. Last date of access: 6 July 2009.
- G. Sheng, M.S. Elzas, T.I. Ören, and B.T. Cron-hjort. Model Validation: A Systemic and Systematic Approach. *Reliability Engineering and System Safety*, (Elsevier), vol. 42, pp. 247–259. 1993.
- N. Shillingford, G. Madey, R.C. Kennedy. Verification and Validation of an Agent-based Simulation Model. *Agent-Directed Simulation Conference*, San Diego, CA. 2005.
- K. Sohraby, D. Minoli, T. Znati. *Wireless Sensor Networks – Technology, Protocols, and Applications*, Wiley-Interscience. 2005.
- R.A. Stachowitz and C.-L.Chang. *Verification and Validation of Expert Systems*. Tutorial: SP2, AAAI88, 1988.
- A. Tolk. *Constraints for V&V of Agent Based Simulation: First Results – A System-of-Systems Engineering Perspective*,

- <https://orsagouge.pbwiki.com/f/Tolk100207.ppt>. 2007.
- M. Wooldridge and N. Jennings. Intelligent Agents: Theory and Practice. *The Knowledge Engineering Review*, vol. 10, no. 2, pp. 115–152. 1995.
- A.W. Wymore. System Theoretic Portents of Predisposition to System Failure. *The 9th International Symposium of the International Council on Systems Engineering*, Brighton, England. <http://www.incose.org.uk/incose99/download/242.doc>. 1999. Last date of access: 6 July 2009.
- X. Xiang, R. Kennedy, R. Madey, C. Cabaniss. Verification and Validation of Agent-based Scientific Simulation Models. *Proceedings of Agent-directed Simulation Conference*. San Diego, CA, 2005.
- L. Yilmaz. Validation and Verification of Social Processes within Agent-based Computational Organization Models. *Computational & Mathematical Organization Theory*, vol. 12, no. 4, pp. 283–312.
- L. Yilmaz and T. I. Ören. Agent-directed Simulation Systems Engineering. *Proceedings of the Summer Simulation Conference*, San Diego, CA. pp. 897–904. July 15–18, 2007.
- L. Yilmaz, P. Davis, P.A. Fishwick, X. Hu, J.A. Miller, M. Hybinette, T.I. Ören, P. Reynolds, H. Sarjoughian and A. Tolk. What Makes Good Research in Modeling and Simulation (M&S): Sustaining the Growth and Vitality of the M&S Discipline. *Proceedings of the 2008 IEEE/ACM Winter Simulation Conference*. pp. 677–689. 2008.
- A.A. Zabek. Lessons Learned from the Design and Execution of a Federation for Experimentation. *Proceedings of the 1999 Winter Simulation Conference*, pp. 1032–1038. 2007.
- B.P. Zeigler. *Multifaceted Modelling and Discrete Event Simulation*, Academic Press, London, UK. 1984.

8**Toward Systems Engineering for Agent-directed Simulation**

Levent Yilmaz

8.1**Introduction**

Complexity is a pervading phenomenon in natural, social, business, artificial, engineered, or hybrid systems. Cells, organisms, the ecosystem, markets, societies, governments, cities, regions, countries, large-scale software and hardware systems, the Internet – all are examples of complex systems (Parunak, 1999). Scientists use various forms of measurements and models to explore, understand, and elucidate the characteristics of such systems, while engineers build and design working artificial complex systems. As the engineered phenomena become more and more complex, we are observing a convergence as engineers try to model the systems in an attempt to analyze them. However, there is another trend that makes these models extremely complex.

Complexity research mainly happens at the borders between various disciplines and thrives on interactions between engineering and the sciences. Predictive modeling comes from the context of theoretical science, with a bias toward deductive reasoning and a resulting preference for validity as a standard quality. The current trend in modeling and simulation (M&S) treats the use of computer models as experimental science. In this new era, the purpose of M&S in dealing with complexity is not necessarily to predict the outcome of a system, rather it is to reveal and understand the complex and aggregate system behaviors that emerge from the interactions of the various individuals involved (Yilmaz, 2006). This viewpoint is based on the observation that emergent engineering applications are becoming dynamic, adaptive, and open systems (Little, 2005), for which the tools of the traditional closed-systems viewpoint are limited. More specifically, it is suggested in Little (2005) that if our critical infrastructures are to continue to provide vital services safely and reliably, the linkages between people, organizations, and technology need to be fully understood and managed holistically. As we start exploring the state space of such systems, the types of M&S applications, as espoused in this paper, will gradually increase and find their place within the engineering domain.

In this chapter we discuss the synergy of M&S and systems engineering (SE) by overviewing the role of M&S in SE and the need for SE for M&S in the large. In particular, we examine types and characteristics of systems that are complex, adaptive, and open. Agent-directed simulation (ADS) is presented as a candidate for addressing the challenges raised by M&S of such systems. A cognitive SE perspective is proposed to engineer such M&S applications.

8.2

What Is a System?

A system is a construct or collection of different elements related in a way that allows the achievement of a common objective (Thayer, 2005). The elements of the system include hardware, software, people, facilities, policies, and other factors coordinated to achieve system objectives.

8.2.1

What Is Systems Engineering?

SE involves the application of engineering and management practices to transform the user needs into a system specification and realization that most efficiently meets the given need. SE entails the technical management functions that control and coordinate the overall system development activities. As such, it revolves around a generic problem solving process that gradually evolves specifications toward a realization of the requirements that satisfy objectives set forth at the beginning of a project.

8.2.2

The Functions of Systems Engineering

SE involves:

- System conception, which deals with the process of involving users in conceiving an application and identifying tentative requirements.
- Problem specification, which identifies and formulates the needs and constraints imposed by the problem domain. Domain analysis constitutes the fundamental component of problem specification. Application analysis entails the description of the parts of the system that are visible to the user.
- Solution analysis, which determines the set of possible ways to satisfy requirements, overviews the solutions, and selects an optimal strategy.
- Process planning, which identifies the tasks and their scheduling and inter-dependencies, estimates the size and cost of the project, and determines the required effort to complete the project.
- Process control and product evaluation, which determine the strategies to control and measure the progress and evaluates the product via testing, inspection, and analysis.

8.3**Modeling and Simulation**

M&S, as a discipline, is vital for the success of many application areas in a multitude of disciplines. M&S has many facets; often specialists in one area have a tendency to ignore other aspects. At one extreme this leads to the point of view of “anything other than war is simulation”. Of course, this point of view stressing the main task is understandable; however, it does not allow a top-down decomposition of the elements of a simulation system to be able to develop advanced simulation environments and applications. Most probably as a result of this point of view, the importance of contributions of simulation to science and engineering, known to practitioners for several decades, has been rediscovered recently (NSF, 2006). However, this is a very good indicator of the acceptance of the value of M&S from another perspective.

Simulation is used for two categories of applications: (1) to gain experience (simulation in training) with live, virtual, and constructive simulations and (2) to perform experiments (simulation in areas other than training). Simulation can also be perceived as (1) an infrastructure to support real-world activities, a computational activity, a model-based activity, a knowledge generation activity, and a knowledge processing activity.

8.4**The Synergy of M&S and SE**

In Chapter 4, we elaborated on how M&S and SE can support each other. Here, we discuss the role of M&S in systems, why M&S requires SE, and why simulation system engineering (SSE) is necessary.

8.4.1**The Role of M&S in Systems**

Simulation is becoming a dominant technology in many SE applications. In defense-related training systems, simulations are being embedded to create virtual scenarios. Symbiotic simulation systems have been proposed as a way of solving this problem by having the simulation and the physical system interact in a mutually beneficial manner. As shown in Figure 8.1, the (multi)simulation (Yilmaz, 2007) is driven by real-time data collected from a physical system under control and needs to meet the real-time requirements of the physical system.

For instance, symbiotic simulation of network systems could be used to make use of measurements obtained from the physical network to optimize and reconfigure the physical network in order to improve its performance and avoid bottlenecks.

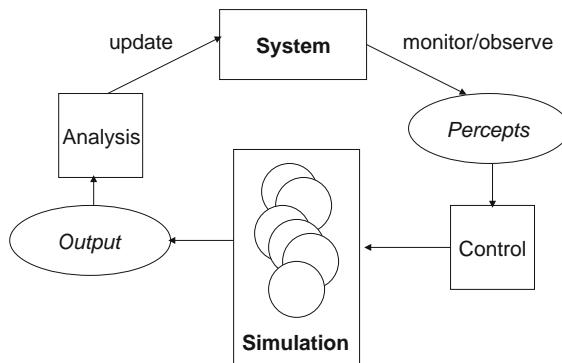


Fig. 8.1 Symbiotic simulation.

8.4.2

Why Does M&S Require SE?

As discussed above, M&S systems are becoming increasingly complex and they are being embedded with other systems in a system-of-systems context to serve larger objectives. In developing such simulations the solution space must be defined before assigning functionality to various components. The SE perspective provides an opportunity to specify the solution for the acquirer prior to allocation of functionality onto hardware, software, and simulation systems.

8.4.3

Why Is SSE Necessary?

M&S development costs are rising partly due to increased complexity. The craftsmanship approach to M&S on a small scale does not transfer to M&S on a large scale. Consequently, such complex and extremely large simulation systems require technical system management and SE oversight. Unless such oversight is present, the following problems are likely to emerge.

- Simulation system becomes unmanageable.
- Costs are overrun and deadlines can be missed.
- Greater risk exposure arises.
- Requirements may not be met.
- The simulation fails to satisfy its objectives.
- Maintenance costs increase.

8.5

Toward Systems Engineering for Agent-Directed Simulation

The emergent types of problems addressed by simulation systems are requiring more and more open, adaptive, and flexible solutions. For instance, in most re-

alistic scientific problems, the nature of the problem changes as the simulation unfolds. Initial parameters, as well as models, can be irrelevant under emergent conditions. Relevant models need to be identified and instantiated to continue exploration. However, manual exploration is often not cost effective and realistic within a large problem state space. Dealing with uncertainty is paramount to analyzing complex evolving phenomena. Adaptivity in simulations is necessary to evolve systems in a flexible manner. The use of intelligent agents is advocated, and agents are applied (Parunak, 1999) in numerous application domains to avoid limitations of conventional methods.

In this section, we elaborate on the essence of the complex adaptive systems that integrate people, organizations, and technology (Little 2005) and argue about the merits of ADS. Finally, we discuss characteristics of an SE perspective for ADS.

8.5.1

The Essence of Complex Adaptive Open Systems (CAOS)

- CAOSs are goal-directed and adaptive: In well-adapted systems, goals and constraints are often implicit and embedded in the process. In a dynamically changing environment, an effective realization depends on self-organizing and adaptive mechanisms that are in place to change the properties of the process to meet the current needs.
- CAOS processes improve over time: CAOS processes are frequently modified to update the structure and mechanisms to keep some measure related to the relevant performance objective near an optimum. Control of adaptation, however, is distributed across all components and subsystems. A useful and credible model for the analysis of the process and prediction of responses to changes in the circumstances must reflect the mechanisms underlying the evolution of the process.
- Many CAOS processes (e.g., sociotechnical systems) are human-centered: As suggested in Little (2005), if our critical infrastructures are to continue to provide vital services safely and reliably, the linkages between people, organizations, and technology need to be fully understood and managed holistically. Human actors that manage processes are adaptive and goal-directed agents. What people actually do, how they communicate and collaborate, how they solve problems, resolve conflicts, and learn behavior matters in the outcome of a process. Hence, representing activities requires modeling communication, collaboration, team work, conflict resolution, and tool and technology usage.

Given the above observations, an SE perspective for CAOS should represent not only technical activities, policies, and procedures, but also the resources, preferences, and cognition of staff members, together with functional and social organization and strategic management, all in unified and coherent terms.

8.5.2

The Merits of ADS

Figure 8.2 presents a unified paradigm of agent-directed simulation that consists of three distinct, yet related, areas that can be grouped under two categories as follows: (1) simulation for agents (agent simulation), that is, simulation of systems that can be modeled by agents (in engineering, human and social dynamics, military applications, etc.) and (2) agents for simulation that can be subdivided into two groups: agent-supported simulation and agent-based simulation.

Agent simulation involves the simulation of agent systems. Agent systems possess high-level interaction mechanisms independent of the problem being solved. Communication protocols and mechanism for interaction via task allocation, coordination of actions, and conflict resolution at varying levels of sophistication are primary elements of agent simulations. Simulating agent systems require understanding the basic principles, organizational mechanisms, and technologies underlying such systems. The principal aspects underlying such systems include the issues of action, cognitive aspects in decision making, interaction, and adaptation. Organizational mechanisms for agent systems include means for interaction. That is, communication, collaboration, and coordination of tasks within an agent system require flexible protocols to facilitate the realization of cooperative or competitive behavior in agent societies.

Agent-based simulation is the use of agent technology to generate model behavior or to monitor the generation of model behavior. This is similar to the use of AI techniques for the generation of model behavior, for example, qualitative simulation and knowledge-based simulation. The perception feature of agents makes them pertinent for monitoring tasks. Agent-based simulation is useful for having complex experiments and deliberative knowledge processing such as planning, deciding, and reasoning.



Fig. 8.2 Categories of the agent-directed simulation framework.

Agent-supported simulation deals with the use of agents as a support facility to enable computer assistance by enhancing cognitive capabilities in problem specification and solving. Hence, agent-supported simulation involves the use of intelligent agents to improve simulation and gaming infrastructures or environments.

8.5.3

Systems Engineering for Agent-Directed Simulation

Before we elaborate on the system conception, problem specification, and solution analysis components of an SE framework for ADS systems, we discuss pertinent requirements for such a framework.

- Distribution and propagation of objectives: In ADS systems (ADSS), decisions and control actions are not only taken by system entities, but also individuals (actors) carrying out functions at various levels. The allocation of roles to decision makers and the propagation of performance criteria to influence local objectives are critical in specifying the behavior of the system.
- Representation of the monitoring function available to actors: The perception of the system state by actors influences the goals to attend, the means for interaction, and subsequent actions. A significant challenge in open systems is to determine if the participants are complying with the applicable rules and policies (Singh, 2004).
- Representation of autonomy: Complete autonomy may have undesirable effects, as in the case of real-world situations. The autonomy of the entities in the system needs to be constrained based on interaction protocols.
- Representation of heterogeneity: Large complex systems end up being heterogeneous. The linkages between models of people, organizations, and technology should be carefully planned to facilitate interoperation by design.
- The role of dynamism: Dynamics refers to the independence of the controller or administrator of the system to update and change configuration without notification to other parties associated with the system. Open systems are particularly dynamic, as, in principle, they do not need administrators due to decentralization.
- The significance of communication: Communication is the basis of interaction in systems (Ferber, 1999). Communication among actors in sociotechnical systems helps preserve their autonomy.

8.6

Sociocognitive Framework for ADS-SE

It is clear from the discussion that the analysis and design of ADSS cannot be based on conventional SE problem and solution analysis methods. Instead, the engineering of ADSS must be based on an analysis of the behaviors that shape the

Dimensions of Problem Analysis for ADSS				
	Means-ends dimension	Activity dimension	Social dimension	Cognitive dimension
Goals and constraints	goals, objectives, organizational structure	activity dependency, scheduling constraints	social norms, form of communication and interaction	cognitive complexity, limits, personality structure, etc.
Interaction function	content of communication	activity in decision making terms	social network	collective decision making
Operational function	workflow, physical processes	activity in terms of operational terms	cooperative patterns of behavior	human /team behavior representation
Organizational function	management of communication, collaboration	coordination of activities	control of social interaction	team structure, cohesion, team archetypes
Meta-level function	strategy update, acquisition and maintenance of resources	planning, (re)allocation of activities	Social network revision/update, organizational adaptation	adaptive cognition

Fig. 8.3 Dimensions of analysis for ADS-SE.

constraints and goals of the application domain in which actors are autonomous and perform on the basis of their local objectives and performance criteria.

8.6.1

Social-Cognitive View

An SE approach to ADSS analysis and design needs to account for the interdependencies among physical processes, activities, and social and cognitive preferences. Figure 8.3 presents a matrix that partitions the problem and solution analysis. The four main dimensions represent the means-ends, activity, social, and cognitive perspectives. Specifically, the means-ends perspective identifies the structure and general global knowledge base of the ADSS.

The activity dimension involves further delimitation of the solution space regarding the meaningful activities that realize task situations defined in the problem domain. The social dimension constrains the form of communication and takes the social norms into account to influence the coordination of interaction. Therefore, the level of control of social interaction and the functional constraints based on activity dependencies together affect the way tasks are carried out. The cognitive domain further refines the analysis by taking the cognitive resource profiles and preferences of individuals and teams into account.

8.6.2

The Dimensions of Representation

Figure 8.4 depicts the mechanism by which the interaction between different dimensions ensues. The means-ends dimension at the bottom involves the problem solving activities that carry out the tasks assigned to individual members.

The ways these activities carried out are influenced not only by the constraints of the domain model but also the strategies imposed by the human behavior subsystem. This subsystem affects the performance of individuals by inducing human behavior variability in terms of cognitive, affective, and personality traits and factors. The social interaction control level is driven by the social subsystem. The form of communication and interaction styles are governed by the team archetypes, organizational culture, and decision making styles at the social organization level. The structure of the communication net and the content of the communication are based on the functional work organization, and hence they are determined by the control requirements of the problem domain. The social interaction control level along with the activities determines the shape of the coordination of work activities.

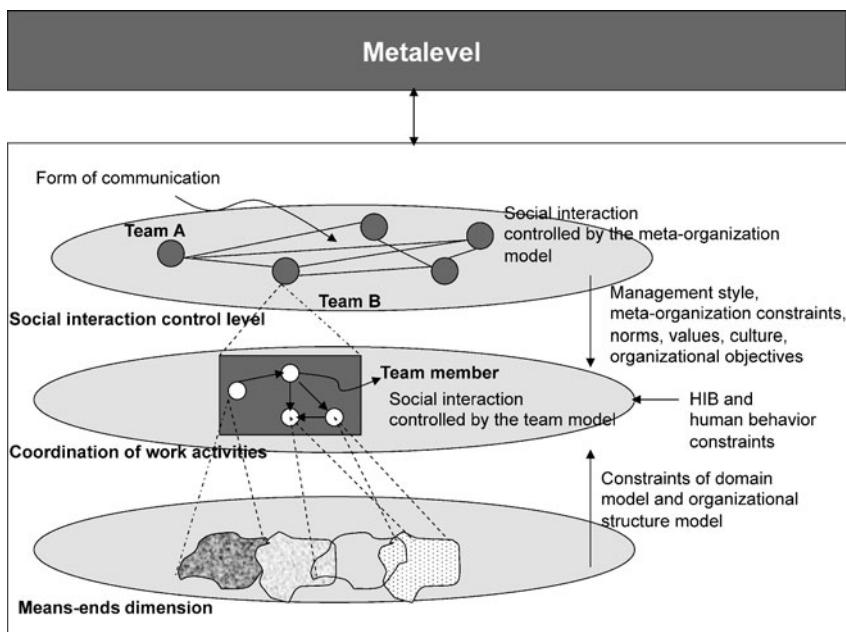


Fig. 8.4 Levels of abstraction for ADS-SE.

8.6.3

The Functions for Analysis

Each dimension shown in Figure 8.3 can be analyzed in terms of constraint, interaction, organization, operational, and meta-level functions. The purpose of the constraint function is to formalize the goals, objectives, and constraints over the elements of the dimension. The interaction function serves to create a link among elements of the domain and to interface heterogeneous agents. The operational function focuses on the physical processes that pertain to workflow within the system. The workflow is defined at various levels of abstraction (activities in domain terms, activities in decision making terms, etc.) depending on the level of decomposition depicted by the dimension. The organizational function concerns the management of the activities and interactions within the organization that represents the configuration of the elements of the ADSS. The meta-level function entails the maintenance and preservation of the resources and agents and flexible adaptation of the behavior of the ADSS based on the observations and monitoring of the environment and system components.

8.7

Case Study: Human-Centered Work Systems

Software development is a knowledge acquisition activity that involves the transformation of the user needs into a software product that realizes the requirements elicited from these needs. Figure 8.5 depicts the elements of the hypothetical organization-theoretic framework for software process simulation.

The transformation processes are influenced by inputs such as resources, the organizational culture (i.e., decision making styles – consensus vs. centralized), norms, values, budget, and objectives. Objectives include product differentiation,

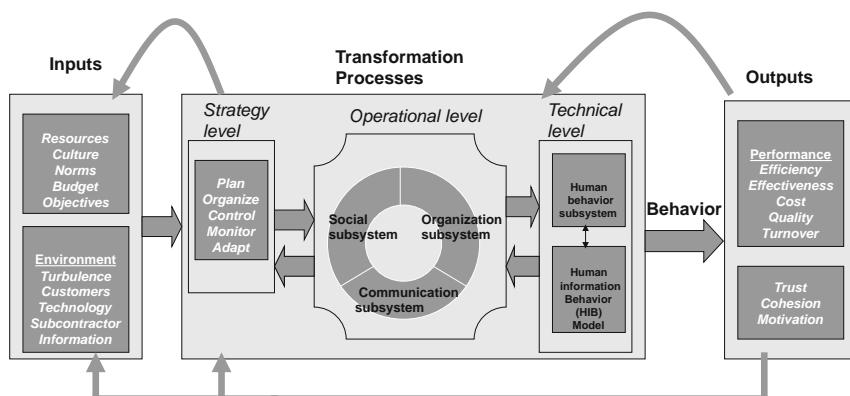


Fig. 8.5 Sociocognitive framework for engineering work systems.

innovation, market expansion, and risk reduction. Environmental inputs entail turbulence (e.g., task uncertainty, turnover, requirements change), the impact of customers, technology, and available information regarding the attainment of objectives. The inputs are transformed by a multiresolution process. Three levels interplay to represent the organizational, human, and social dynamics that shape the behavior during software development.

- The *strategy level* can be viewed as the meta-level control mechanism that models the behavior of the management. It is responsible for monitoring, controlling, and adapting the operational level via dynamic model updating. It is also responsible to (re)organize the social and physical structure of the organization.
- The *operational level* consists of three dimensions that collectively model the process, associated activities and tasks, the social and physical structure and interaction in the organization, and the communication mechanism among the actors that carry out the process.
- The *technical level* refers to the human-activity level. In particular, the human activity is based on the Human Information Behavior (HIB) perspective that examines the human behavior in relation to sources and channels of information via information seeking, searching, and use mechanisms. The knowledge acquisition view of software development, in conjunction with the human information behavior model, provides an accurate representation of how humans actually work in practice. The HIB model is supported by the human behavior subsystem. Modeling personality traits, cognitive complexity factors, and affective factors enable representing individual differences to bring variability and credibility to process simulations.

The outputs depicted by the framework include performance metrics such as productivity (e.g., effectiveness and efficiency), project cost and duration, product quality, and turnover. Attitudinal behavior outputs measure engineering team and human cognition factors such as trust, motivation, and cohesion, which further impact the inputs and the transformation processes.

8.7.1

Operational Level – Organizational Subsystem

The formal organizational subsystem defines such things as specification of workflow, activities, work breakdown and organization structure (including authority), task structure (representation of formal requirements), and job satisfaction. Figure 8.6 presents the conceptual elements of the organization subsystem of the operational level of the transformation system shown in Figure 8.5. The work domain model specifies the means-ends structure of the process in terms of goals/constraints, abstract functions, general functions, and work activities. The highest level of abstraction in the means-ends structure is the set of constraints and goals, which are the policies that govern the interaction between the work system and its environment. For software development processes, productivity,

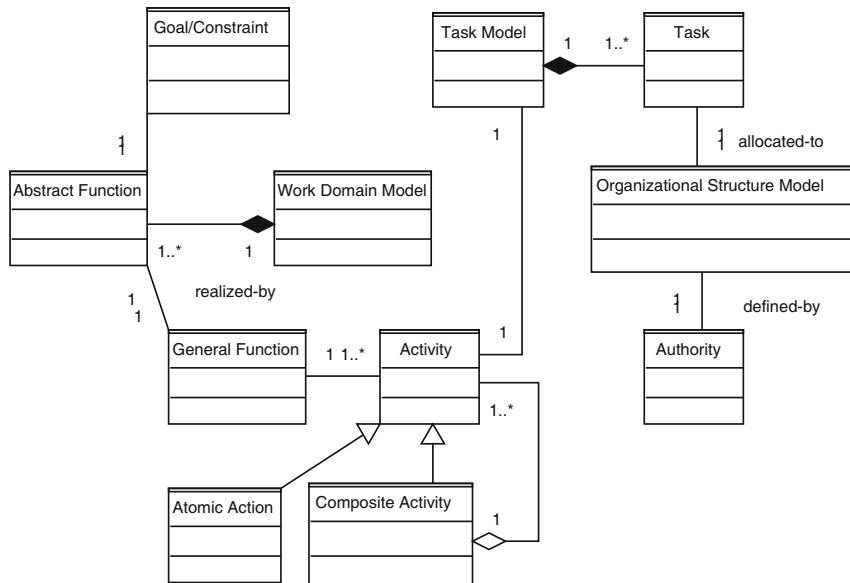


Fig. 8.6 Conceptual model for the organization subsystem.

cost, and quality, as well as production within the constraints of the financial resources, are potential constraints. Abstract functions of the organizational system in our context denote the representation of concepts that are necessary for allocating resources to general functions and to activities. Departmental functions such as quality assurance, product development, controlling, planning, and human resource management are abstract functions for which general work activities are defined. General work functions are at a lower level of abstraction and are defined in terms of activities and task sequences of groups and individual agents.

The organizational structure model depicts the physical form and configuration of the organization. The set of linkages that connect agents, tasks, and resources constitute the structure of an organization. There can be many structures. The authority (e.g., centralized vs. decentralized) and communication structures are the most common ones.

8.7.2

Operational Level – Organizational Subsystem

A course-grain and high-level conceptual model for the social subsystem is shown in Figure 8.7. The focus of the social subsystem is the meta-organization model that specifies the relations between actors, resources, artifacts, tasks, and teams. The meta-organization model specifies the architecture of cooperative work and the criteria for division of work between teams. It is important to distinguish between the work organization (i.e., organization subsystem) perspective and the so-

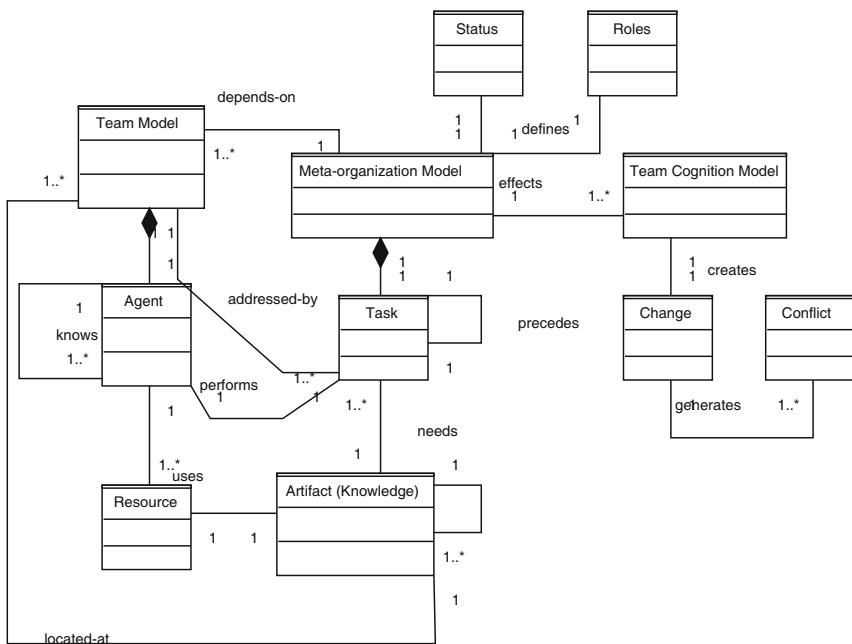


Fig. 8.7 Conceptual model for the social subsystem.

cial organization aspect depicted by the meta-organization model within the social subsystem.

The work domain model of the organization subsystem analyzes and specifies the coordination activities determined by the interaction of the control requirements of the work domain and the behavior of teams and engineers. On the other hand, the social organization perspective imposed on the interactions among teams and engineers depends on the management style, culture, norms, and values and the configuration of the social networks.

The team model embodies the team composition, structure, and explicit coordination and collaboration styles. Common coordination mechanisms in organization theory are rules, plans, hierarchy, and mutual agreement. Various team archetypes also influence the behavior of teams. The team cognition model incorporates elements such as trust, motivation, and cohesion that are effective in the performance of a software development team. The team cognition mode constitutes mechanisms that suggest specific changes and adaptation requests that may cause conflicts that have to be resolved by the strategy subsystem of the overall framework. The meta-organization model makes use of role and status information to improve coordination among teams and update team cognition parameters, respectively. For instance, the role of team leader is critical in allocating a specific task to a team via its leader. Also, the status information assigned to individual agents helps assign weights to decisions made by the members of a team to derive a team decision.

8.7.3

Operational Level – Integration of Organization and Social Subsystems

The operational level can be considered as a distributed control subsystem that serves a loosely coupled software process simulation that is viewed as a work system. The cooperation among the actors evolves from two directions. The work activities specified by the organization subsystem affect the control activities bottom-up, while the social organization and its cooperation mechanism propagate top-down. The software development work organization emerges as a result of the interaction between the social practice and management style depicted by the social subsystem and the control requirements of the work domain model. Figure 8.4 depicts the mechanism by which the interaction ensues. The work activity level at the bottom involves the problem solving activities that carry out the tasks assigned to individual team members. The ways in which these activities are carried out are influenced not only by the constraints of the work domain model, but also the strategies imposed by the HIB model. The human behavior subsystem affects the performance of individuals by inducing human behavior variability in terms of cognitive, affective, and personality traits and factors. The social interaction control level is driven by the social subsystem of the operational level. The form of communication and interaction styles are governed by the team archetypes, organizational culture, and decision making styles at the social organization level. The structure of the communication net and the content of the communication are based on the functional work organization, and hence they are determined by the control requirements of the work domain. The social interaction control level, along with the work activities, determines the shape of the coordination of work activities. Specifically, the constraints of the work domain model (i.e., software process technology) and the structure of the organization (i.e., authority, hierarchy) explicitly delineates how team members need to coordinate to fulfill tasks in accordance with the standards and process guidelines. The management style and social practice of the organization further constrain the flow of information and interaction among team members.

8.7.4

The Technical Level

The technical level (HIB and human behavior system) of the proposed framework not only affects the coordination effectiveness at the work activity coordination level (Figure 8.4), but also influence the performance of team members, as they carry out work activities.

8.7.4.1 Human Information Behavior (HIB) Model

Modeling the activities of humans as they carry out tasks requires realistic representations of domain-independent behavior regarding how humans solve problems in real life. Unfortunately, M&S of software processes is often done at an ab-

stract level so that individual and social work practice involving collaboration, communication, “off-task” behaviors, multitasking, interrupted and resumed activities, and informal interactions are not captured. Work activities imposed by the processes can be viewed as knowledge acquisition activities that require engineers to seek, search, use, and synthesize information to derive knowledge (i.e., design constraints and models) that is eventually embodied in the software artifact. The HIB model elaborates on the common characteristics of information behavior. Information behavior is defined as human behavior as it pertains to sources and channels of information. Information seeking requires interacting not only with computers but also other manual and natural sources (e.g., face-to-face communication between team members) to obtain information to satisfy a goal. Information search involves micro-level behavior involving interaction with information systems to locate information. Information use consists of mental and physical human acts that pertain to incorporating the discovered information into one’s knowledge base. The definition of each specific work activity in terms of primitive information seeking, search, use, and synthesis operations constitutes the foundation of the application of HIB.

8.7.4.2 Human Behavior Subsystem

Not every team member performs and interacts in the same way. According to personality psychologists (Bern, 1983), the fundamental task in human behavior analysis is to translate observations of persons with particular traits behaving in specific manners in particular situations into patterns (assertions) that certain kinds of people behave in certain kinds of ways in certain kinds of situations. Others have also emphasized the importance of human behavior in terms of sound and predictable patterns that specify well-defined groups of behavior in relation to groups of situations. To have a realistic basis for simulating software development, one should consider individual behaviors as part of the human aspect. In this article, we stress the role of cognitive complexity of individuals and its relation to one of the five traits of human personality, that is, openness to the success of software teams. Even a brief review of the basic concepts of cognitive complexity and openness will shed light on their relevance to the success of software teams.

8.7.4.3 Cognitive Complexity

In software engineering, as is the case in many other complex systems, the ability to cope with complexity is a fundamental issue and influences the quality of decisions. As early as the 1970s, systems engineers (Athey, 1976) elaborated on the importance of increasing the cognitive complexity of an individual to increase his/her effectiveness in coping with complex situations. Figure 8.8 shows different levels of information processing of an individual depending on the situational complexity. For a low situational complexity, the individual may need to have a low level of information processing to cope with the situation. If the situational complexity increases, his/her information processing level may also increase. However, for each individual there is a critical point beyond which the level of processed informa-

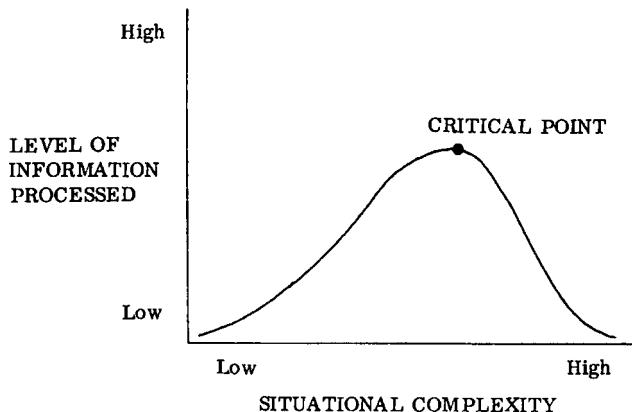


Fig. 8.8 Relationship between situational and behavioral complexities (Yilmaz and Ören, 2007).

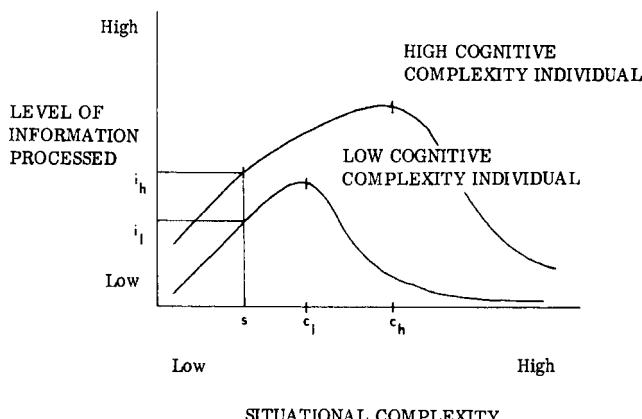


Fig. 8.9 Situational and behavioral complexities across individuals (Yilmaz and Ören, 2007).

tion, and hence the individual's information processing effectiveness, decreases. After that critical point, an increase in the situational complexity may worsen the individual's ability to cope with complexity.

The information processing curves of two types of individuals, that is, high and low cognitive complexity individuals, are compared in Figure 8.9, which illustrates two important points. First, c_h , the critical point of a high cognitive complexity individual is higher than c_l , the critical point of a low cognitive complexity individual. Thus increasing the cognitive complexity of an individual within the applicable limits of course may increase the range of situational complexity within which he/she can perform effectively. Or, depending on the task, it may be advisable to assign an individual with cognitive complexity commensurate with the task. Second, for a given situational complexity, the level of information processed by a high cognitive complexity individual, i_h , is greater than that of a low cognitive complexity individual, i_l .

8.8

Conclusions

The characteristics of emergent open, complex, and adaptive M&S applications are overviewed to make the case for ADS-SE. An ADS view of developing such applications is presented within the framework of a cognitive SE perspective. Since large and complex system development is inherently an organizational effort, we need to find ways to understand the influence of alternative organizational structures, strategies, and operational mechanisms on the effectiveness of processes. Developing ADS models to analyze the performance of processes for such large complex system development endeavors requires principled application of sociocognitive SE methods. Such principles should embody realistic assumptions that pertain to (1) strategic management of the organization in an adaptive goal-directed manner, (2) organizational structure, functions, and work activities imposed by the process technology, (3) social work organization that reflects the social practice, norms, management style, and culture, and (4) human work activities and behavioral traits.

References

- Bern, D.J. (1976) *Training the Systems Analysts to Solve Complex Real World Problems*. Proceedings of the 14th Annual Computer Personnel Research Conference, (ed. T.C. Willoughby), July 29–30, 1976, The Special Interest Group on Computer Personnel Research (SIGCPR) of the ACM, pp. 103–120.
- Bern, D.J. (1983) Constructing a Theory of the Triple Typology: Some Thoughts on Nomothetic and Idiographic Approaches to Personality. *Journal of Personality*, 53, 566–577.
- Ferber, J. (1999) *Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence*, Addison-Wesley.
- Little, R.G. (2005) *Organizational Culture and Performance of Critical Infrastructure: Modeling and Simulation of Socio-Technological Systems*. Proceedings of the 38th Hawaii International Conference on System Sciences, Hawaii, USA.
- U.S. NSF. *NSF Blue Ribbon Panel on Simulation-Based Engineering Science: Revolutionizing Engineering Science through Simulation*, NSF, May 2006.
- Van Dyke Parunak, H. (1999) Industrial and practical applications of dai. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, pp. 377–421.
- Singh, M.P. (2004) Agent-based abstractions for software development, in *Methodologies and Software Engineering for Agent Systems: The Agent-oriented Software Engineering Handbook*, Springer, pp. 5–16.
- Thayer, R.H. (2005) Software system engineering: A tutorial, in *Software Engineering Volume 1: The Development Process*, Wiley Inter-Science, pp. 31–46.
- Yilmaz, L. (2006) Expanding our horizons in teaching the use of intelligent agents for simulation modeling of next generation engineering systems. *International Journal of Engineering Education*, 22(5), 1097–1104.
- Yilmaz, L. (2007) Toward next generation simulation-based computational tools for conflict and peace studies. *Social Science Computer Review*, 25(1), 48–60.
- Yilmaz, L. and Ören, T.I. (2007) *On Multiresolution Simulation Modeling of Team and Human Behavior for Software Process Design*. Proceedings of the Tenth World Conference on Integrated Design and Process Technology, pp. 87–94.

9

Design and Analysis of Organization Adaptation in Agent Systems

Virginia Dignum, Frank Dignum, and Liz Sonenberg

9.1

Introduction

One of the main reasons for creating organizations is to provide the means for coordination that enables the achievement of global goals. However, organizations cannot be understood without considering the environment in which they exist. In fact, organizations can be seen as open systems in that they have to interact with their environment in order to realize those goals (Aldrich and Pfeffer, 1976). On the one hand, the organization provides stability and efficiency for the individuals working within it, but on the other hand, the organization must be flexible in order to adapt to changes in its environment. As such, organizations are complex dynamic goal-oriented processes, with outcomes dependent both on the contextual situation as on the coordinated and purposeful action of different entities in the environment.

Organizational structure facilitates the delegation of authority, allocation of resources, and decision making within the organization. Ideally, the organization is designed to fit its environment and to provide the information and coordination needed for its strategic objectives.

Organizational success is brought about by the organization's ability to bring all its information and assets to bear, and the ability to recognize and take advantage of fleeting opportunities. That is, for any enterprise to succeed, the structure, its strategy, and its resources must be aligned together. However, organizations are not static: they change, disappear, or grow. Entities can migrate, organizational objectives and environmental conditions can change, or operational behavior can evolve. In those cases reorganization is necessary to guarantee the maintenance or improvement of the utility of the system. That is, the reorganized instance should perform better in the modified environment than the original instance. From the perspective of the individual agents, their participation in an organization also depends on utility factors. As a rule, they will continue their participation in an organization as long as, from their perspective, their utility is maintained.

Whether the aim is to design a new organization or to analyze an existing one, managers require the knowledge and tools that enable them to effectively identify and enforce communication and control structures, and to assess and respond to environmental changes that are paramount for the survival of the organization. Organization modeling requires integrated models of people, tasks, goals, and dynamic environments (Sierhuis, 2001). The use of agent organization models for analysis, design, and development for complex domains is growing rapidly (Castelfranchi, 2001; Dignum, 2004; Hübner et al., 2006). Our research aims at testing congruence theories by developing formal models that enable the specification and comparison of different organizational structures, their performance properties, and congruence. Our focus is on the decision processes that enable the evaluation of the fit of an organizational design to some environmental conditions and determine the changes that result in better performance given those environmental conditions. That is, we assume that organizations try to locate the structure that best fits their environment and will make changes in their design accordingly. Our approach fits in the tradition of computational organization theory, which views organizations as a distributed computational system of multiple autonomous agents that work collectively toward the realization of common objectives, within the borders described by global requirements and environment (Carley and Svoboda, 1996).

In this chapter, we will discuss the notion of organization in human society and how it can be used to better design and analyze multiagent systems (MASs). Our interest in drawing on research in human organizations is twofold: first, we look to draw on general organizational principles that may apply to artificial as well as human organizations; second, we are ultimately interested in being able to build hybrid human-agent networks, and so staying within the bounds of organizational properties that have some analog in human behavior seems desirable. We will present a model for the specification of organization simulations that enable the study of reorganization issues. We will further see that current approaches to organizational simulation often focus on a partial view of the organizational spectrum (structure, individual capabilities, and environmental conditions) and present a conceptual framework that enables the classification and comparison of different simulations.

This chapter is organized as follows. In the next section we describe our view of the organization model. In the three subsequent sections the main components of this model are discussed. In Section 9.6 we introduce the elements necessary for describing reorganization. In Section 9.7 we show how organizations can be designed with a view to reorganization. In Section 9.8 we define several dimensions along which the simulations of reorganization can be measured and we show how this can be used to compare different simulations of reorganization. Finally, we draw some conclusions in Section 9.9.

9.2

Organizational Model

The definition of organizations as instruments of purpose implies that organizations have goals, or objectives, to be realized. The objectives of an organization are achieved through agent action, which means that an organization should employ the relevant agents, so that it can “enforce” the possibility of making its desires happen. Furthermore, one of the main reasons for creating organizations is efficiency, that is, to provide the means for coordination that enables the achievement of global goals in an efficient manner. This means that the agents in the organization need to coordinate their activities in order to efficiently achieve those objectives. Moreover, at any moment, unexpected change can also occur that is not the result of the action of any of the agents in that world, leading to the need to evaluate the conditions of the environment and possibly decide to change the current strategy.

In its most simple expression, an organization consists of a set of individuals and a set of objectives in a given environment. Organizational objectives are not necessarily shared by any of the individual participants; they can only be achieved through combined action. According to contingency theory (Donaldson, 2001), in order to achieve its goals it is thus necessary that an organization employ the relevant agents and that it structure its interactions and responsibilities to enable an efficient realization of the objectives (So and Durfee, 1998). This leads to the need for at least the following three elements in an organization:

- *Organizational structure* consists of roles, their relationships, and predefined (abstract) interaction patterns. Organizational structure must reflect and implement the global objectives of the organization. *Roles* have objectives determined by the global aims of the organization and can be grouped into subgroups. Role objectives determine possible dependencies between different roles. Roles describe classes of agents, their activities, and possibly their norms and behavior rules. Roles are related to other roles by *dependency relations*. Desired *interaction patterns* between roles can be specified.
- An *agent* participates in the organization (system) by playing one or more roles. Role enactment either is achieved through allocation by the system developers who determine which available agent is the most adequate for a task or is decided by the agents themselves. In both cases, analysis techniques are needed to support enactment decision by comparing and evaluating different role allocations (Nair et al., 2003). The set of agents active at a given moment in an organization is called the *population*. An agent population achieves the animation of organizational structures.
- The *environment* is the space in which organizations exist. This space is not completely controllable by the organization and therefore results of activities cannot always be guaranteed. Two dimensions of environment are unpredictability and (task) complexity. The environment also includes the description of tasks and resources (such as size and frequency).

In the following sections, we look at each of these aspects in more detail, and in Section 9.7 we present a design framework based on this model.

9.3

Organizational Structure

Organizational structure has essentially two objectives (Duncan, 1979). First, it facilitates the flow of information within the organization in order to reduce the uncertainty of decision making. Second, the structure of the organization should integrate organizational behavior across the parts of the organization so that it is coordinated. The design of the organizational structure determines the allocation of resources and people to specified tasks or purposes and the coordination of these resources to achieve organizational goals (Galbraith, 1977). Both in human enterprises as in MASs the concept of structure is central to the design and analysis of organizations, as discussed in this section.

9.3.1

Organizational Structures in Organization Theory

Organization theory has for many decades investigated the issue of organizational structure. The key characteristic of organizational structure is that it links the elements of the organization by providing the channels of communication through which information flows.

Notable is the work of Mintzberg on the classification of organizational structures (Mintzberg, 1993) according to which environmental variety is determined by both environmental complexity and the pace of change. He identifies four types of organizational forms, which are associated with the four possible combinations of complexity and change. Each of the four forms of organization depends on fundamentally different coordination mechanisms. Table 9.1 summarizes Mintzberg's taxonomy of organizations.

Organizational design mostly adopts a multicontingency view, which says that an organization's design should be chosen based on the multidimensional charac-

Table 9.1 Organizational forms according to Mintzberg.

Organizational form	Machine bureaucracy	Professional organization	Entrepreneurial startup	Adhocracy
Complexity	Low (simple)	High (complex)	Low (simple)	High (complex)
Change pace	Low (stable)	Low (stable)	High (dynamic)	High (dynamic)
Coordination mechanism	Standardized procedures and outputs	Standardized skills and norms	Direct supervision and control	Mutual adjustment of ad hoc teams

teristics of a particular environment or context. These characteristics include structural (e.g., goals, strategy, and structure) and human (e.g., work processes, people, coordination, and control) components. Contingency theory states that even if there is not one best way to organize, some designs perform better than others. In fact, Roberts states that the central problem of organizational design is finding a strategy and an organizational structure that are consistent with each other. This requires finding a dynamic balance in the face of strategic and organizational changes (Roberts, 2004). Organizational design is therefore about understanding the characteristics of the environment and the demands of this environment in terms of information and coordination. Given that there are no exact recipes to construct the optimal organization, means to evaluate a certain design and determine its appropriateness given the organization's aims and constraints is a main issue in organization theory.

9.3.2

Organizational Structures in Multiagent Systems

In multiagent systems (MAS) research, organization has been defined as “*what persists when components or individuals enter or leave an organization, that is, the relationships that makes an aggregate of elements a whole*” (Ferber et al., 2003). Establishing an organizational structure that specifies how agents in a system should work together helps the achievement of effective coordination in MAS (Barber and Martin, 2001). A social structure may be explicitly implemented in the form of a social artifact existing independently of the implementations of agents, may be realized as part of the implementations of the agents, or may exist only intangibly, in the form of the policies or organizational rules followed by the agents during interaction (Malyankar, 1999).

In *organization-oriented MAS*, design starts from the social dimension of the system and the system described in terms of organizational concepts such as roles (or function, or position), groups (or communities), tasks (or activities), and interaction protocols (or dialog structure), thus on what relates the structure of an organization to the externally observable behavior of its agents.

In general, two types of MAS structure are distinguished:

- *Emergent MAS*: global behavior cannot be specified in advance, but emerges from the interaction of local behaviors (Mussio, 1998).
- *Designed MAS*: have an explicit, predefined interaction structure that determines the coordination of the agents participating.

Emergent systems can be seen as conglomerations of single agents with hardly any fixed interaction or explicit social structure. There is no notion of common goals or plans, and agents are free to enter or not into interaction with others. That is, emergent organizational behavior is primarily a bottom-up process in which agents look for interaction and local control decisions that have been effective in the past and give similar decisions preference in the future. By aggregating these preferences together, an emergent organizational structure can be created. Individual agents may

not be aware of the emergent global behavior, which is only perceivable by an external observer. An agent behaves following its own local laws under the influence of the other agents and of an external environment. Individual agents may compete for resources, yet produce a common effect. Structure has a temporal determination: it is cumulative over time in a single direction, nonreversible, and determines the action of agents differently as it evolves. Structure is implicit in the activity of the agents. New agents cannot use that structure but must learn from scratch and adapt as they go along. The emergent paradigm can be applied to both open and closed domains.

In contrast, *designed systems* are created using organizational-design knowledge and task-environment information to develop an explicit organizational structure that is then elaborated by the individual agents into appropriate behaviors. Social structure is determined by organizational design, which is independent of the agents themselves. Such structures implement the idea that agent interactions occur not just by accident but aim at achieving some desired global goals (Weigand et al., 2003). That is, there are goals external to each individual agent that must be reached by the interaction of those agents. The desired behavior of a society is therefore external to the agents. Organizational hierarchies, teams, shared blackboards, global plans, and auction systems are all social structures. In closed domains, designed structures completely determine the communication primitives available and possibly describe the resources available in the environment. Furthermore, they may organize agents into groups or teams and specify joint action. A special case of designed systems are *agent societies*, which include social concepts such as norms and ontologies. Besides the specification of action-oriented behavior as in designed MAS, agent societies enable the specification of normative behavior and as such provide agents with the possibility of reasoning about their own behavior, plans, and goals. Agent societies are norm-oriented structures, appropriate to model open domains, also due to the explicit specification of ontological aspects.

9.4

Organization and Environment

Environments affect organizations through the way in which resources are available or withheld, and organizational form determines the efficacy in obtaining resources. That is, the interaction between an organization and its environment, which is determined by the organization's structure, is a main determinant of the organization's success.

9.4.1

Environment Characteristics

Three key environment characteristics are used in the evaluation and/or determination of organizational structures (Dess and Beard, 1984):

Task complexity: ranging from simple to complex; refers to the heterogeneity and range of an organization's activities.

Volatility: ranging from static to dynamic. Most research in organization theory suggests that turnover, the absence of pattern, and unpredictability are the best measures of environmental stability-instability. It is also important to measure the rate of environmental change.

Resource capacity: Also referred to as *munificence*, describes the extent to which the environment can support growth. Capacity can range from abundant to scarce availability of resources.

By considering volatility as the key feature of the environment, Burns and Stalker (1961) provided a classification of organization forms that fit to specific circumstances of change or stability. *Mechanistic* organizational structures are hierarchical and characterized by (1) centralization of authority, (2) formalization of procedures and practices, and (3) specialization of functions. Mechanistic organizations are comparatively simple and easy to organize, but find it difficult to cope with rapid change. *Organic* organizational structures (also called open organizations) are characterized by (1) flat relations (communications and interactions are horizontal), (2) low specialization (knowledge resides wherever it is most useful), and (3) decentralization (great deal of participation in decision making). Organic organizations are comparatively more complex and harder to form, but are highly adaptable, flexible, and more suitable where the external environment is rapidly changing and is unpredictable.

Simple, static, and abundant environments are fairly certain, whereas complex, dynamic, and scarce environments are uncertain. There is considerable evidence that mechanistic models are more appropriate in the former, and organic models are more suitable for the latter type of environments. This implies that organization structure should be such that it is able to deal efficiently with environmental uncertainty (Hall, 1991). An environment with low uncertainty exhibits stable conditions while an environment with high uncertainty places more pressure on the organization to respond effectively.

Depending on the type of organization and on the perceived impact of the changes in the environment, adaptation is achieved by behavioral changes at the agent level, by modification of interaction agreements, or by the adoption of a new social structure or a new strategy. Even though in most MASs organizational adaptation is realized by reengineering the system (i.e., external assessment and modification of a system), for MASs to be truly autonomous, mechanisms for dynamic reorganization must be available. The concept of *dynamic adaptation* refers to modifications in the structure and behavior of an MAS, such as adding, removing, or substituting components, done while the system is running and without bringing it down (Valetto et al., 2001). Dynamic adaptation demands that systems be able to evaluate their own "health" (i.e., success and other utility parameters) and take action to preserve or recover it by performing suitable integration and reconfiguration actions. Reorganization of organizations should therefore describe both situations in which the operational behavior of the organizational changes,

due to arrival or departure of agents, as well as situations in which the social structure of the society changes, that is, roles, relationships, norms, or interactions change.

Most existing approaches to reorganization consider only the behavioral aspects, that is, reorganization only affects the current population of agents in the system, both at the social (i. e., interactions and relationships) (Carley and Gasser, 1999) and at the individual level (Hannebauer, 2002). The contribution of this chapter is that we discuss a number of aspects that are important for structural reorganization, that is, permanent changes to the organizational structure of the system.

9.4.2

Congruence

Some recent research in the investigation of organizational restructuring in human organizations (Diedrich et al., 2003; Entin et al., 2003) has identified aspects of organizational structuring that need to be made explicit in the design of organizational adaptation. In human settings, organizational performance has been demonstrated empirically to be associated with the degree of congruence (or “fit”) between organizational structure and properties of the task or environment (Donaldson, 2001). Accordingly, it is to the advantage of the organization to monitor the fit between its structure and mission and to alter its structure when a misfit is identified. There is empirical evidence that high performing organizations can discern when environmental forces have changed the state of congruence (i. e., the goodness of fit), thus driving changes in the strategies (e. g., communication patterns, backup behaviors) that they employ (Entin and Serfaty, 1999). Rarely, however, do human organizations make changes to their organizational structures (i. e., asset allocation, team member roles and responsibilities) in order to facilitate congruence, and at least some of the explanation for this relates to characteristics of human behavior that are not necessarily replicated in artificial organizations. However, we believe there are aspects of human organizational adaptation from which lessons can be drawn for the design of mechanisms for the adaptivity of artificial organizations. In particular we are interested in understanding what can be identified from human organizational behavior about the triggers for reorganization and strategies for implementation. Of special interest is being able to reveal *explicitly* the kinds of knowledge that need to be considered when making reorganization decisions.

Of note is the extensive empirical work of the Aptima group¹⁾ on organizational adaptation in military settings. They have manipulated experimental conditions to explore the degradation of organizational performance in a fine-grained way – monitoring the nature and quantity of communication and the perceived workload of individuals, as well as measures of task performance (Diedrich et al., 2003; Entin et al., 2003). They sought to identify how organizations cope with incongruence, and in particular to identify the conditions that might be salient enough to cause

1) See www.aptima.com for more information.

organizations to alter not only their strategies, but also their structures. Their data pointed toward a set of indicators that have the potential to yield diagnostic information regarding congruence early in a mission scenario, including performance measures (composite variables such as mission tasks processed, latency, and accuracy), team coordination processes (e.g., communication patterns), and workload levels (e.g., subjective assessments).

9.5

Organization and Autonomy

Many applications require a set of agents that are individually autonomous (in the sense that each agent determines its actions based on its own state and the state of the environment, without explicit external command) but corporately structured. Traditionally, agent autonomy is viewed as a property of some piece of software, that is, one of the features that the software must possess to be considered an agent. This descriptive property can also be interpreted as a design requirement: you can build an agent in any way you want, but it must exhibit the autonomy property in the end (Weigand and Dignum, 2004). Abdelkader (2003) discusses two interpretations of autonomy: self-governance (the agent is steered in selecting what goals have to be achieved by a set of motivations) and independence (the agent is independent from other agents) and he prefers the former as independence is not a sufficient criteria. Carabelea et. al (2004) identify five forms of autonomy: user autonomy, social autonomy, norm autonomy, self-autonomy, and environmental autonomy. They rightly state that autonomy is a relational property [in line with the work by Castelfranchi (1994)]. That is, “X is autonomous *from* (other agent or object) Y *for p* (the object of autonomy)” is defined further as: “the behavior of X regarding p cannot be imposed by Y”. They call this an external perspective, which should be supplemented by an internal perspective. They further suggest that one should be able to identify the parts of the agent that give it an autonomous character (e.g., a goal maintenance module).

We agree that autonomy is not the same as independence. If autonomy is interpreted as independence, then the whole idea of agent societies is doomed to failure. After all, what makes a society interesting, for its members or for others, is that there are certain dependencies (Castelfranchi, 1994). What is often overlooked in the discussions is that these dependencies work two ways: not only the agent is dependent on its environment, but also the environment (other agents) will depend on the agent to some extent. In other words, the agent will perform a certain role, a function with an added value. Rather than *assuming* autonomy to be a required property of agents, and from there infer some architecture that would guarantee this property, we have proposed a different approach: namely, to *require* autonomy as an assumed property of agents, and to infer from there some architecture that respects this property (Weigand and Dignum, 2004).

There is a growing recognition that a combination of organization structure and individual autonomy is often necessary. Selznick states that “*all formal organiza-*

tions [follow] ordered structures and stated goals ... and will develop an informal structure within the organization which will reflect the spontaneous efforts of individuals and subgroups to control ... the [organizational] environment ... Informal structures are indispensable to ... the process of organizational control ... and stability" (Selznick, 1953). Effective MASs must therefore yield coordinated behavior from individually autonomous actions (Parunak and Brueckner, 2001). That is, on the one hand, "pure individualism", as reflected by the principle of individual rationality assumed by emergent models, is often not a good choice as a guideline for the decision making of autonomous agents that participate in MAS (Jennings and Campos, 1997). Individual rationality is for instance insufficient for describing a range of desirable social behavior such as being helpful or doing things for the greater good. On the other hand, models that limit the action of its agents to the strict realization of predetermined protocols are not agile enough and, furthermore, do not take full advantage of the potential of participating agents. Already in 1993, Wellman noted that "*combining individual rationality with laws of social interaction provides perhaps the most natural approach to generalizing the Knowledge Level analysis idea to distributed computations*" (Wellman, 1993). In Weigand et al. (2003), we have introduced two requirements for MAS models that reconcile organization and autonomy:

- The *internal autonomy requirement* enables the design of open societies, with heterogeneous participants, by imposing that social structure is represented independently of the internal architecture of agents.
- The *collaboration autonomy requirement* enables the design of evolving societies by indicating that social activity is specified without fixing a priori all interaction structures and protocols.

These requirements reflect an organizational design perspective, that is, they should be taken as guidelines for the design of MAS models. From an agent perspective, it is also necessary to consider the influence of social interaction on the behavior of independent agents. Ossowski (1999) proposes *structural functionalism* as a model of how society, or social structure, "molds" individual behavior so as to achieve globally functional behavior. In structural functionalism theory, social structure is seen as "*an arrangement of persons in institutionally controlled or defined relationships*" (Radcliffe-Brown, 1952). Social relations arise either from person to person or between social roles (e.g., king and subjects). It exists "*between two or more individual organisms when there is some adjustment of their respective interests, by convergence of interest, or by limitation of conflict that might arise from divergence of interests*".

There are two ideas in structural functionalism that are of special interest for our work. On the one hand is the *biasing effect* of social structure: Norms and social institutions impose external constraints on an agent's freedom of choice. They are primitive notions, not reducible to any other, that actually influence the behavior of a "socialized" autonomous agent. Still, they are not assumed to completely determine agent behavior, but just bias self-interested action in a certain ("socially desired") direction. On the other hand is the *teleological character* of social structure: Seen from the outside, social structure induces the coordination of the behavior of

autonomous agents toward a global function of society. From an agent's perspective social structure induces a specific form of cooperation in the sense of mutual adjustment of individual action.

9.6 Reorganization

One of the main reasons for having organizations is to achieve stability. However, environmental changes and natural system evolution (e. g., population changes) require the adaptation of organizational structures. Reorganization is the answer to changes in the environment. Organizational studies often relate reorganization to flexibility. Flexibility can be defined as the "*ability to do something other than that which was originally intended*" (Evans, 1991). As reorganization is opposed to stability, the key question is then: under what conditions is it better to reorganize, knowing that stability will be (momentarily) diminished, and when to maintain stability, even if that means loss of response success? In order to answer this question, it is necessary to define the *utility* of an organization. Furthermore, we also need to consider the autonomous behavior of the entities that form the organization.

9.6.1 Organizational Utility

Organizational success means the organization's ability to bring all its information and assets to bear and the ability to recognize and take advantage of fleeting opportunities. Success is one way to measure the utility of a system. Reorganization is therefore desirable if it leads to increased utility of the system. That is, the reorganized instance should perform better in some sense than the original instance. Given the assumption of agent autonomy, it is also necessary to define agent utility, as each agent should, in principle, be able to determine whether a reorganization has resulted in increased utility for the agent itself. Utility is thus evaluated differently from the perspectives of the society and of the agents.

Society Utility We define the utility of an organization based on organizational properties:

- *Goal success*: how well are global objectives met?
- *Interaction success*: how often do interactions result in the desired aim?
- *Role success*: how often do enacting agents realize role goals?
- *Structure success*: how well are global objectives achieved in an organizational structure?

For example, a given combination of structure and population is said to be successful if the overall success of the organization is higher in that situation than for others. This is an example of a reorganization at the behavioral level. At the structural level, reorganization implies a change of structure. Society utility depends

also on the cost of the reorganization. That is, any function to measure organization utility must take into account both the success of a given structure and the cost of any change needed to achieve that structure from the current situation (Glasser and Morignot, 1997).

Agent Utility is different for each agent, taking into account issues such as its own goals, resource production, and consumption. Basically, we can assume that rational agents will participate in a society if their individual utility increases. Furthermore, different social attitudes will result in different evaluations of individual utility. That is, the utility function of a social agent may take into account some measure of social utility, whereas for a selfish agent only individual concerns matter.

From the perspective of the individual agents, their participation in an organization also depends on utility factors. Utility is, however, appreciated differently from the perspectives of the society and of the agents. On the one hand, the organization will only admit an agent if the overall utility of the society increases (Glasser and Morignot, 1997). On the other hand, assuming rational agents, the agent will only join an organization if its own utility increases.

9.6.2

Organizational Change

In early work in reorganization, restructuring was only possible in the initialization phase of the system. During the actual problem solving phase, the structure was fixed. More dynamic approaches to reorganization are still mostly related to changes in the *behavior* of the organization. That is, reorganization affects the current population of agents in the system, both at the social (i.e., interactions and relationships) (Carley and Gasser, 1999) as well as the individual level (Hannebauer, 2002). Existing implementations of organizational adaptation include approaches based on load balancing or dynamic task allocation. The latter is often the case in organizational self-design in emergent systems that, for example, include composition and decomposition primitives that allow for dynamic variation of the organizational *structure* (macroarchitecture) while the system population (microarchitecture) remains the same (So and Durfee, 1993). Another common approach is dynamic participation, in which agent interaction with the organization is modeled as the enactment of some roles, and adaptation occurs as agents move in and out of those roles (Cavedon and Sonenberg, 1998; Dignum, 2004; Glasser and Morignot, 1997; Tambe, 1997). However, few of these systems allow agents to change the problem solving framework of the system itself (Barber and Martin, 2001). Basically, reorganization is a response to two different stimuli: a reaction to (local) changes in the environment or as the means to implement modified overall intentions or strategies.

Based on the above considerations, we have separated out the following reorganization aspects (Dignum et al., 2004, 2005):

Behavioral change: Change at the behavioral level, that is, the organizational structure remains the same, but the behavior of agents enacting organizational roles change. Examples are when agents join or leave the society, when they change between existing roles, or when their characteristics change (e.g., more or less consumption or production of some resources). It does not affect future enactments and therefore there is no need for organizational memory.

Role-based control: One or more roles in the system are empowered to change enactment plans and/or operational objectives. Meta communication is not really necessary, since for the other roles change is “imposed”.

Shared control: Agents must be able to sense changes in the environment and evaluate their consequences for the current operation. Behavioral change occurs through consensus between all agents.

Structural change: Aims at accommodating long-term changes, such as new situations or objectives. Structural change influences the behavior of the current but also of future social instantiations. Examples of structural change are adding, deleting, or modifying structural elements (e.g., roles, dependencies, norms, ontologies, communication primitives). Change at the social level implies a need for social-level learning. That is, by keeping an organizational memory, society itself can reflect on the difference between desired and actual behavior and decide on social-level changes (roles, norms, etc.).

In human societies, reorganization maneuvers have both a temporal and an intentional aspect. The timing of reorganization can be either *proactive* – preparing in advance for an unpredictable future change – or *reactive* – making adjustments after an event has occurred. The intentional aspect of a reorganization may be *offensive*, in which case the organization aims at gaining a competitive advantage, or *defensive*, aiming at organizational survival.

Proactive situations often imply a modification of behavior: agents dynamically change their behavior, that is, the way they enact their roles can change the patterns of interaction, or even the definition of the roles or their relationships. That is, the expected effects of change can be evaluated without drastic structural changes. In this way, organizations can test “what-if” situations and reason whether a more permanent, structural change should be required. Reactive reorganization will often result in modifications to the structure of the organization.

Another dimension of the reorganization problem concerns the ways in which the reorganization decision is taken, that is, who has the authority to take a decision to reorganize, and how the decision is conveyed and implemented. For example, in distributed decision-making situations it may be that all roles are collectively responsible for a change decision, whereas in other situations (for example those typified by military structures such as C3 (Tidhar and Sonenberg, 2003) (command, control, and communications) different roles may have the authority to effect changes at different levels.

The ability to reason on the timeliness of reorganization (i.e., proactive, reactive) is to a great extent dependent on the capabilities of the agents enacting organizational roles. Proactive reorganization requires high-level reasoning capabilities.

That is, in order to make a proactive reorganization decision, agents must be endowed with mechanisms to reason and evaluate current and desired behavior and utility. Reactive reorganization, on the other hand, only requires agents to be able to sense and react to environmental events, and therefore simpler agents are sufficient.

Furthermore, reorganization decisions can be evaluated in terms of timing (reactive or proactive) and intention (defensive or offensive) (Evans, 1991). Together, these considerations form the five Ws of reorganization, as follows:

What – the aspects of an organization that are to be reorganized.

- Behavior: change in the individual characteristics of the current population, as a response to environmental changes
- Structure: change in the global characteristics of the organization, as a response to changes in intent or strategy

Who – authority to take reorganization decisions, how decisions are taken:

- Directive, role-based decision making
- Collaborative, consensus-based decision making

When – the timing, when should the reorganization occur:

- Proactive, preparing in advance for an unpredictable future change
- Reactive, making adjustments after an event has occurred

Why – the strategic reasons for reorganization

- Offensive, aimed at gaining a competitive advantage
- Defensive, aimed at organizational survival

Whether – the threshold for reorganization, when the fit is so bad that reorganization is likely to be beneficial

- High threshold, stability is seen as more desirable than flexibility.
- Low threshold, flexibility is seen as more desirable than stability.

9.7

Organizational Design

In order to investigate the reorganization of organizations we first have to design organizations using the elements developed in the previous sections. The Language for Agent Organization, LAO, described in Dignum and Dignum (2007), is a formalization of the concepts introduced in Section 9.2 that enable the specification and analysis of organization models. Based on LAO, we have developed an ontology of organizations, OntoOrg (Dignum and Tick, 2007). By using these concepts as first-class modeling entities (Miles et al., 2003) we allow for the specification of open systems and can describe both emergent and designed organizations. A similar modeling approach has been advocated by Ferber and Gutknecht (1998).

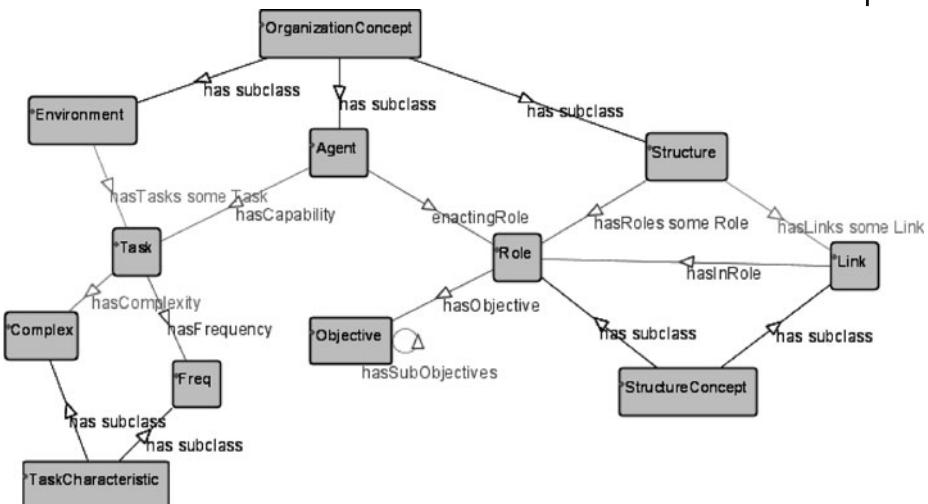


Fig. 9.1 An excerpt of OntoOrg ontology.

The OntoOrg ontology implements LAO concepts and extends it with the representation of organizational roles and role-enactment relations. The three main concepts of LAO (AGENT, ENVIRONMENT, and STRUCTURE) are structured hierarchically as subclasses of the class ORGANIZATIONCONCEPT, as depicted in Figure 9.1. These concepts are further specified and related using class slots and properties. Organization STRUCTURE is described by two concepts: ROLE and LINK. ROLES are characterized by their OBJECTIVES and are enacted by AGENTS. A LINK defines a dependency between two roles and indicates that tasks can be passed from one to the other role. The LINK concept has two slots: ISINROLE, indicating the depender role, and ISOUTROLE, indicating the dependee role. Role enactment by agents is defined by the relation ENACTINGROLE between the classes AGENT and ROLE. Furthermore, agents have capabilities, represented by the slot HASCAPABILITY, related to the tasks in the domain. TASKS are part of the ENVIRONMENT and have the characteristics FREQUENCY and COMPLEXITY.

We have chosen to implement OntoOrg using OWL²⁾ due to its reasoning capabilities and tool support. OWL explicitly allows for inheritance and other logical relationships, which makes OWL models more powerful than SQL schemas. Furthermore, OWL models are RDF models, which can be encoded as XML. Since it is technically easy to share XML documents between software applications, regardless of the programming language and operating system platform, it is also easy to share OWL models. OWL provides a model for refinement of specifications, for example, by intersecting with further conditions or by explicit subclass relations. Because OWL is basically a description logic (Antoniou and van Harmelen, 2004), disambiguation can be performed automatically by an OWL-DL reasoner, which

2) Web Ontology Language, <http://www.w3.org/2004/owl/>.

can also make relations between concepts explicit through subsumption (Dobson, 2006).

9.7.1

Designing Organizational Simulations

Based on OntoOrg, we have developed the *Ontology-based Organization Simulator (OOS)*, which supports developing and deploying agent simulations. OOS addresses a number of current needs of simulation developers, including model composition at different levels of abstraction and comparison and composableility of simulation models. In particular, it provides a framework for assembling simulations from a library of organizational structures, environmental components, and agent behaviors.

The OOS approach combines formal specifications with ontology-driven translation generation techniques to facilitate robust simulation analysis and support semantic modeling and interoperability. The structural change of an organization is achieved by modifying slots or adding instances to the ontology. On the other hand, behavioral change is achieved by loading agents with different behaviors into the simulation. By using an independent agent platform modifications of agent behavior can be made independently of the organizational model. Furthermore, the agent platform enables one to change the whole agent population at once, so that the same organization can be simulated with a different set of agents. That is, organizational structure and agent behavior can be changed separately. The current OOS prototype described in Tick (2007) enables the conversion of instances of the OntoOrg ontology into a custom JAVA-based format and semiautomatic generation of models which can be used by the Repast³⁾ simulation environment to visualize and run the simulations. In addition, OOS provides a library of agent behaviors based on the 2APL agent platform,⁴⁾ which allows one to use software agents with cognitive capabilities within the Repast environment.

Creating a simulation in OOS consists of the following four steps:

1. Describe the application domain using OntoOrg.
2. Convert the organization ontology instance into JAVA format in order to be accessible to the Repast simulator environment.
3. Load the converted organizational model into the simulator.
4. Choose and load agent models from the library into the simulator.

In short, the transformation translates the organizational ontology classes (in RDF) into JAVA classes.⁵⁾ For each ontology class slot, a (private) variable is added to the corresponding JAVA class, and methods are generated to set and get slot values. At runtime, these methods are executed together with information about the RDF

3) <http://repast.sourceforge.net/>.

and is based on the OntoJava converter:

4) <http://www.cs.uu.nl/2apl/>.

<http://www.aifb.uni-karlsruhe.de/WBS/aeb/>

5) This transformation has been developed by
L.D. Acay at the University of Melbourne

ontojava.

organization instances to create JAVA objects representing the organization being simulated.

9.7.2

Application Scenario

In this section, we describe a small scenario in order to illustrate the use of the OOS simulation framework to organizational design. We consider a builders organization employing painters, tilers, and other workers. Instances in OntoOrg represent specific organizations and their components. Agent capabilities, for example, C_a *sand-door*, representing the fact that agent a has the capability *sand-door*, are represented as triple $\langle a, \text{HASCAPABILITY}, \text{sand-door} \rangle$. In this scenario, we assume a hierarchical organizational structure consisting of one manager agent and 12 worker agents, where the manager has the capability of assigning tasks to the workers. Two different instances of this organization were specified:

- **Specialists (*Spec*):** each worker can only perform tasks of one type (four workers for each type). The time to complete a task is calculated randomly as $T_s \in [50, 100]$.
- **Generalists (*Gen*):** all workers can perform all tasks; however, they take longer to complete a task: $T_g = 2 * T_s - 25$.

A fragment of the OntoOrg representation of a builders organization illustrating this example is depicted in Figure 9.2, which shows part of the instantiation of OntoOrg for this domain.

In order to test the congruence of an organization and environment, our approach was to design two sufficiently different environments and two organizations that exploit the differences between environments. The idea is that if the objectives are incongruent with the organization, that is, in the current organizational situation objectives cannot be achieved, organizations will need to reorganize in order to fulfill their missions. Recall that an organization O is incongruent with a set of desires or goals if it does not possess the capability to realize those objectives. For the builders organization, we define the following types of environments describing the tasks to be done. In this example we consider three types of tasks: *paint-door*, *paper-wall*, *tile-floor*, and we set the frequency at 25% (that is, there is a 25% chance that a new task will be requested).

- **Heterogeneous case (*Het*):** all task types occur with the same probability.
- **Homogeneous case (*Hom*):** mostly one type of task: tasks of *paint-door* type occur in 80% of the cases, other tasks with 10% probability each.

The idea is that a manager receives tasks to be executed and distributes those to a worker who has the capability to perform those tasks and is currently free. If no capable agent is free, the manager keeps the tasks in a waiting list. There is a penalty for keeping tasks in the waiting list, resulting in less profit. Every completed task gives a profit of x to the organization. For each waiting task the organization loses $x/10$ for each time slot the task is waiting. In total, four different scenarios are

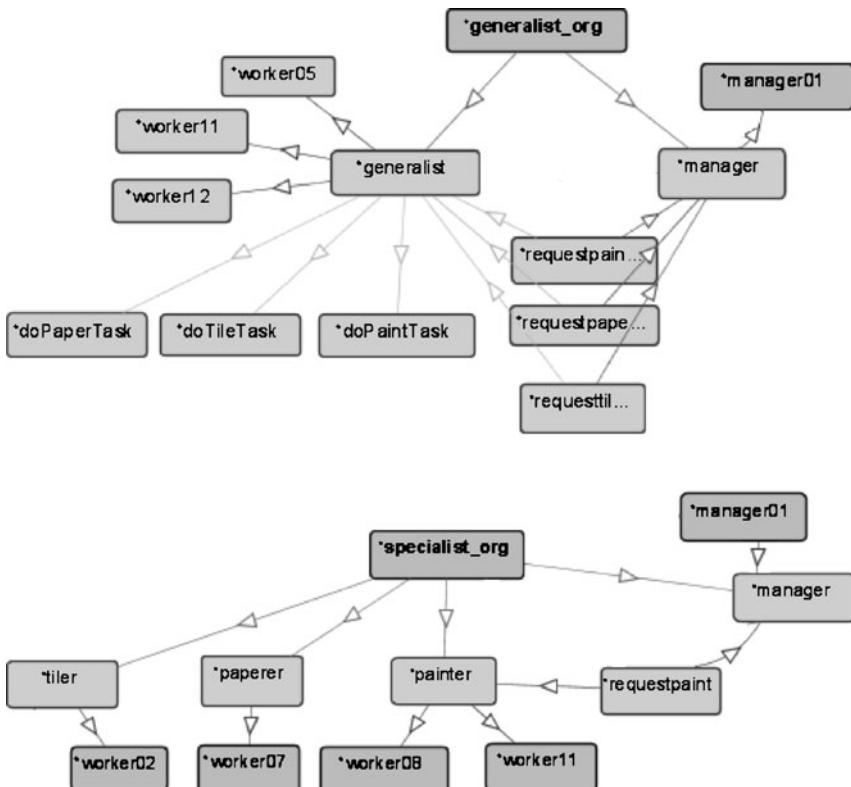


Fig. 9.2 Generalist and specialist organizational instances.

obtained, as follows: Het-Gen, Het-Spec, Hom-Gen, Hom-Spec. We used OOS to generate Repast simulations of the four scenarios, each corresponding to a different instantiation of the organization ontology.

Each scenario (which had a length of 400 time units) was run 10 times, resulting in 40 different simulations. For each scenario, we calculated the number of completed tasks, the number of waiting tasks, and the average completion length. Figure 9.3 shows the results obtained.

As can be expected, the performance of generalist organizations is not influenced by the type of environment. Specialist organizations do not perform well in homogeneous environments (workers capable of *paint-door* are too busy, other workers mostly idle). In such cases, reorganization should be considered. We provided manager agents with the capability to check organizational performance and adapt accordingly. In the experiment, manager agents of specialist organizations in a homogeneous environment monitor the number of waiting tasks and perform a reorganization action when that number reaches a given threshold. Specifically, when the manager notices that there are ten or more waiting tasks, he/she will add

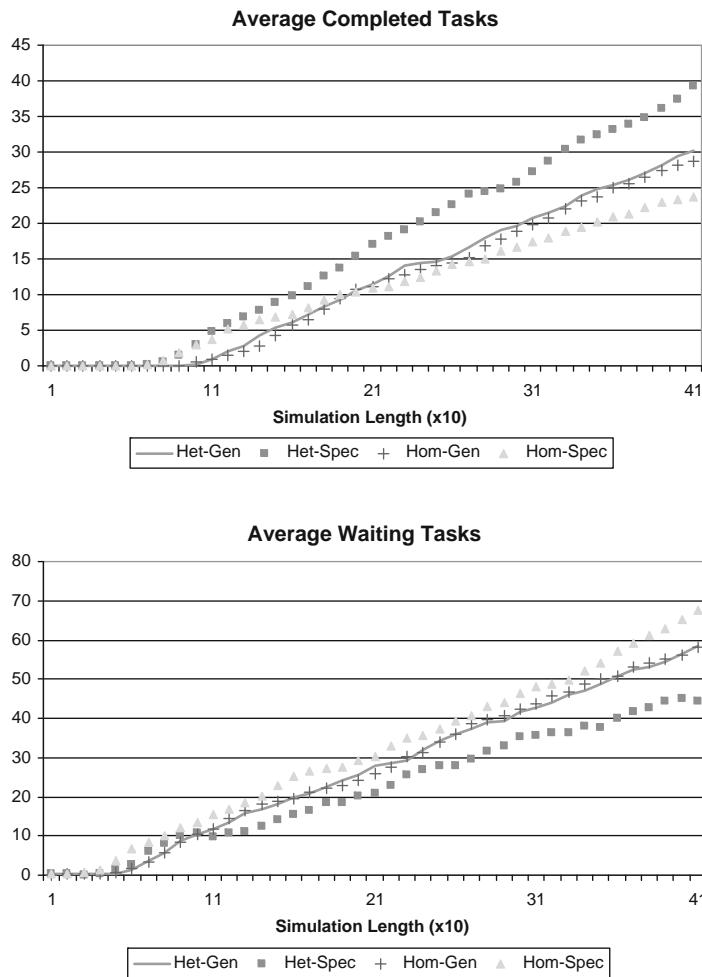


Fig. 9.3 Organizational performance.

one agent trained to perform painting tasks to the organization. Reorganization cost, that is, the cost of training one agent, is set at $x * 3$, where x is the task profit.

Figure 9.4 shows the difference in performance between organization Hom-Spec and organization Hom-Spec-Reorg endowed with such reorganization capabilities, for task completion times $T_s \in [25, 50]$ and task frequency 25%. As can be seen, reorganization results in fewer waiting tasks and therefore higher profit due to fewer penalties. Depending on the values for profit, penalties, and reorganization costs, simulations can be used to determine an effective threshold.

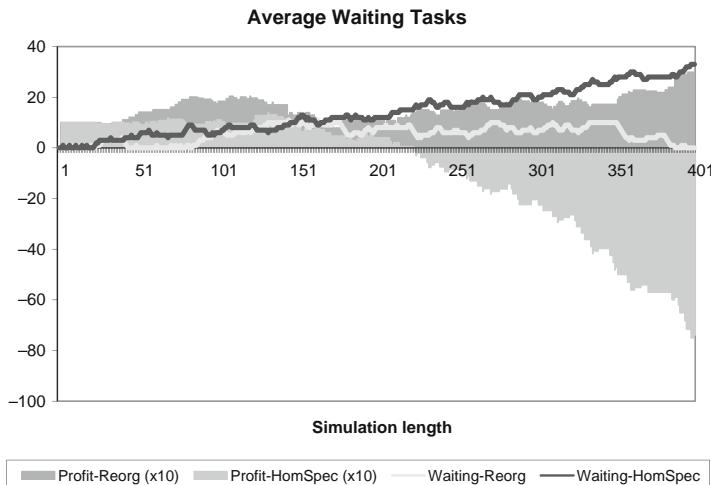


Fig. 9.4 Reorganization effect.

9.8

Understanding Simulation of Reorganization

Simulations are general-purpose techniques that achieve their usefulness through abstraction. Different domains can be modeled in a similar way using abstract concepts and equations. The success of simulation approaches is demonstrated by the wide use of applications across many research areas (e.g., social sciences, biology, economics, and management sciences) (Railsback et al., 2006). However, the weakness of this paradigm lies in the following observation (Miller and Baramidze, 2005): mapping between the real world and an abstract model resides largely in the minds of the developers; high-fidelity, multifaceted, multilevel modeling is difficult to achieve, and reengineering of models from components is limited. Simulations are generally iterative in their development. One develops a model, simulates it, learns from the simulation, revises the model, and continues the iterations until an adequate level of understanding is developed. The quality of the simulation is thus largely dependent on the quality of the model. Especially in the case where the investigation of large numbers of individuals and complex environments is the aim, it is almost always necessary to turn to simulation. Agent-based simulations for organizations have been proposed on the basis of the following arguments (Carley and Gasser, 1999):

- In real situations, the nonlinear behavior of individuals and groups cannot be ignored or eliminated.
- Differential equations do not deal with the differences of discrete items in the organization such as persons, tasks, and groups.
- Agents in an organization are autonomous, acting according to their goals and capabilities.

As discussed in the previous sections, the efficiency of organizations is usually measured in terms of performance, or the degree to which goals are achieved. Various organizational simulation projects have studied the influence of different factors on organizational efficiency using different techniques and aiming at proving different hypotheses. Depending on the situation, this can be the turnover of the organization, the relative frequency with which the right decision is made, the ratio of survival, and so on. Performance, or efficiency, is potentially affected by the organization's design, the cognitive or information processing capabilities of the agents, the operating conditions, the task, and the task environment faced by the organization (Carley and Svoboda, 1996). Reorganization can then be seen as a means to achieve better performance, or create a more efficient organization, by changing one or more of the factors that affect performance.

Furthermore, research on formal models of organizations have demonstrated that (Carley and Svoboda, 1996) (a) there is not one optimal organizational design, (b) structural constraints, task constraints, cognitive constraints, and other environmental conditions can dictate organizational outcomes. Reorganization is thus a complex process that involves many different factors. Both in the social sciences and in computer science many simulation studies have been conducted that aim to increase understanding of reorganization in human and agent organizations, respectively.

9.8.1

Reorganization Dimensions

The organization model proposed in Section 9.2 and the considerations described above suggest a 3-D space in which the results of different organizational simulations can be placed and eventually compared. Axes in this 3-D space correspond to the groups of factors listed above, that is, agent capabilities, organizational structure, and environmental conditions. Figure 9.5 shows the different projections in the reorganization space. Each point in the space represents a specific organization instance (that is, with a given structure, task, agent capabilities, and operating conditions). For each point the performance can be calculated. In this perspective, reorganization means a move into another point in a space that has a better performance. In order to cope with complexity, such a movement in existing simulation studies is done on a 2-D projection of the overall space, which means that one set of factors is kept fixed in that particular simulation. Existing studies on reorganization, however, relate to simplified spaces, typically 2-D projections on this 3-D space. In what follows we briefly introduce different existing simulation projects and discuss its place in this 3-D space.

9.8.2

Analyzing Simulation Case Studies

In this section, we present three different existing simulation studies and discuss how these studies fit the 3-D model introduced above. The choice for these three

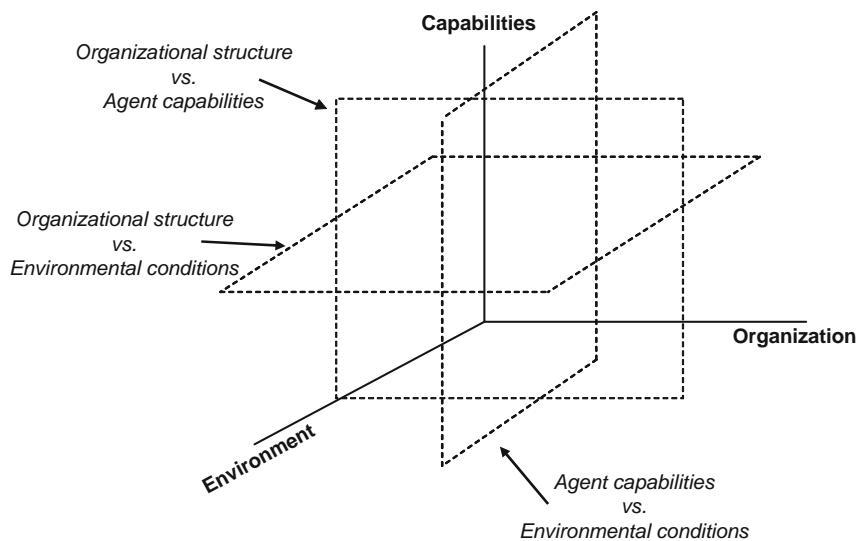


Fig. 9.5 Dimensions of reorganization.

cases is intended to illustrate different cross sections of the model and should by no means be taken as an extensive analysis of existing simulation studies.

9.8.2.1 Adaptive Decision Making (Barber and Martin, 2001)

Description of the Simulation The research question in this project was to determine whether adaptive decision making frameworks provide better results than statically fixed decision making protocols (DMF) or random changes in decision making policies. The experiments were performed in the domain of naval radar frequency management. Each agent in this domain controlled one given naval radar and attempted to minimize the overall radar interference in the environment. Agents experience different types of operating conditions (geographical position changes, communication may fail, positioning sensor may report wrong information, etc.). On the other hand, given the aims of the project, agents were organized according to different decision making frameworks. Three DMFs were considered:

- Locally Autonomous (LA): each agent makes decisions alone to attempt to resolve interference.
- Master/Command Driven (M/CD): A master agent makes all decisions and the command-driven agents wait for frequency assignments (this implements a hierarchical control relation).
- Consensus (CN): All agents determine and transmit frequency assignments for all other agents. A voting process determines which assignments are chosen.

Experimental results, achieved by different runs under fixed operating conditions and fixed DMF, provide empirical evidence to determine which DMF performs best

under which operating conditions. In the second part of the project, the efficiency of reorganization was studied in an environment where operating conditions changed by comparing the results of agents using adaptive DMF against the results of agents that kept to a fixed DMF or to agents that randomly changed their DMF.

Projection on Reorganization Dimensions The project assumes fixed agent capabilities. The different organizational instances are therefore points on the 2-D plane Organization/Environment (cf. Figure 9.5). In the authors' experiments, involving three agents, the environmental characteristics Communication (Comm), Position Sensing (PosSen), Interference Reciprocal (Recip), and Geographical Position (Pos), the points on the Organization/Environment plane are determined by the following values on the axes:

Organization: indicate how many agents follow which DMF strategy, and consequently how their relations are organized. For example, 3LA, or 2M/CD and 1LA.

Environment: indicate the environmental conditions in a particular situation, for example, Comm UP, PosSen DOWN, Recip SYMMETRIC, PosX, and so on.

The simulation determines the performance value for each of these points on the Organization/Environment plane. Given that, in this domain, environmental changes are not controlled by the agent organization, the reorganization entails a change to another value on the Organization axis given an (external) change on the Environment axis.

Figure 9.6 shows the typology of the Organization/Environment plane where the points indicate the empirically determined best organizational structure given

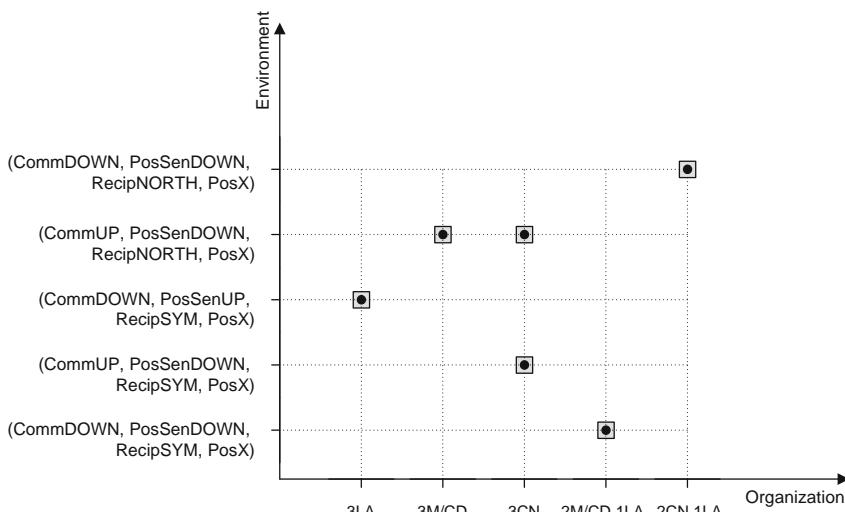


Fig. 9.6 Simulation space of ADMF project.

a certain environmental situation (note that the points shown are merely examples and do not necessarily represent the results determined by the empirical experiments, to which we have no access). The ADMF strategy proposed by the authors is able to determine the best Organization value for each given Environment value. Reorganization according to the ADMF strategy would be represented as a line connecting best structure points corresponding to the changes in Environment values.

9.8.2.2 Cognitively realistic agent models (Sun and Naveh, 2004)

Description of Simulation This project aimed at exploring the interaction between more complex cognitive models of agents and the performance of an organization. In particular, the authors were interested in observing performance at both ends of the learning curve – that is, both after a moderate amount of training and after extensive training. The domain of the experiment is that of classification decision making: agents must determine whether an object can be classified as friendly, hostile, or neutral. Although not directly aimed at studying the reorganization of organizational structures, this work does provide guidance on the interplay between agent capabilities and organizational structures. As an agent model the project uses CLARION (Sun, 2006), an integrative cognitive architecture with a dual representational structure that captures both implicit and explicit learning. The project incorporated different simulation experiments involving differences in cognition and organizational structure:

- Cognition: Performance after only some initial learning was compared to that after a considerable amount of learning, with different sets of cognitive parameters.
- Organization: Team and hierarchies were considered. Further, organizations consisting of identical agents were compared to organizations where one agent was a slower learner and organizations where all agents learned at different rates.

The interest of the researchers was to develop an artificial system that captured human performance both in terms of capabilities and social interaction. Although not directly concerned with reorganization issues, this project does reach some conclusions relating cognitive capabilities and organizational structure. Empirical studies show that teams seem to perform better at the start of a learning project, whereas this trend seems to disappear or reverse toward the end, independently of the cognitive setting.

Projection on Reorganization Dimensions The project assumes a fixed task environment (classification of objects). Performance is measured in terms of the ratio of correctly classified objects. The different organizational instances are therefore points on the 2-D plane Organization/Agent Capabilities (cf. Figure 9.5) as shown in Figure 9.7. Besides different structures, the organizations were also distinguished by the kind of information access: blocked (all agents see all of an ob-

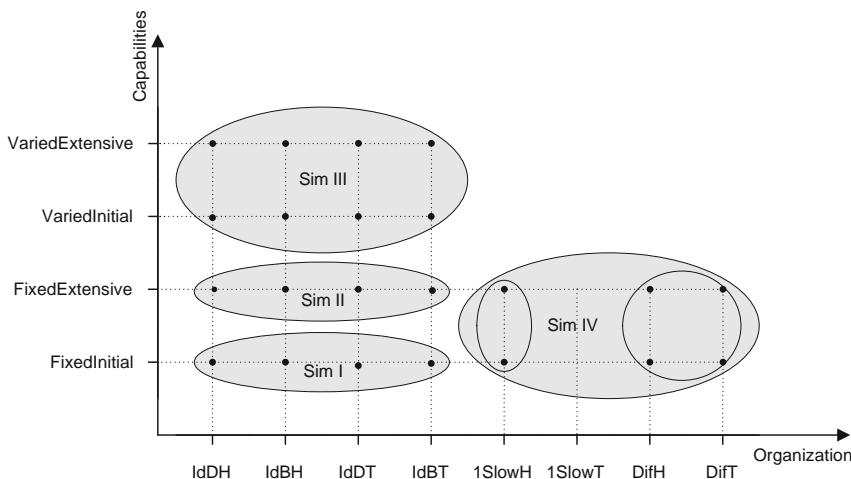


Fig. 9.7 Simulation space of CLARION project.

ject's characteristics) or distributed (each agent sees a different subset of an object's characteristics). In the figure above, points on the Organization axis represent the hierarchical organization of identical agents using distributed access (IdDH), the hierarchical organization of identical agents using blocked access (IdBH), the team organization of identical agents using distributed access (IdDT), the team organization of identical agents using blocked access (IdBT), the hierarchical organization with one slower learning agent (1SlowH), the team organization with one slower learning agent (1SlowT), the hierarchical organization with agents learning at different rates (DifH), and the team organization with agents learning at different rates (DifT). On the Capabilities axis, the figure distinguishes four points corresponding to differences between initial and extensive amounts of learning and variations in cognitive parameters. In reality, the experiments described considered many more differences in cognitive capabilities. Without entering into details of the model used, those differences refer to the rate of learning of the neural network used, the threshold for considering a rule useful, the threshold for generalizing a rule, and so on.

The study suggests that a more cognitively realistic simulation can better capture and predict human performance. These results can be used to identify the best organizational model given the cognitive capabilities of agents and formulate organizational policies relating to the training of novice and experienced personnel. For instance, the simulation showed that agents organized in a team fared much better when using a high generalization threshold (that is, having a cautious approach to formulating new policies). On this basis, such a group of agents may attempt to improve performance by switching to a team organization.

9.8.2.3 Villa (Dignum et. al, 2006)

Description of the Simulation Our aim in designing this simulation was to provide experimental evidence for the reorganization ideas described in this chapter. The simulation is based on a fictive domain where performance of a group of creatures (divided into three groups: Gatherers, Hunters, and Others) is measured by its capability to survive in a more or less hostile environment. The simulation environment, VILLA, was designed to meet the following requirements: (1) be simple enough to enable empirical evaluation of the results, but (2) be complex enough to emulate situations where reorganization really matters. Gatherers and Hunters are responsible for keeping the food stock supplied. Both roles have different characteristics (e.g., Hunters must hunt in groups whose hunting capability increases with the group's size) and the conditions of the environment are set such that it is not guaranteed that the food collection activities of agents are always successful.

Projection on Reorganization Dimensions Given the aims of the VILLA simulation, studying performance of different groups of agents under different environmental conditions, it is clear that one of the axes in this project is the Environment axis. In fact, the project can be decomposed into two different studies, one that measures performance on the Agent Capabilities/Environment plane and the other that measures performance on the Organization/Environment plane. Differences in environmental conditions are specified by the size of the grid, the probability of finding food (low in hostile environments, high in friendly environments), and the level of the initial food stock. Figure 9.8 shows the different types of behaviors analyzed, using a fixed static organizational structure (e.g., 7 Gatherers, 6 Hunters, 4 Others).

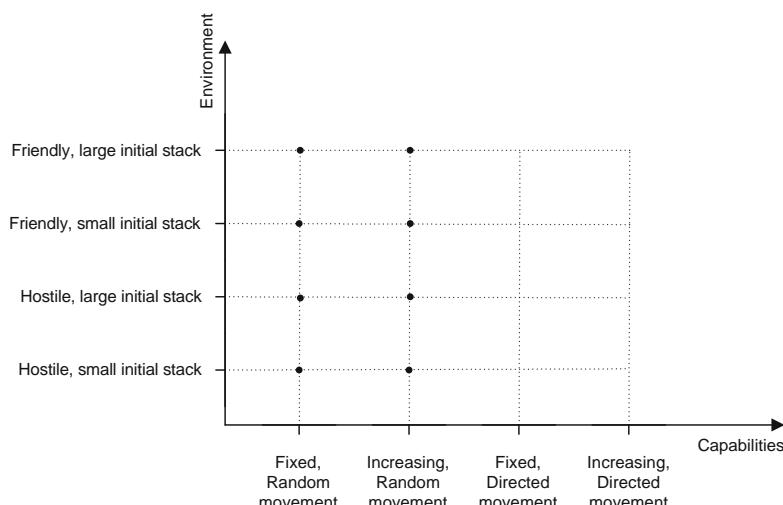


Fig. 9.8 Simulation space of behavior study in VILLA project.

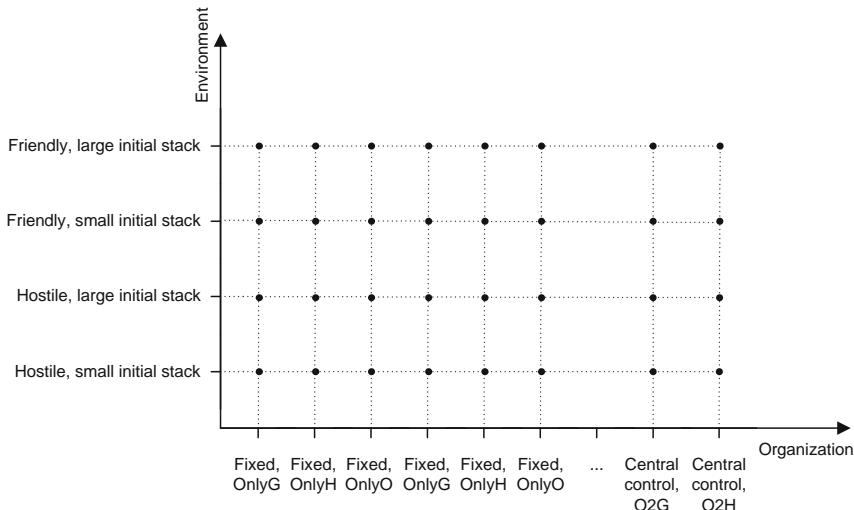


Fig. 9.9 Simulation space of structure study in VILLA project.

The other set of experiments concerned the relation between environmental and organizational structure, depicted in Figure 9.9. Here we analyzed the influence of changing the organization's typology in both static and dynamic settings. In the first setting, different typologies of organizations were tested by changing the relative numbers of Gatherers, Hunters, and Others. In the second setting, a new type of agent was introduced that was able to evaluate the current organizational utility (e.g., measured in terms of number of deaths, or the average health of creatures) and, when that utility fell below a given threshold, could request that an agent playing the role of Other play either the role of Gatherer or that of Hunter.

9.9 Conclusions

Adaptation is needed in order to enable systems to enforce or adapt to changes in the environment. This issue has been discussed by many researchers in both organization theory and in distributed systems, resulting mostly in domain-oriented empirical solutions. The lack, in most cases, of a formal basis makes difficult the development of theories about reorganization, prevents the comparison of approaches and results, and makes difficult the adaptation of models to other domains or situations.

In this chapter, we introduced both a model for organizational design and a conceptual framework to classify organizational simulations. The organizational design model provides a structured and efficient way to deploy many different organizational designs by using an ontology to describe organizational structures, environmental characteristics, and agent capabilities and provides semiautomatic

means to generate simulations from the ontology instances. Moreover, the simulation framework incorporates the cognitive characteristics of agents. The existing OOS prototype is being used for the modeling and analysis of more realistic organizations, in particular, to implement simulations of existing agent systems that simulate organizations and organizational adaptation, which we previously represented using LAO (Dignum and Dignum, 2007; Tick, 2007). The framework is based on a formal language, LAO, that provides a formal, provable representation of organizations with their environment, objectives, and agents in a way that enables one to analyze their partial contributions to the performance of the organization in a changing environment. Moreover, the model includes enough realistic concepts that enable it to represent the more “pragmatic” considerations faced by real organizations. LAO enables the specification and comparison of abstract organizational designs and supports the decision making process on the choice of design. We developed a concrete ontology, OntoOrg, based on LAO that is used for the development of organizational simulations.

We further proposed a conceptual framework that enables the classification of different organization simulation along three axes of a 3-D model including agent capabilities, environmental characteristics, and organizational structure.

The simple scenario presented in Section 9.7 demonstrates the use of simulations to support the process of assessment of organizational performance. By varying the types and capabilities of the agents and the characteristics of the environment, it is possible to analyze the fit of organizational structures and the potential of different reorganization decisions to the performance of the organization. Currently, we are extending the evaluation of the model and the OOS tool to the analysis of more realistic organizations. In parallel, we are also engaged on the development of richer agent architectures that will enable the simulation of human-like decision making mechanisms including cultural, personality, and normative issues.

Acknowledgements

This work was partially done during a sabbatical of Frank and Virginia Dignum at the University of Melbourne. The authors are grateful to L.D. Aca and C. Tick for the development of the OOS prototype. The research of Virginia Dignum is funded by the Netherlands Organization for Scientific Research (NWO), through Veni-grant 639.021.509.

References

- | | |
|---|--|
| Abdelkader, G. (2003) <i>Requirements for achieving software agents autonomy and defining their responsibility</i> . Proceeding of the Autonomy Workshop at AAMAS 2003. | Aldrich, H. and Pfeffer, J. (1976) Environments of organizations. <i>Annual Review of Sociology</i> , 2, 79–105. |
|---|--|

- Antoniou, G. and van Harmelen, F. (2004) *A Semantic Web Primer (Cooperative Information Systems)*, The MIT Press.
- Barber, K.S. and Martin, C.E. (2001) *Dynamic reorganization of decision-making groups*. Proceedings of the 5th Autonomous Agents.
- Burns, T. and Stalker, G. (1961) *The management of innovation*, Tavistock, London.
- Carabelea, C., Boissier, O. and Florea, A. (2004) Autonomy in multi-agent systems: A classification attempt. *Autonomy 2003*, vol. 2969 of *LNAI*, (eds M. Nickles, M. Rovatsoso and G. Weiss), Springer, pp. 103–113.
- Carley, K. and Gasser, L. (1999) Computational organization theory. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, (ed. G. Weiss), The MIT Press, pp. 299–330.
- Carley, K. and Svoboda, D. (1996) Modeling organizational adaptation as a simulated annealing process. *Sociological Methods & Research*, 25 (1), 138–168.
- Castelfranchi, C. (1995) Guarantees for autonomy in cognitive agent architecture, in *ATAL'94*, vol. 980 of *LNAI*, Springer.
- Castelfranchi, C. (2001) Engineering social order. *Engineering Societies in the Agents World*, LNAI 1972, (eds A. Omicini, R. Tolksdorf and F. Zambonelli), Springer, pp. 1–19.
- Cavedon, L. and Sonenberg, L. (1998) *On social commitments, roles and preferred goals*. Proceedings of the 1998 International Conference on MultiAgent Systems (ICMAS98), pp. 80–87.
- Dess, G. and Beard, D. (1984) Dimensions of organizational task environments. *Administrative Science Quarterly*, 29 (1), 52–73.
- Diedrich, F.J., Entin, E.E., Hutchins, S.G., Hocevar, S.P., Rubineau, B. and MacMillan, J. (2003) *When do organizations need to change (part i)? coping with incongruence*. Proc. of Command and Control Research and Technology Symposium.
- Dignum, V. (2004) *A Model for Organizational Interaction: based on Agents, founded in Logic*, SIKS Dissertation Series 2004-1, Utrecht University, PhD Thesis.
- Dignum, V. and Dignum, F. (2007) *Logic for agent organization*. Proceedings of FAMAS@Durham'07.
- Dignum, V., Dignum, F. and Sonenberg, L. (2004) *Towards dynamic organization of agent societies*. Workshop on Coordination in Emergent Agent Societies, ECAI 2004, (ed. G. Vouros), pp. 70–78.
- Dignum, V., Dignum, F. and Sonenberg, L. (2006) Exploring congruence between organizational structure and task performance: A simulation approach. *Coordination, Organization, Institutions and Norms in Agent Systems I*, Proc. ANIREM'05/OOOP'05, vol. 3913 of *LNAI*, (eds O. Boissier et al.), Springer, pp. 213–230.
- Dignum, V. and Tick, C. (2007) *Agent-based analysis of organizations: Formalization and simulation*. Proc. IAT'07, IEEE, pp. 244–247.
- Dignum, V., Dignum, F., Furtado, V., Melo, A. and Sonenberg, L. (2005) *Towards a Simulation Tool for Evaluating Dynamic Reorganization of Agents Societies*. Proceedings of WS. on Socially Inspired Computing, AISB Convention.
- Dobson, G. (2006) *OWL and OWL-S for dependability-explicit service-centric computing*. Service-Oriented Computing: Consequences for Engineering Requirements (SOCCER'06).
- Donaldson, L. (2001) *The Contingency Theory of Organizations*, Sage.
- Duncan, R. (1979) What is the right organizational structure: Decision tree analysis provides the answer. *Organizational Dynamics*, Winter, pp. 59–80.
- Entin, E.E., Diedrich, F.J., Kleinman, D.L., Kemple, W.G., Hocevar, S.G., Rubineau, B. and Serfaty, D. (2003) *When do organizations need to change (part II)? incongruence in action*. Command and Control Research and Technology Symposium.
- Entin, E.E. and Serfaty, D. (1999) Adaptive team coordination. *Journal of Human Factors*, 41, 321–325.
- Evans, J. (1991) Strategic flexibility for high technology manoeuvres: A conceptual framework. *Journal of Management Studies*.
- Ferber, J. and Gutknecht, O. (1998) *A meta-model for the analysis and design of organizations in multi-agent systems*. ICMAS'98, IEEE Computer Society, pp. 128–135.
- Ferber, J., Gutknecht, O. and Michel, F. (2003) From agents to organizations: An organizational view of multi-agent systems, in

- AOSE: Agent-Oriented Software Engineering IV*, vol. 2935 of *LNCS*, Springer-Verlag.
- Galbraith, J. (1977) *Organization Design*, Addison-Wesley.
- Glasser, N. and Morignot, P. (1997) The reorganization of societies of autonomous agents, in *MAAMAW*, pp. 98–111.
- Hall, R. (1991) *Organizations: Structures, Processes, and Outcomes*, Prentice Hall.
- Hannebauer, M. (2002) *Autonomous Dynamic Reconfiguration in Multi-Agent Systems*, vol. 2427 of *LNAI*, Springer-Verlag.
- Hübner, J., Sichman, J. and Boissier, O. (2006) S-moise+: A middleware for developing organised multi-agent systems. *COIN I*, vol. 3913 of *LNAI*, (eds O. Boissier et al.), Springer, pp. 64–78.
- Jennings, N.R. and Campos, J.R. (1997) Towards a social level characterisation of socially responsible agents. *IEEE Proceedings on Software Engineering*, **144** (1), 11–25.
- Malyankar, R. (1999) *A pattern template for intelligent agent systems*. Agents'99 Workshop on Agent-Based Decision Support for Managing the Internet-Enabled Supply Chain.
- Miles, S., Joy, M. and Luck, M. (2003) Towards a methodology for coordination mechanism selection in open systems. *Engineering Societies in the Agents World III*, LNAI 2577, (eds P. Petta, R. Tolksdorf and F. Zambonelli), Springer-Verlag.
- Miller, J. and Baramidze, G. (2005) *Simulation and the semantic web*. WSC '05: Proceedings of the 37th Winter Simulation Conference, pp. 2371–2377.
- Mintzberg, H. (1993) *Structures in Fives: Designing Effective Organizations*, Prentice Hall.
- Mussio, P. (1998) *Emergent evolution of cooperative structures*. Workshop on Human and Machine Perception: Emergence, Attention and Creativity.
- Nair, R., Tambe, M. and Marsella, S. (2003) Role allocation and reallocation in multiagent teams: Towards a practical analysis, in *AAMAS 2003*.
- Ossowski, S. (1999) *Co-ordination in Artificial Agent Societies, Social Structure and Its Implications for Autonomous Problem-Solving Agents*, LNCS 1535, Springer-Verlag.
- Radcliffe-Brown, A.R. (1952) *Structure and Function in Primitive Society*, Cohen & West.
- Railsback, S., Lytinen, S. and Jackson, S. (2006) Agent-based simulation platforms: review and development recommendations. *Research in Organisational Behavior*, **28**, 609–623.
- Roberts, J. (2004) *The Modern Firm. Organizational Design for Performance and Growth*, Oxford University Press.
- Selznick, P. (1953) *TVA and the Grass Roots: A Study of Politics and Organization*, University of California Press.
- Sierhuis, M. (2001) *Modeling and Simulating Work Practice*, SIKS Dissertation Series 2001–10, University of Amsterdam, PhD Thesis.
- So, Y. and Durfee, E. (1993) An organizational self-design model for organizational change. In: AAAI Technical Report WS 93-03, p. 8–15.
- So, Y. and Durfee, E. (1998) Designing organizations for computational agents, *Simulating Organizations*, (eds K. Carley, M.J. Pritula and L. Gasser), pp. 47–64.
- Sun, R. (2006) The clarion cognitive architecture: Extending cognitive modeling to social simulation. *Cognition and Multi-Agent Interaction*, (ed. R. Sun), Cambridge University Press.
- Sun, R. and Naveh, I. (2004) Simulating organizational decision-making using a cognitively realistic model. *Journal of Artificial Societies and Social Simulation*, **7** (3).
- Tambe, M. (1997) Towards flexible teamwork. *Journal of Artificial Intelligence Research*, **7**, 83–124.
- Tick, C. (2007) *Organization structure analysis by simulation*, Master's thesis, Utrecht University.
- Tidhar, G. and Sonenberg, L. (2003) *Engineering organization-oriented systems*. Proceedings of Workshop on Autonomy, Delegation and Control: From Inter-Agent to Organizations and Institutions, (ed. H. Hexmoor), AAMAS.
- Valeto, G., Kaiser, G. and Gaurav S.Kc. (2001) *A mobile agent approach to process-based dynamic adaptation of complex software systems*. 8th European Workshop on Software Process Technology, pp. 102–116.

- Van Dyke Parunak, H. and Brueckner, S. (2001) Entropy and self-organization in multi-agent systems, in *Agents'01*, pp. 124–130.
- Weigand, H. and Dignum, V. (2004) I am autonomous, you are autonomous. *Agents and Computational Autonomy*, LNAI 2969, (eds M. Nickles, M. Rovatsos and G. Weiss), Springer, pp. 227–236.
- Weigand, H., Dignum, V., Meyer, J.J. and Dignum, F. (2003) Specification by refinement and agreement: designing agent interaction using landmarks and contracts. *Engineering Societies in the Agents World III*, LNAI 2577, (eds P. Petta, R. Tolksdorf and F. Zambonelli), Springer-Verlag.
- Wellman, M. (1993) A market-oriented programming environment and its application to distributed multi-commodity flow problems. *Journal of Artificial Intelligence Research*.

10

Programming Languages, Environments, and Tools for Agent-directed Simulation

Yu Zhang, Mark Lewis, and Maarten Sierhuis

10.1

Introduction

Agents are autonomous computer programs capable of independent action in environments that are typically dynamic and unpredictable (Wooldridge, 2002). The study of multiagent systems (MAS) focuses on systems in which many agents interact with each other (Sycara, 1998). Agent-directed simulation (ADS) with MASs is comprehensive in the integration of agent and simulation technology by including models that use agents to develop domain-specific simulations (this is often referred to as agent-based modeling) and by also including the use of agent technology to develop simulation techniques and toolkits that are subsequently applied, either with or without agents (Ören et al., 2000). ADS can serve as a viable tool to help researchers from many disciplines re-create and predict the actions of complex phenomena, especially for domains that require a long time to evolve or require exposing real people to dangers (Pan et al., 2007) such as disease transmission (Carley et al., 2006), corporation management (Schreiber and Carley, 2004), and military operations (Carley and Schreiber, 2002).

For the simulation community, ADS approaches provide a new way of distributing and managing large-scale simulations, and this presents a challenge requiring the development of new theory and techniques in simulation. Parunak et al. (1998) recently concluded that “agent-based modeling is most appropriate for domains characterized by a high degree of localization and distribution and dominated by discrete decision”. These characteristics make ADS different from traditional distributed simulation paradigms from the following perspectives.

- *Agents are not equivalent to jobs.* Traditionally, distributed systems research has considered the cause of load imbalances to be an inherently user-oriented problem (Kafeel and Ahmad, 1998). The entire cluster of machines is often considered to be a common computational resource available to and shared among a large number of end users each with their own and often competing objectives (Hwang and Xu, 1998). Load balancing techniques employed here often start, pause, or move user-submitted jobs to ensure

the system is fully but not overly utilized. In ADS, it is the agents who share and consume system resources, but the data dependencies among them are often exponentially higher than the jobs of distributed systems. The opportunities for executing independent calculations in parallel are often limited to ensure the validity of the system, and long chains of dependent calculations have to be frequently dealt with. Multiagent distribution techniques must be able to remap entire portions of the simulation environment to physical hardware without disrupting this tenuous ordering of execution flow.

- *Agents are often embedded in an environment.* Traditionally the actions of agents are executed within some environment to allow for the observation of their outcomes. The individual agents themselves become tied to the environment that surrounds them including other nearby agents. This environment can either be spatial in nature or logic relations between agents. In either case, for efficiency reasons any ADS should aim to store agents and their associated environmental data on the same machine. Such a policy would encapsulate a majority of an agent's computation within one machine rather than across several.
- *MASs utilize communication extensively.* Unlike the traditional distributed computing model where each machine carries out its workload independently until the simulation terminates, at which point results from each are aggregated to form a final result, the distributed ADS requires intermediate communication between machines. In addition to messages sent between agents, machines may also communicate among each other about a shared data source. The cost of communication should play a crucial role in determining the load of an individual machine, factoring in message length and network latency as well as the frequency of the number of messages sent/received.
- *Workloads are heterogeneous and potentially volatile.* The amount of work carried out by intelligent agents during each interval of a simulation is by no means homogeneous or stable. An individual agent's workload is a function of the number of messages it has to process and/or send, the degree of change in its environment, and the complexity of its decision making. A load balancing policy should be able to rapidly and accurately adapt to fluctuations in both the workloads of individual machines and the entire system.
- *MASs rely heavily on shared data.* Achieving a complete decomposition of the environment agents share is impossible for many problem domains. Rather, a small portion of shared data must be maintained by each machine and synchronized at the end of each interval of the simulation. A proper load balancing policy should manage this overhead by avoiding scheduling data synchronization during peak workload times or on machines that are already or close to reaching an overloaded state.

This chapter discusses ADS from the perspective of the above new challenges. We first introduce different architectures for designing large-scale distributed ADS systems. This is followed by a general description of ADS techniques with respect to four aspects: language, environment, service, and application. We then review five well-known ADS platforms from the point of view of these four aspects: Ascape, Mason, NetLogo, Repast, and Swarm. Finally, we give an extensive review of two other ADS platforms: Brahms and CASESim. Brahms is the only platform that is both a simulation language and an agent-oriented language, and CASESim creates a distributed spatial environment that is suitable to agent-based social systems.

10.2

Architectural Style for ADS

With large-scale software systems, their architectural styles are thought to be key to creating successful large complex systems (Perry and Wolf, 1992). Originally introduced by Shaw and Garlan (1996), architectural styles are a way of abstractly representing the components, interactions, and constraints of the system. Many higher-level architectural styles such as client-server, pipes and filters, blackboard, and layers have already been adopted and implemented in many existing large-scale software systems with much success. These architectures allow for much more robust systems and improve the overall design process by allowing the reuse of previous patterns and code segments (Buschman et al., 1996).

Existing large-scale ADSs have used a variety of different styles, each with its benefits and drawbacks. Two of the most commonly used styles are the layered approach (Khosla et al., 2005) and the centralized synchronous approach (Davidsson et al., 2005).

The layered approach as utilized in MASs split the system into layers of functionality. Thus, the system might be composed of an object layer, a distributed processing layer, an agent layer, an optimization layer, and a problem-solving layer each building on the structures and functionality of the lower layers. While this style is effective when dealing with isolated problem-solving agent systems, its structure makes it difficult to represent aspects crucial for ADS. In the case of the agent and the environment, neither one is based on the functionality of the other, but both are mutually important and exclusive groupings of functionality in a social simulation. The layer approach has no way of effectively and easily modeling the relationship between these two. Also, since each layer is completely dependent on the functionality of the previous layer, this style lacks the modularity of functionality that is desired within a large-scale ADS.

Another commonly used approach, the centralized synchronous system, consists of a set of components that are synchronously controlled by an overarching hierarchical component. Each lower-level component relies on commands sent from the central component before it is allowed to act. This architectural style is useful when there is a need for exact control over the timing of actions and the use of resources. This centralized control also allows for better encapsulation of the lower process-

es, which makes switching them out relatively easy. While both of these features are beneficial in ADS, the centralized approach suffers from an inability to scale effectively. Since all the computation and communication must go through a single node, this creates a bottleneck effect, which in the end kills the effectiveness of the overall distributed network. Thus, for an agent-directed simulation, where scalability is essential, the centralized style becomes an inappropriate approach to the software.

Since the traditional approaches have been shown to be ineffective for systems designed for ADS, there have recently emerged nontraditional styles, specifically the service-oriented architecture (SOA) approach. The SOA style is built on the principle of modularity, separating functionality into encapsulated modules that can operate independently (Bell, 2008). By encapsulating the functionality like this it allows for the ability to switch out individual modules depending on the needs of the user. This architectural style is often used to make business and web applications because of its interoperability and flexibility as far as adding and removing services (Erl, 2005). With its increasing use in multiagent simulation, the SOA model has numerous benefits for a multiagent simulation system. (1) SOA stresses the loose coupling of its encapsulated services; in other words the system is designed so that each service has as few dependencies on other services as possible. This loose coupling of services allows for individual services to be distributed to different machines without worrying about difficulties with interdependencies (Erl, 2005; Bell, 2008). This is crucial in ADS because not only does it cater to the need for ADS to be highly distributed, but it also allows the end user to be able to control the system from any other computer, which is highly desirable for non-computer-science researchers who will most likely be the end users for the system. (2) SOA also allows for optimization based on service importance. This allows for certain services to be weighted more highly and receive more computational time than others. This way the main system can be given priority during the simulation, while the visualization can be given priority while the results are being viewed and interpreted. The SOA style gives the flexibility as well as the robustness that is needed with a highly distributed ADS system.

10.3

Agent-Directed Simulation – An Overview

ADS presents a framework where agents are used to generate model behavior in simulation as well as that provides simulation services such as visualization and data analysis (Yilmaz, 2006). Agent-oriented languages (AOIs) is about programming languages in which to define agents and their cognitive abilities such as perception, beliefs, goals, and intentions (Shoham, 1997). Cognitive modeling (CM) has been inspired by the advances in cognitive neuroscience and is a way of specifying how the brain itself is organized in a way that enables individual processing modules to produce cognition (Buchler et al., 2008). Figure 10.1 presents the three types and

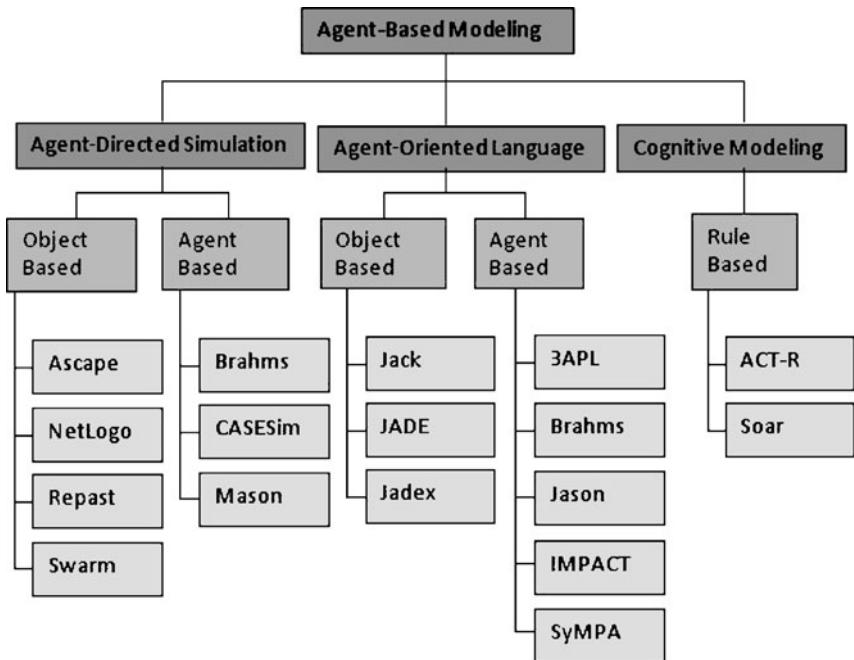


Fig. 10.1 Types of agent-based modeling.

some representative platforms for each type.⁶ Readers should be aware that the selection of a platform in Figure 10.1 is only a matter of taste and reflects our own viewpoint; we may ignore many innovative platforms that could be good choices for some projects.

10.3.1 Language

An ADS language should include the core capabilities of agents. Traditionally the design of agents has centered on the common abstract notion of an agent execution cycle. This structure serves as a high-level map for the internal components of any agent-based system. This relates not only the data structures that comprise an agent's knowledge about the environment but the algorithms that act on and control that flows between these structures. In a vast majority of cases agent architectures differ only by the data structures and algorithms they choose to utilize. Figure 10.2 illustrates this cycle graphically, with details about each of the five major steps listed directly below.

6) Brahms appears in two places in Figure 10.1 because it is the only platform that belongs to both ADS and AOL.



Fig. 10.2 Traditional agent execution cycle.

- *Observation*. This step collects information on current environmental conditions and maps those conditions to precepts. It is important to note that this step is absolutely domain dependent and limited in its scope by its implementation. For example, if this model were to be implemented within some sort of robotic system that utilizes a video camera for input, then the agent's observation step would be limited in the amount and types of information it could take in as sensory input.
- *Updating Knowledge Base (KB)*. An agent's KB will be updated under three cases: (1) when the agent observes the environment, it will update the KB by the new percepts; (2) when the agent performs an action, it will update the KB by the effects of the action; and (3) when the agent receives a communication message, it will update the KB by that message. For all cases, the function update must check the entire KB for inconsistencies.
- *Decision*. Here agents make two separate decisions: (1) what action to perform and (2) what message to communicate and to whom.
- *Communication*. In general agents working within a multiagent environment cannot force other agents to perform a specific action or directly alter their KB. However, they can exert influence over other agents through communicative actions. Multiagent researchers have built upon John Searle's Speech Act theory (Searle, 1969) to develop a number of formal languages and ontology, such as FIPA-ACL (Foundation for Intelligent Physical Agents – Agent Communication Language) (FIPA, 2008) and KQML (Knowledge Query and Manipulation Language) (Finin et al., 1997) so agents can understand one another.
- *Action*. The functional nature of an agent's action step is rather intuitive and simple; its purpose is to ensure a successful, coherent, and fault-proof execution of the optimal action that was recommended by the agent's decision making mechanism. No real further explanation of the act is necessary as this function is highly dependent on the implementation.

10.3.2 Environment

ADS uses agent technology to develop simulation techniques. As Davidsson et al. (2005) point out, this makes the modeling approach of ADS unique and it should be captured by the following aspects.

Simulated Entities The simulated entities are the foundation components of the studied system. Three different categories of entities are identified. (1) *Agents* include software agents, robot agents, or human agents. (2) *Environment objects* include any natural objects in the environment. (3) *Organizations* include any group format of agents.

Agent Types According to Russell and Novig (2002), there are four basic agent types. (1) *Simple reflex agents* select actions on the basis of the current percept, ignoring the rest of the percept history. (2) *Model-based agents* have their mental states such that they can keep track of the part of the world they can't see now. (3) *Goal-based agents* not only know the current state of the environment but also goal information that describes situations that are desirable. (4) *Utility-based agents* have a utility function that can help them compare different world states and generate high-quality behavior.

Communication Agents interact with each other in MASs. The interaction can be classified as direct interaction (i. e., by communication) and indirect interaction (i. e., by observation). Therefore interagent communication is important for agent interactions.

Simulated Environment One important characteristic of agents that distinguishes them from jobs in distributed systems is that agents are dependent on the local environment near them. This environment can be either a spatial space or some relative locations between agents. Thus separating agents and their local environment would increase the network communication load as the agent acts on its environment, which is now on a separate machine. This schema also does not take into account the abundance of intraserver communication between agents. The network lag caused by this communication often slows the system down more than the computational load. Therefore, to model the environment, a well-balanced distributed ADS needs to take the dependence of agents on their environment into account.

Mobility Mobility means the ability of an agent to migrate between machines in a network of machines. Defining this ability needs to consider two factors. (1) Agent-environment dependency. Since agents are dependent on the local environment, we need to avoid this dependency issue of when the agents are removed from the local data they need. (2) Intraserver communication. When agents freely migrate between machines, intraserver communication will increase in order to

have the communication routed through another machine and the extra computation to determine the machine to send the message.

Adaptivity Adaptivity means that agents can learn from their experience and adapt to new environments. Therefore, adaptivity is an important feature that can lead to agents' autonomous behavior.

10.3.3

Service

As scientists from various domains increasingly resort to ADS for a more thorough understanding of real-world phenomena, the need for simulation services that facilitate the rapid development of MASs is growing.

Visualization An ADS platform should provide a means of visualizing the simulated scenario. This includes a variety of textual and graphical (2-D or 3-D) browsers that allow the modeler to detect trends and relationships in the simulation scenario. Based on Mostafa and Bahgat (2005), the task of agent visualization poses major challenges that are not typically faced in traditional visualization techniques. The most notable of these challenges is the fact that an agent's state is continuously changing, and thus the data to be visualized are dynamic rather than static. At the same time, the responsiveness of the system is a key requirement, so those changes have to be visually reflected within an acceptable time limit. The volume of data to be visualized poses another challenge; a large number of agents is typically involved in a simulation, and each agent has a set of attributes that the modeler may like to observe. Moreover, the nature of simulation data is usually not restricted to variable-value pairs with which traditional visualization techniques are most effective. Rather, the simulation produces data concerning actions, events, and communication between agents in addition to traditional variable-value pairs describing agents' states.

Data Analysis Data analysis is the process of looking at and summarizing data with the intent of extracting useful information and developing conclusions. Data analysis methods vary for different kinds of data. For example, in the social sciences, statistical analysis usually is used for numerical data, while qualitative data analysis is used for nonnumerical data such as words, photographs, observations, and so on. Providing powerful tools for data analysis, this capability needs to be built in and easy to use because it is needed by even the most novice modelers.

10.3.4

Application

The ADS technologies are broadly applied to two agent levels: the organization, interaction level (e.g., communication, negotiation, coordination, collaboration) and the individual agent level (e.g., deliberation, computational autonomy). Applica-

tions are found in various areas such as biology (Casal et al., 2005), business (Chen et al., 2008), commerce (Serzysko et al., 2007), economics (Deguchi et al., 2003), engineering (Jennings, 2000), environmental sciences (Bousquet and Page, 2004), individual (Bonabeau, 2002), group (Goldstone et al., 2006), and organizational behavior (Hoggendoorn, 2007), management (Domingos and Richardson, 2001), simulation gaming/training (Acquisti and Sierhuis, 2002), and social systems (Sal-lach and Macal, 2001).

10.4

A Survey of Five ADS Platforms

We review five of the seven well-known ADS platforms in Figure 10.1: Ascape, NetLogo, Repast, Swarm, and Mason. Each of them is reviewed from four perspectives: language, environment, service, and application. The other two platforms, Brahms and CASESim, are introduced in Sections 10.5 and 10.6, respectively. We give extensive overviews of these two platforms because Brahms is the only platform that is both a simulation language and an AOL and CASESim creates a distributed spatial environment, which is typically suitable for agent-based social systems.

10.4.1

Ascape

The name Ascape (<http://ascape.sourceforge.net>) comes from the combination of the terms “agent” and “scape”, which are the two fundamental concepts upon which Ascape was built. The term “scape” is derived from Sugarscape (Epstein and Axtell, 1996), in which its definition was an area where agents lived. In Ascape, a scape is generalized to include any collection of agents. Agents can be a member of more than one scape, and every Ascape model has a root scape to which all agents belong.

Scapes are essentially collections of agents with very rich functionality. They know not only what they contain, but also how to iterate across what they contain, details of the topology of what they contain, appropriate ways to interpret what they contain, and so on. More importantly, scapes are themselves first-class agents. Anywhere that a model can use an agent, it can use a scape. Thus, scapes can always be composed of other scapes. This allows for a very natural method for model composition. It also ensures that any behavior that occurs across an Ascape model occurs in a well-defined way.

In Ascape, models are always composed of hierarchies of scapes and agents (Figure 10.3). For example, the model of a demographic prisoner’s dilemma has a lattice across which players move, play, and reproduce, and a vector that contains the population of players. A “hosting” mechanism allows players to place themselves upon the lattice. The players and lattice belong to a root scape.

To create a basic model (Ascape includes preformed models that nonprogrammers can use), a user needs to define the model class, which can extend a ba-

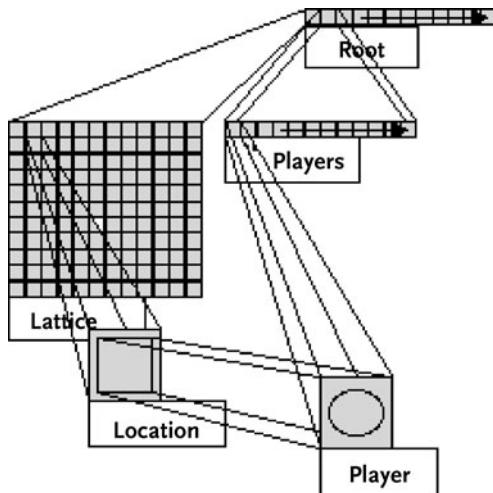


Fig. 10.3 An example hierarchical model in Ascape.

sic scape geometry, several of which are included with Ascape (e.g., ScapeArray2DVonNeumann, ScapeArray2DSmallWorld, ScapeArray2DMoore). Next, the model can use instance variables to track data like wealth or age. Then, the `createScape()` method needs to be called. This method populates the root scape with agents, which might be scapes themselves. These agents are classes themselves, and they can extend different agent classes, depending on what the user wants.

Rules determine how agents across a scape behave. Rules have the option of being executed across all agents (RULE_ORDER execution) or across each agent in turn (AGENT_ORDER) and are derived from the scape to which the agents belong (e.g., an Inmates scape that contains the population of Prisoner agents). There exist predefined rules for use, and the option of user-created rules is also available.

10.4.1.1 Language

Observation. The agent uses the methods of the cell (of the grid/lattice) to determine its surroundings. Such methods include:

- `getHostCell()`: To determine what cell the agent is in.
- `countNeighbors()`: To count the number of agents adjacent to the agent.
- `isNeighborAlive()`: To determine whether the agents adjacent to the agent are alive.

This abstraction by the Ascape framework allows the agent's code to be independent of the geometry of the scape (where the cell is). This allows for the user to change aspects of the scape while still retaining the functionality of the agent's code.

Updating KB. Scapes are initialized with certain rules to run the simulation by (or apply to the agents). These include such rules as RANDOM_WALK_RULE, DEATH_RULE, and UPDATE_RULE. The update rule calls the Update method.

What `Update()` does is save and update the instance variables of the agent during the last period. If the user is using a view to collect and analyze statistics, each view is notified of the updates to each scape. The view will then respond to the changes, as dictated by the user, at which point the model is free to iterate again.

Decision. The decisions an agent makes are based on a strategy applied to the agent. This strategy is an attribute of a specific agent, initialized when the agent is created. Any number of conditions, random or not, can be used as well to determine an agent's strategy. Users can select from the many predefined strategies, and of course they are free to create new ones for their specific purposes. Such strategies are COOPERATE or DEFECT for a simulation of the Prisoner's Dilemma.

Communication. Ascape does not provide support for agents to communicate using a particular protocol. However, a basic communication is available by use of an agent's methods. For example, if one agent knows where a lot of sugar is and finds another agent, it can call a method that gives the location of the sugar to the other agent. The user must create these methods of communication.

Action. Agents take action based on rules and strategies. Each scape can have a different method of activation for these rules. Basically, the rule is instantiated, conditions are tested, and the agent is told to take an action or not.

10.4.1.2 Environment

Simulated Entities. Ascape is used to simulate agents and environments, as well as the countless physical objects that could be present in those environments (e.g., water sources, sand dunes, farms). There are definitely rules that exist for agents to organize into groups or enterprises, and much research has been done in these areas.

Agent Type. The agent types available for use or creation in Ascape range from simple reflex agents up to utility-based agents; this is dependent on the complexity of rules and strategies the user has defined for the agents to follow. In order to create a utility-maximizing agent, the user would need to implement the appropriate algorithm for the simulation. User tools are available (from other companies) to aid in this task, but are not directly part of the Ascape package.

Mobility and Communication. Ascape currently only supports parallel execution on a single machine. This is done by simply serializing some subset of agents and squirting them across some grid architecture such as JGrid. The developers of Ascape have also designed a number of models that run in a distributed setting including some generalized view classes to support use of a proxied agent design, but these models have not been publicly released.

Adaptivity. Ascape does not support a particular learning methodology. The user, however, is able to customize an agent's behavior very specifically. If the user wanted to create a type of learning or adapting agent, he could, for example, give an agent a memory array that records its interactions with other agents and affects its future actions.

10.4.1.3 Service

Visualization. The Model-View-Controller visualizes the visual simulation. Once the user creates a root scape, he/she can call the `createViews()` method to determine the view to be displayed. Statistical collection (such as “agent is alive”) can also be added and Ascape automatically displays this to the GUI view. Statistics collectors can be added to any scape, and if one is added to the root scape, data are collected over all the elements of that scape.

Also automatically added to the view are any instance variables that have appropriate get and set methods. This aspect of Ascape allows the user to concentrate on implementing agent-based models while letting Ascape take care of the actual simulation and display of the models. The Model-View-Controller handles this separation of code, collecting data from the model (for display or other output) after each iteration.

The GUI tools available for use are very user-friendly, with simple panels, textboxes, and drop-down menus. There are a myriad of options regarding every feature of Ascape, including graphs and agent behavior rules, as well as the ability to record a video of your simulation. Variations on models can be investigated intuitively and interactively by turning rules on and off through the Model Settings window.

Data Analysis. Ascape provides some basic statistical tools for data analysis, such as count, sum, average, minimum, maximum, variance, and standard deviation. The user can create others.

10.4.1.4 Application

The main areas of study with Ascape are social and economic simulations. Recent work has focused on developing specific models in Ascape of interest to government and corporate clients. These models include growing artificial societies, modeling social networks, the emergence of economic classes in society, cooperation among agents, and the effect of topography.

10.4.2

NetLogo

NetLogo was first authored by Uri Wilensky in 1999 and has since been under continuous development by the Center for Connected Learning at Northwestern University. NetLogo was designed to simulate natural and social phenomena over time using a group of autonomous agents. In comparison with cutting-edge simulation systems, NetLogo was designed to simulate relatively simple phenomena with agents containing relatively simple decision making processes.

10.4.2.1 Language

The founding idea behind NetLogo is to provide a simple and relatively adjustable environment in which to run custom or provided simulations. NetLogo uses a unique set of commands for the user to input the domain-specific simulation items such as agent and environmental behavior. While this unique set of com-

mands makes creating simple situations considerably easier for the beginning programmer, it also greatly limits the systems possibilities in the mind of a more advanced programmer. Through a unique set of NetLogo-specific commands, one is able to create sliders and switches to adjust parameters set forth within the simulation by the programmer. While not providing “domain-specific” tools per se, NetLogo does provide built-in support for items such as bounding area, X-coordinate, and Y-coordinate.

Observation. NetLogo’s most basic information storage unit is a “variable”. The best analogy to this in Java is a string. A NetLogo “variable” can store all kinds of information and, depending on what context it is used in, the information is parsed into the correct form. These “variables” can be united in tables; however, multidimensionality is not supported. Because NetLogo only allows one class in a program, to observe something one just accesses the proper “table” or “variable”.

Update KB. As in observation, updating of a KB is done purely by assigning new values to “variables” or “tables”. This again can be done because NetLogo requires all code to be within one class and therefore all variables are accessible.

Decision. NetLogo offers no functions to assist with decision making. Decision making processes are coded traditionally, using comparison operators and various other basic programming utilities.

Communication. In NetLogo, agents that move around a board are called Turtles. Turtles do not have direct methods of communication; however, to create and run a set of Turtles you must have the three other types of agents: patches, links, and observers. Links and observers are agents that you have one of a piece within a basic simulation and the number of patches varies. Most simply: an observer observes the simulation and outputs the data to the interface, a link connects the data set to the observer, and patches instantiate and run a set of Turtles. Turtles can communicate via a patch, but it gets a bit messy.

Action. Action functions are very well supported within NetLogo. For Turtles, there are four basic movement commands: Left (which turns the agent x degrees to the left), Right (which turns the agent x degrees to the right), Forward (which moves the agent x units forward), and Backward (which moves the agent x units back). Patches have a variety of different commands depending on the type and purpose of the patch. These commands range from changing a Turtle’s display color to running “action” for a whole set of agents.

10.4.2.2 Environment

NetLogo is a cross-platform modeling environment, claiming to run efficiently in Linux, Windows, and Mac OS X. Because of the simplistic nature of NetLogo, it has lent itself thus far to simulating well-known natural phenomena as well as reaction- and model-based social simulations. Some forays are beginning to be made into implementing NetLogo as a more intelligent MAS.

Simulated Entities. As stated above, NetLogo’s simplistic nature has lent itself thus far to natural phenomena (such as chemical reactions and photosynthesis simulations) as well as some reaction- and model-based systems (such as foraging

ants and the beginning stages of a simple Sugarscape (Epstein and Axtell, 1996)). The horizon for expansion into more complicated simulated entities does exist, but unfortunately, due to the lack of a formal programming language, the more complex system one develops, the more inadequacies one finds in the provided programming commands.

Agent Type. NetLogo has been shown to work with simple reflex and model-based agents. The functionality does exist for goal-based and utility-based agents; however, these higher-level agents become increasingly hard to program in the “lacking” provided interface language.

Communication. The method of communication, if any, between agents is chosen and implemented completely by the simulation user. No tools for communication are provided in NetLogo, and generally the level of simulation being run in NetLogo does not require such a feature.

Mobility. NetLogo is designed to be a single-machine modeling environment for MASs of a couple thousand agents or fewer. As such, NetLogo need not concern itself with cross-machine communication or movement.

Adaptivity. There are no agent learning tools incorporated within NetLogo; however, if one went to lengthy ends, general adaptivity could be implemented.

10.4.2.3 Service

Inherent Programming Language. Instead of having simulations written in a common programming language such as Java or C++, NetLogo chooses to provide a unique language in which to construct simulations. This has both advantages and disadvantages.

Advantages:

- Small learning curve for the beginning programmer;
- Easy access to basic visualization;
- Easy debugging.

Disadvantages:

- Lack of complex functions, severely limiting the advanced programmer;
- Increased difficulty in constructing model environment extensions.

Visualization and Data Analysis. Possibly NetLogo’s strongest characteristic, a variety of easy-to-use visualization and data analysis tools are readily available. This is made possible by a library of basic visualization commands such as plot (blank) on X and (blank) on Y. Unfortunately, only two-dimensional modeling is currently available and there is no predicted expansion into three-dimensional modeling.

10.4.2.4 Application

NetLogo was designed with the inexperienced programmer in mind and, as such, has attracted mostly simplistic naturally occurring phenomena as simulated entities. More recently, however, some people have begun extending NetLogo to more complicated simulations.

Reflex and Model-Based Applications:

- LeafMacro – Model of a plant growing and responding to surrounding conditions.
- LeafMicro – Model of photosynthesis within a leaf.
- WaterCycle – Model of a water cycle in a closed environment, responding to a few stimuli such as temperature, dew point, and so on.
- Sugarscape – Only in single-commodity phase with no reproduction included.

Goal- and Utility-Based Applications (currently in development):

- Extension of Sugarscape – multicommodity, mating, and loaning;
- Extension of Ants Foraging – communication, intelligent trade, and group dynamics.

NetLogo is a great simulation environment for the beginning programmer looking to create a simplistic simulation using reflex or model-based agents. Visualization and data analysis for such simulations is also easily programmable and well documented. NetLogo is not good for experienced programmers looking for higher-level agents or simulations. It also is not good for any simulation requiring advanced data analysis or three-dimensional modeling.

10.4.3

Repast

RePast (repast.sourceforge.net) groups agents in a series of “contexts” that defines an abstract population and interactions of its members without actually providing implementations of them (Figure 10.4) (Repast, 2008). These contexts can provide any number of different pieces of information as well as available behaviors for agents in it. Some of this information may be, but is not required to be, related to spatial organization. Agents can freely use whatever information is available within a given context, and their behaviors are restricted to what the context provides. Contexts also can have multiple subcontexts, and agents can freely move across contexts as the local circumstances change. In addition to contexts, there is also the concept of “projections” upon concepts, each of which facilitates a particular kind of relationship agents can make, and can define their interactions among each other. Many projections can be applied to each context, allowing for an arbitrary set of possible relationships. Below is a visual example of a context containing agents in the upper left corner, three different kinds of projections on the right side (network, map, and grid projections), and two different subcontexts at the bottom.

10.4.3.1 **Language**

Observation. Repast allows for agents’ observation of their environment through the contexts they are registered in. The contexts contain their internal states through a series of variables, such as time, and data fields, which are n -dimensional fields

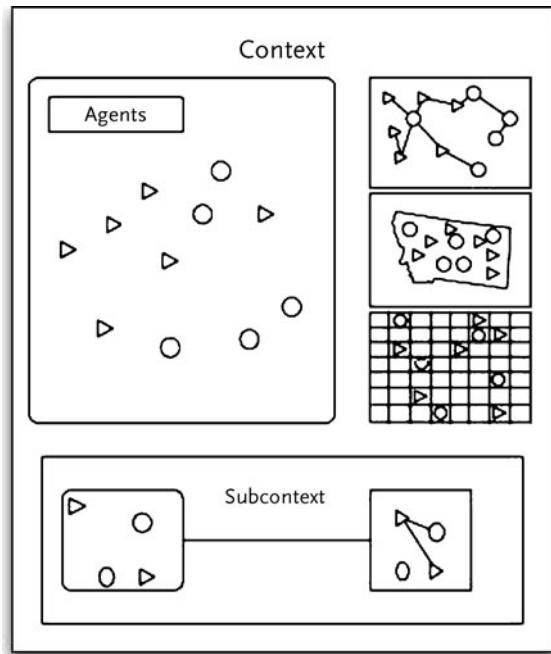


Fig. 10.4 A simple context in repast.

of values with which agents can interact, designed to be highly generic in that each value is derived from a set of coordinates.

Updating KB. This information can be mapped onto matrices, transformed, or applied to functions to produce meaningful results, depending upon the demands of any individual implementation. As conditions change, agents take actions, and the environment gets updated, and this information can change, providing a function similar to that of individual KBs.

Decision. The decision making process is up to the end user, as Repast does not provide any built-in decision capabilities.

Communication. Communication is done through projections, which, based upon a given relationship, will match the desired agents to be communicated with and send references to the calling agent, which then makes direct calls to the methods of those agents.

Action. Actions are also up to the agent developers, but the actions any agent may take at a particular time may be restricted by the set of appropriate actions the developer specified in the contexts. This allows for context-sensitive actions, where the same circumstances may merit different actions depending upon the situation.

10.4.3.2 Environment

Simulated Entities. The entities that Repast works with are referred to as proto-agents, which are models that maintain a set of properties and behaviors but may

not have the ability to learn. This is a fancy term to describe the same thing done in CASE, so that objects such as sugar do not get handled independently of the more intelligent agents in the simulation.

Agent Type. Repast also does not provide any particular type of agent design other than reflex agents, for which context information can be obtained rather easily, and agents can be defined to take specific actions based upon the information they receive. If the agent developers wish to have a more complicated agent type, such as model-based, goal-based, or utility-based, then it is their own responsibility to produce the code for it.

Communication and Mobility. Repast provides context-oriented communication between agents and handles the distribution of agents across servers invisibly through the use of a program called Terracotta. Terracotta works outside of the Repast code, monitoring the Java Virtual Machines (JVMs) of multiple machines to create a memory map of the combined heap spaces. It then works to essentially combine the multiple JVMs together to act as if they were one. Then the program is able to access every object as though it were local, and issues such as locking and remote calls are made irrelevant.

Simulated Environment. As strictly defined by its structure of contexts and projections, agents are dependent on their local and, to a slightly lesser degree, parent environments. However, this by no means restricts them and their possible actions, as they can choose to move freely in and out of contexts, which may or may not have spatial definitions.

Adaptivity. Unfortunately, since Repast was designed to be so general, and therefore does not define any kinds of learning behaviors for agents, adaptation is not an integral part of it, and although it provides a general interface for neural nets, the implementation must also be developed on the client side.

10.4.3.3 Service

Visualization. Repast provides a small variety of different visualization tools, including the ability to make various charts that can compare the information of agents within contexts. There is also a two-dimensional graphing tool that can plot agents and their locations and values, if they have any. Lastly, it has a basic three-dimensional graphing tool that can show agents and their environments in a more interactive and visual way by representing them as models on a field. In addition to built-in tools for visualization, Repast was designed to be highly extensible, and is compatible with many different external visual and data analysis tools. Examples of such visual tools include VisAD, for scientific visualization, and the iReport visual report designer.

Data Analysis. Tools for data processing and analysis include the R statistics environment for statistical computing, the Weka data mining platform, and the MATLAB computational mathematics environment.

10.4.3.4 Application

Repast was mainly designed for use in the social sciences, in order to simulate various social phenomena. However, Repast has also been used in some biology studies to show things such as evolutionary models and various applications of genetics. Repast also has a weak link to neuropsychology, but only when linked to social applications.

10.4.4

Swarm

Swarm (<http://www.swarm.org>) was developed at the Santa Fe Institute starting in 1994 and is still being developed today. It was designed to give scientists useful, abstract tools to be able to do research with instead of having to write research-specific programs that would never be able to be reused. This is a common problem within nontechnological research; many of the scientists have to write their own programs to get the simulations they want, and they never know if programs like that have already been written so it is a huge duplication of effort. Swarm was written to target highly complex, adaptable systems. It is “a system of independent agents that interact via discrete events” (Minar, 2008).

10.4.4.1 Language

The original system was written in Objective C, but there has recently been another version released that was written in Java, so it is totally object-oriented and compatible with Java, C, and Scheme. Swarm has its own set of libraries that are essential for the user to be able to construct a simulation. These libraries can be classified in three categories:

- Generic libraries. The *defobj* library is used to define the infrastructure of the model; it provides a usable interface to be able to work with the complicated processes that run underneath it. The *collections* library contains the classes that are used to track the agents in the system such as maps, lists, sets, and so on. The *random* library contains a number of random number generators that have all been published and can be used for trials that need to be randomized or run numerous times with completely unique sets of data. This can sometimes be crucial to experiments because if the generator shows a pattern, it could throw off the results enough to invalidate them. The last essential library is *tkobjc*, which provides the tools to make a graphical user interface. It has the classes for much of what would be needed for a GUI such as buttons, graphs, and data printouts.
- System-specific libraries. The *swarmobject* library gives the agents their behavior and implements system distribution management techniques and the Swarm class, which gives code to the model and observer agents. The *activity* library contains all the data structures for scheduling and for execution support. This class has the methods to be able to keep track of time during a simulation by interpreting the timestamp that is given to each event

as it occurs within the simulation. The *simtools* library is used mainly for visualization and will be discussed later.

- Domain-specific libraries. The *space* library is based on a two-dimensional array used to house the locations of objects within the model and to build on them. The *ga* library is a set of genetic algorithms that was user-contributed; this could contribute to the evolution of the agents if they were programmed to reproduce. The *neuro* library can be used to implement a variety of neural networks, which is a common learning tool used for agents. Both of the last ones are still under development since they are a fairly new addition to Swarm.

These libraries are used to help facilitate the execution of the agents' cycle through the simulation.

Observation. The *swarmobject* library gives agents the ability to observe their surroundings, and the *activity* library tells the agents that observation is their initial behavior.

Updating KB. The agents' KB would be updated by the new perception from observation. This gives the agents the ability to learn or evolve as the simulation runs.

Decision and Action. The agents' execution is based on a schedule that is made up for them at the beginning of the simulation. The decision making capability of the agents exists because the agents can choose to execute the actions when they are supposed to or they can choose to execute them before or after the scheduled time. As with an animal example, they would have to eat food; their next step would be to hide from predators, and the last step would be for the predators to go looking for weaker animals and eat them if they found any. This schedule would be executed over and over until the end of the simulation.

Communication. Swarm provides interagent communication, but the communication protocol is user defined because Swarm does not come with any standard protocols like KQML or FIPA_ACL.

10.4.4.2 Environment

Simulated Entities. Swarm is predominantly used to simulate potential situations in real life. The agent is the basic unit that makes up every model; most of the time the models represent living things or physical entities such as animals or people in some environment. These agents are then used to make up groups called swarms, which consist of all the agents in the model.

Agent Type. The system can have a plethora of different agent types depending on the environment, so they can range from simple agents to complex ones. But in general, the agents tend to be simple because they are given a predefined schedule and execute them without having much rational thought. They also include the options for probe agents, which will be discussed in the services part of the overview.

Simulation Environment. Since the structure of the system is hierarchical, swarms can be built on top of each other. For example, a collection of cells exhibits this feature because there is the initial swarm of cells, but each cell has another swarm

of organs within itself and those organs have collections of their own components. This tree can go indefinitely until the most basic elements of the model are defined. The spatial explicitness of the system is clearly defined in the space library with the definition of multidimensional arrays that can be modified depending on the wishes of the researcher.

Communication. Agents can communicate with each other, but they do not have any preprogrammed methods within Swarm for communication. These methods must be written by the user if they wish the agents to communicate during the simulation.

Mobility. Swarm does not attempt to load balance, but it has several features to facilitate a distributed single simulation. One feature is that it can read and write objects/agents to and from HDF5 (HDF5 is a data model that can represent complex data objects and a wide variety of metadata) in a parallel, random-access fashion. So, if one wanted a shared global space for agents, one way to implement it would be to use a multiprocess Swarm simulation coordinated via a shared blackboard (HDF5). The other feature is the XML-RPC distributed objects module of Swarm. Here there are proxy objects that talk to other objects in other processes (usually on other machines). Roughly, HDF5 would make more sense on a dedicated cluster and XML-RPC on a loose collection of different machines (say over the Internet). The former could support many transactions in flight at once and the latter relatively few.

Adaptivity. The agents can learn from experience by various learning algorithms provided by Swarm such as neural networks and genetic algorithms. But the modeler needs to define this learning action in the agents' schedule before the simulation starts to run. Basically, the agents can only do what is in their schedule to be executed; they cannot veer off course and try to do something else.

10.4.4.3 Service

Visualization. One type of agent that is unique to Swarm is the probe agent, which can be implemented in the system to gather data and not make the user interact with the program at all. These agents can be programmed separately from the model to give the impression that they are scientists behind glass observing the simulation as it runs and gathering the desired data. The *simtools* library is a collection of classes that is needed for the execution of the simulation; it helps with visualization and data collection during a run. This class has two modes of operation: a fully graphical one to enable the user to interact with it as the simulation runs and an offline version to enable data collection without direct interaction with the system; this second version allows the probes to be able to gather the data and store these data for later consideration.

Data Analysis. Swarm provides simple data analysis such as averaging and can also calculate probabilities within a simulation. That is the extent of the data analyzing tools, but the Swarm library seems to be more focused on visualizing the data than analyzing it.

10.4.4.4 Application

Swarm is mostly used in biological applications because it is better suited to research the workings of groups of animals and how they go about their natural processes. Other application fields include chemistry, economics, physics, anthropology, ecology, and political science. In economics it has been applied to study the choice of driving polluting cars versus buying new ones. In ecology, predator and prey populations have been studied and simulated. In theoretical biology, it has been used to make headway in discovering the origin of life in algorithmic chemistry. In biology, it has been used to study the dynamics of insect colonies. Swarm has also been used to create the classic MAS program of Sugarscape (Epstein and Axtell, 1996).

10.4.5

Mason

Mason (Mason, 2008) was developed at George Mason University. It is a fast, easily extendable discrete multiagent simulation toolkit written in Java. Mason was designed to fit applications from swarm robotics to social complexity environments. Modeling and visualization are carefully separated in Mason, allowing for easy cross-platform running and even easier dynamic presentation of simulation data.

10.4.5.1 Language

The founding idea behind Mason is to design a fast, efficient simulation toolkit that will run a variety of different simulations. Mason is written in Java and is built from the bottom up. It does not have a built-in library of domain-specific tools or functions, because the developers believe their time is better spent designing a universal modeling and visualization system and leaving the domain-specific simulation tools to be developed by the users.

As of now, Mason does not have integrated support for the basics of multiagent simulations such as Observation, Updating KB, Decision Making, Communication, or Action. The developers considered these tools to be “domain specific” and expect the programmer writing the simulation to include them.

10.4.5.2 Environment

Simulated Entities. Mason is a cross-platform toolkit, being able to run on Linux, Windows, and Mac OS X. Since Mason mainly emphasizes modeling specificity, it is able to play host to a very wide variety of simulations on everything from living organisms (ant foraging model) to particle physics (atom collision model). Because the simulation side of the program is written entirely by the user, the options available in terms of agent and communication types are fairly unlimited. The base structure of the program is a single-process simulation, with computing being delineated in a client/server fashion.

Agent Type. Mason has been shown to work with simple reflex, model-based, goal-based, and utility-based agents; however, there is no built-in support for any of these agent types. All agents are to be designed by the toolkit user.

Simulated Environment and Mobility. Mason is designed to be an efficient single-machine simulation toolkit for MASs. As such, Mason does not concern itself with agent movement because it is all in one process on one machine.

Communication. The method of communication, if any, between agents is chosen and implemented completely by the toolkit user. No tools for communication are provided in Mason.

Adaptivity. There are no agent learning tools incorporated within Mason; however, there have been simulations in which users of the toolkit have had their agents adapt to the surrounding environment, such as ant simulation.

10.4.5.3 Service

Mason is a great multiagent simulation toolkit for someone who wants to have a variety of visualizations and models but not a lot of built-in tools (in terms of pre-made simulation items). Mason is also great for having reproducible results across different platforms, something key to showing the scientific validity of a given simulation.

Visualization. As an accompanying suite to Mason, the user can have a large battery of visualization and modeling tools at his/her disposal. Visualization options range from a broad variety of textures in 2-D and 3-D. All visualizations can be recorded into QuickTime movies to be reviewed at a later time.

Data Analysis. Mason provides the ability to create charts and graphs from any aspect of a simulation and apply the data to a library of simulation models, all of which can run independently or within any other Java framework or application. The ability to live output stream data from a simulation is also available.

10.4.5.4 Application

Mason has aimed to be a toolkit used by social scientists and cellular biologists for research. Mason lends itself to being a small project toolkit rather than an industrial or military simulation system because of its very nonspecific nature.

Novel Uses:

- HeatBugs – a classic multiagent example popularized by the Swarm multi-agent simulation toolkit;
- Ants – an ant colony foraging simulation using two pheromones;
- Solar System – simple demo of the planets orbiting the sun;
- Keepaway – soccer keepaway simulator.

Research Uses:

- Anthrax propagation in the human body;
- Implementation of Dewdney's bugs;
- Solar System – simple demo of the planets orbiting the sun;
- Traffic and stoplight simulation.

10.5

Brahms – A Multiagent Simulation for Work System Analysis and Design

Brahms (<http://www.agentisolutions.com>) is a multiagent modeling language for simulating human work practice that emerges from work processes in organizations. The same Brahms language can be used to implement and execute distributed MASs based on models of work practice that were first simulated. Brahms demonstrates how a multiagent belief–desire–intention (BDI) language, symbolic cognitive modeling, traditional business process modeling, activity and situated cognition theories are brought together in a coherent approach for the analysis and design of organizations and human-centered systems. Brahms is being developed and used by the Work Systems Design and Evaluation group in NASA's Ames Intelligent Systems division.

10.5.1

Language

The Brahms language was designed to model people's work practices, as opposed to an organization's work process (Clancey et al., 1998). The concept of work practice is derived from the social sciences, in particular, business anthropology (Baba, 2006; Suchman, 1987). The study and observation of people in an organization leads to insights into the practices of people. It is these practices that implement organizational procedures and processes. The Brahms language was developed to model organizations at the practice level by representing each individual's activities as they are performed within a situational context of people, places, and systems and individual, group, and organizational practices (Sierhuis et al., 2007). To avoid misconceptions about the often ill-defined concept of work practice, we define what is meant by this (Sierhuis, 2001).

Work Practice The collective performance of contextually situated activities of a group of people who coordinate, cooperate, and collaborate while performing these activities synchronously or asynchronously, making use of knowledge previously gained through experiences in performing similar activities.

The Brahms language is a pure agent-oriented language (AOL). It is not a set of Java libraries enabling agent-based programming in the Java language. Instead, Brahms is a full-fledged multiagent language allowing the modeler to easily and naturally represent multiple agents. Brahms is both a modeling and simulation language and a language for implementing MASs.

Brahms is a BDI-like language. It enables easy creation of groups of agents that execute activities based on local beliefs. Below is a simple taxonomy of some important language concepts.

GROUPS are composed of
AGENTS having
BELIEFS and doing
ACTIVITIES executed by

PRECONDITIONS, matching agent's beliefs
 PRIMITIVE ACTIVITIES
 COMPOSITE ACTIVITIES, decomposing the activity
 DETECTABLES, including INTERRUPTS, IMPASSES
 CONSEQUENCES, creating new beliefs and/or facts
 THOUGHTFRAMES defined by
 PRECONDITIONS, matching agent's beliefs
 CONSEQUENCES, creating new beliefs

Agency. A Brahms model is always about the activities of agents. An agent is therefore the most central construct in a Brahms model. Brahms agents adhere to the attributes we associate with agency. They are autonomous, can be deliberative, as well as reactive, proactive, and bounded rational. The term “strong agency” is used for an agent that has the above properties and has humanlike behavior. An important aspect in modeling work practice is that a person’s bounded rationality is not only based on his or her limited problem-solving knowledge, but more importantly based on the social ability and practical knowledge created by context, environment, and practice. The Brahms AOL has the concept of an agent that allows for modeling and simulation this type of human behavior.

Organization. In the ACTS theory, actions and decisions of intelligent agents are a function of an agent’s cognitive architecture and knowledge (Carley et al., 1998; Newell, 1990). However, mechanisms by which an agent processes information, learns, and makes decisions are a function of not only the cognitive architecture of the agent, but also, importantly, the (social) position of the agent in the organization and the organizational group tasks in which the agent is engaged. Thus, the ACTS theory refocuses the attention of the researcher interested in organizations on the details through which the task and social environment influence the individual agent and the group adaptation and performance.

The concept of a *group* in Brahms is an organizational modeling concept similar to the concept of a template or class in object-oriented programming. A group represents a collection of agents that can perform similar work and have similar properties. A group defines the *attributes* and *relations*, the *initial beliefs* and *initial facts*, and the *activities*, *workframes*, and *thoughtframes* of members in the group. The difference with classes in object-oriented programming is that the relationship between a group and its members is not an IS-A relationship, but a MEMBER-OF relationship. This is why Brahms speaks of “agent A is a member of group G”, instead of “an instance of a group G”. An agent can be a member of one or more groups and will inherit attributes, relations, initial beliefs, initial facts, activities, workframes, and thoughtframes from the groups it is a member of. This way the modeler can model both functional roles and structural or organizational structures, as well as informal and social groupings (such as “those who meet at the water-cooler”).

Observation. In modeling people, an important aspect is the environment in which people live. Brahms provides an extensive conceptual environment and context model, called the *geography model*. The geography model includes *areas* and *areas-definitions* (i.e., types of areas) connected by *paths* over which an agent can

move from location to location. Areas can contain *subareas*. This way, a modeler can model any environment with a hierarchically connected graph of nodes called areas. Brahms also has a way to model artifacts within an environment, as behavioral, nonbehavioral (inanimate), and/or data objects and classes. An environment can thus be described in terms of locations (and sublocations) with its artifacts (objects) and people (agents).

The state of the environment (an agent's context) is represented as *facts* in the world. Agents can observe the facts in the world, which turns facts into *beliefs* on which they can act. Thus, an agent has the ability to notice, sense, and detect facts in the world, using a language construct called a detectable. Detectables are context specific, and, just as people are not omniscient, Brahms agents will only detect those facts in the world that are relevant to the activity they are performing. The modeler specifies which facts an agent can detect in an activity, as such modeling context-specific behavior.

Updating KB. In Brahms an agent acts according to its beliefs. An agent can update its belief set by concluding new beliefs based on its current beliefs, using either reasoning with production rules (thoughtframes) or by executing activities in situation-action rules (workframes). Furthermore, as mentioned above, agents can turn facts into beliefs by detectables in workframes. Last, but not least, agents can get new beliefs by communicating with other agents and objects.

Decision. Decision making in Brahms is done using thoughtframes. Thoughtframes (TFRs) define deductions, often referred to as production rules. TFRs are taken to be inferences an agent makes. TFRs do not perform actions, consume no time, and cannot be interrupted. The only allowable statements in a TFR is one or more consequences, concluding new beliefs. Because TFRs represent reasoning, the agent cannot create or change facts in the world in a TFR. A TFR consists of a variable declaration section, one or more preconditions, and one or more consequences (a conclude statement).

TFRs can be placed within composite activities (see below), allowing the modeling of problem-solving activities that take time. This is seen as: "While the agent is 'in' the activity, the agent reasons using its TFRs". Conclusions of new beliefs in TFRs can execute new TFRs and/or workframes (WFRs). Preconditions are similar for TFRs and WFRs, matching the beliefs in the belief set of an agent.

When the preconditions of a TFR match the beliefs of the agent or object, its consequences are immediately executed, similar to forward-chaining production rules. An important point is that preconditions for agents only match with the beliefs of the agent.

Communication. In Brahms communication between agents and between agents and objects is done by communicating beliefs. The communication of beliefs is done with a communication activity that transfers beliefs from one agent to one or several other agents, or to/from an object (information carrier). A communication activity is used to model different types of communications that can be observed in the world. Examples are: face-to-face conversations, reading or writing a document, or data entered into computers. An agent or object has to have the belief before it

can communicate (i.e., tell) the belief to another agent or object. The recipient agent or object will have the communicated beliefs added to its belief set.

A Brahms library is a reusable Brahms model that is provided with the Brahms environment. Brahms libraries define groups and/or classes with defined attributes and activities that can be inherited from in a user model. The *communicator library* is used for agents to send and receive FIPA (Foundation of Intelligent Physical Agents) communicative acts.⁷⁾ The communicator library implements the external activities for agents to communicate with other agents through FIPA communicative acts. The Communicator group and class specify a set of activities that can be used to create, read, manipulate, retract, and send CommunicativeAct objects.

Action. The central concept in Brahms, for the main purpose of modeling human behavior, is the concept of activity. An *activity* is an abstraction of real-life actions that takes time and helps accomplish a daily task. A model of an agent's activities describes what the agent actually does over time (i.e., its behavior) based on the causal relationship between the decision to perform an activity and the past and present state of its context. In describing people's real-life activities, each activity in the world takes time no matter how short. A person is always within an activity taking action. Sleeping is an activity, waiting for the bus is an activity, simply doing nothing is an activity. Indeed, being alive is an activity. The following key points can be made about activities.

- Reasoning is an activity taking time, not just an inference or deduction. Thus logical inferences happen within an activity.
- Activities might not involve goals and tasks. For example, answering the phone is an activity that might not be part of any specific task that is being accomplished. In fact, it might be an activity that is interrupting the task being worked on.
- Modeling activity behavior involves more than logical inferencing, namely, the representation of chronological activities that agents do.
- The activation of an activity is constrained by preconditions that are associated with an activity template it is part of (a WFR). For example, activities may have preferential start times, as expressed in preconditions for the template, which may refer to the time in hours, minutes, seconds, day of the year, and/or day of the week.
- An activity may be interrupted by a scheduled activity, such as going to lunch at noon. Time may change the priorities of activities and different people might do the same activities at different times.

In summary, activities are socially constructed engagements, situated in the real world, taking time, effort, and application of knowledge, with a defined beginning and end, while not necessarily needing goals in the sense of problem-solving tasks, and being interruptible, resumable, and able to be disrupted. For more discussion

7) <http://www.fipa.org/specs/fipa00037/index.html>.

about the theory behind activities in Brahms, we refer the reader to Clancey (2002) and Sierhuis et al. (2006). Brahms has the following different types of activities.

Primitive activities: These are the lowest-level activities for an agent. They are user-defined, take some time, but are not further specified in any detail. Parameters are time and resources. At any time during execution an agent is always executing some primitive activity. If an agent is not executing a primitive activity, one can say that the human-behavior model is underspecified.

Predefined activities: These are language-level primitive activities with predefined semantics (e.g., communicate, move, get, put).

Java activities: A Brahms Java activity is a user-defined primitive activity implemented as a Java class using the Brahms Java Application Programming Interface (JAPI). Java code may cause an action to happen completely outside the Brahms virtual machine (BVM) (e.g., a pop-up dialog that says “hello world”). A Java activity can also do things within the BVM. Java code can generate output parameter values and assign them to unbound variables in a WFR or generate new agents or objects within the Brahms model being executed. Java activities can also create new beliefs and facts, as well as interface to external systems.

Composite activities: These are user-defined detailed activities that are decomposed into subactivities. The lowest activity in a composite activity is always either a primitive, predefined, or Java activity. A composite activity describes what an agent does while it is “in” the activity. A composite activity can be interrupted when one of its lower-level activities gets interrupted.

An activity requires one or more WFRs to execute. Since activities are called within the do-part of a WFR, each is performed at a certain time within the WFR. The body of a WFR has a top-down, left-to-right execution sequence (Figure 10.5). The preference or relative priority of WFRs can be modeled by grouping them into ordered composite activities. The WFRs within a composite activity, however, can be performed in any order depending on when their preconditions are satisfied. In this way, WFRs can explicitly control executions of activities, and execution of WFRs depends not on their order but on the satisfiability of their preconditions and the priorities of their activities.

10.5.2 Environment

The Brahms Agent Environment (BAE) is a collection of tools for developing complex agent models for the purpose of simulating work practice or for developing MAS solutions to support the people that are part of a work practice. The BAE also supports the development of distributed agent-based solutions in support of an organization’s workflow, enabling a software engineering methodology referred to as “from simulation to implementation”. The Brahms tools that are included in the BAE are as follows.

- *Brahms Compiler (BC)*. The BC is a compiler for the Brahms language. The compiler compiles .b source files into .bcc byte code files.

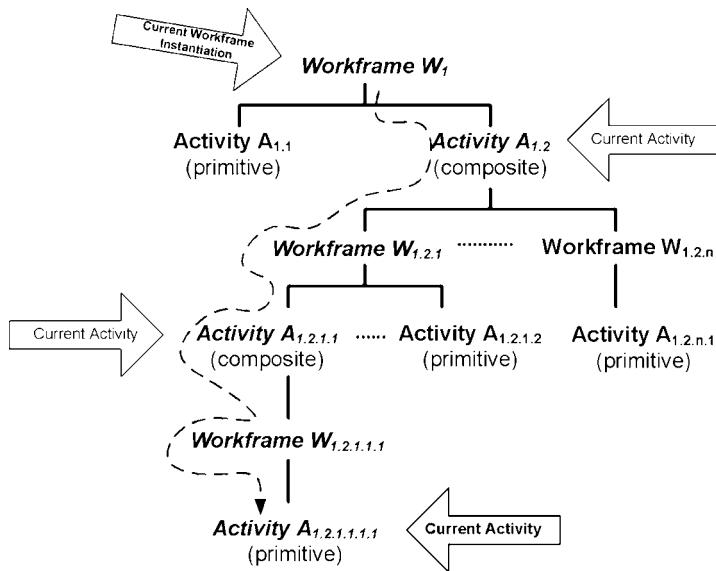


Fig. 10.5 Workframe-activity hierarchy.

- *Brahms Virtual Machine (BVM)*. The BVM is both a simulation engine for simulating Brahms models and an MAS execution environment for real-time agents.
- *Composer*. The Composer is a dedicated integrated development environment (IDE) for Brahms. It provides a project editor, model editors, a source code editor, and several postexecution displays. The modeler can both compile and run a model from within the Composer. The Composer is a useful tool for those who use Brahms for modeling and simulating work practice.
- *Brahms Eclipse Plugin*. The Brahms Eclipse Plugin is a plugin for the Eclipse development environment. The plugin is useful for those who use Brahms as a MAS development tool, and also develop and integrate with Java code.
- *AgentViewer*. The AgentViewer is a postexecution event timeline viewer for agent and object beliefs, workframes, activities, and thoughtframes, as well as interagent and interobject communications. The AgentViewer is a kind of debugging tool, although it is used after executing the model and is not an interactive debugging environment. During the execution of the model (either in simulation mode or in real-time mode), all events are stored by the event logger of the BVM in an ascii-formatted history file. Using the AgentViewer application, this history file can be parsed into a MySQL database that the AgentViewer uses to generate the TimeLine view.
- *Communication Display*. The Communication Display provides a network diagram of the agent and object communications. It shows the sender and receiver of the communications, as well as the number of beliefs and/or CommunicativeActs communicated.

Simulated Entities. The simulated entities in a Brahms model are agents and objects. Agents can represent either people, systems, or software agents. Objects can represent behavioral objects, such as computer systems, telephones, and other behavioral artifacts in the world, or inanimate artifacts, such as chairs, tables, and so on, as well as data objects, such as pieces of paper with information on them. In Brahms we try to model how work happens by interaction between people and systems in an environmental constraint by the context of that environment. Time moves forward in seconds (or a larger defined simulation-clock grain size), and the simulated entities get to behave based on their execution of activities and interaction with the environment and each other, as well as their reasoning about it. In Brahms, agents are not just simple entities but instead are complex behavioral and reactive entities representing how people get to do what they do.

Agent Type. Brahms has now two agent types: complex behavioral Brahms agents and external agents implemented in Java using the Brahms Java Application Programming Interface (JAPI). External agents are most often used to interact with external systems during a simulation, such as databases or graphical user interfaces. Besides agents, Brahms also allows the modeling of complex behavioral artifacts as objects with capabilities similar to those of agents. The choice to model a real-world entity as an agent or an object is up to the model builder. The defining difference is what notion of agency the modeler is using. For example, whether an autonomous robot is modeled as an agent or as a behavioral object depends on the modeler's definition of an agent and on the other entities in the model, as well as what is needed to model the robot's behavior and its interaction with the other entities.

Similarly, whether one implements an agent as a Brahms agent or an external Java agent depends on the purpose of the agent in the model. Brahms agents are “heavy” in terms of memory and efficiency. On the other hand, Java agents do not have beliefs, and no activity and/or reasoning behavior. Therefore, behavioral agents, such as models of people, are better implemented as Brahms agents.

Simulation Environment. Brahms is a discrete-event multiagent simulation engine. The simulation clock is event-driven to make the agents execute in the most efficient way. Each agent has its own discrete-event activity-based inference engine running in a separate Java thread. To keep time and events synchronized between each agent in a simulation, there is a central event scheduler and simulation clock that advances the simulation. All events internal to an agent, such as the creation of new beliefs through reasoning and action are kept internal to each agent. All events that need to be distributed to other agents, such as the creation of facts in the world and communication of beliefs to other agents, are distributed through the central event scheduler (Figure 10.6).

Communication. Brahms is completely implemented in Java and the BVM runs “on top of” the Java Virtual Machine (VM). In simulation mode, agents cannot be distributed over multiple BVMs and all agent communication is done through a dedicated Brahms communication layer. However, Brahms can also be run in a distributed real-time mode. In this mode multiple BVMs can be connected through a distributed directory service. Every BVM then has its own local agent directory, which connects up with other agent directories managed by an appli-

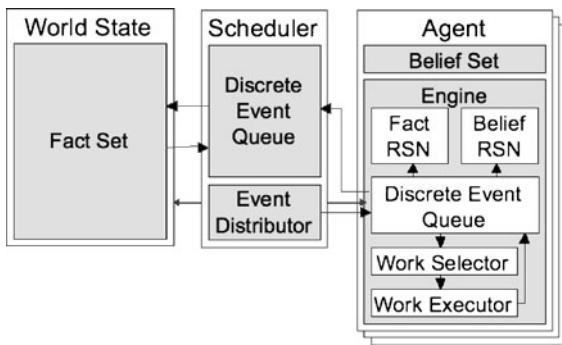


Fig. 10.6 The central event scheduler.

cation service. This way distributed agents running over a network in different BVMs can find each other. A dedicated secure socket layer (SSL) allows agents to communicate securely over the network.

Mobility. Each BVM has its own geography model. However, this model is not distributed over multiple BVMs. This means that Brahms agents cannot move from one BVM to another.

Adaptivity. Brahms agents are BDI agents. Adaptation of behavior is accomplished by changing the beliefs of agents and facts in the world. Today, the Brahms language is not reflexive, meaning that agents cannot modify or add new activity behavior based on a learning algorithm. All activities an agent can perform need to be modeled a priori.

10.5.3

Service

Visualization. Brahms does not provide a visualization library for visualizing agent behavior during a simulation. However, since the BVM is written in Java and runs within the Java VM, integration with Java is supported through Java activities and external Java agents written using the extensive Brahms JAPI. It is thus possible to integrate a Brahms simulation with any Java visualization library.

As described above, Brahms provides a postsimulation visualization of agent interaction and activity performance through an agent timeline and communication display tool. Figure 10.7 shows a screenshot of this postsimulation visualization environment. The pane on the left-hand side of the figure allows the user to select which agents and objects to view. The middle pane is a timeline showing at the top agent Alex_Agent performing activities and communicating to the Boa_ATM object. On the right is the Communication display, showing the communication of beliefs between several agents and objects.

Data Analysis. The AgentViewer uses a MySQL database to retrieve simulation information for each agent. This MySQL database can also be used to easily perform data analysis using Excel. A Brahms modeler can instrument an agent with

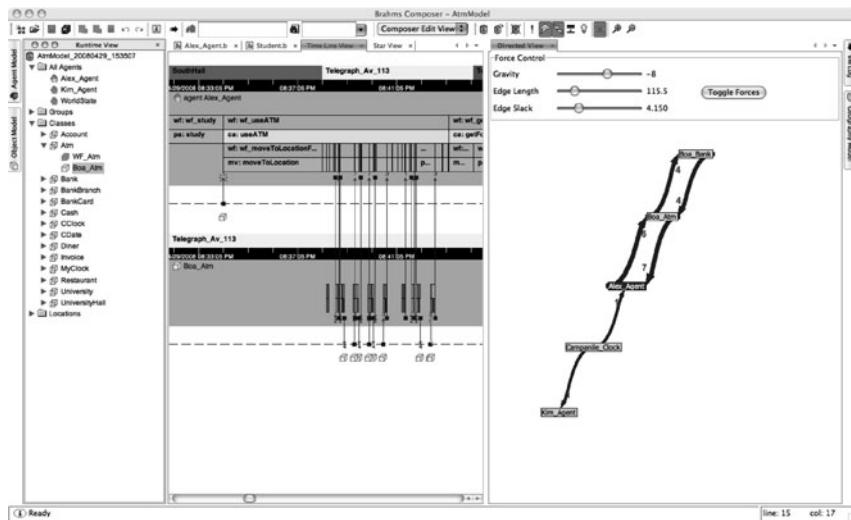


Fig. 10.7 Brahms AgentViewer timeline and communication displays.

beliefs or facts that keep track of statistical measures during a simulation. Using a MySQL ODBC connection with Excel or other spreadsheet tool, the data analyst can easily retrieve these statistical measures from the database using simple SQL statements. From this calculations and graphs can be easily produced.

10.5.4

Application

Brahms has been used in many simulation projects – at the NYNEX and Bell Atlantic phone companies simulating the T1 ordering process, at NASA modeling and simulating the Apollo astronauts on the lunar surface, a day in the life of the astronauts on the International Space Station and flight controllers in the Mission Control Center (Sierhuis et al., 2000; Sierhuis, 2001; Sierhuis et al., 2002; Sierhuis and Clancey, 2002; Sierhuis et al., 2006, 2007; Acquisti and Sierhuis, 2002), by students at the Universities of Twente (Bruinsma and de Hoog, 2006) and Amsterdam (Netten and Someren, 2008), and by several research organizations throughout the world. Furthermore, at NASA Brahms has also been used to develop a distributed multiagent human-robot exploration system (Clancey and Sierhuis, 2000; Hirsh et al., 2006), as well as a simulation of collaborative traffic flow management for future concepts of the US National Airspace (Wolfe et al., 2007, 2008), and most recently to simulate and implement an intelligent workflow application for NASA's Mission Control Center (Clancet et al., 2008).

10.6

CASESim – A Multiagent Simulation for Cognitive Agents for Social Environment

CASESim is a multiagent simulation environment being developed at Trinity University. The system has been designed primarily to simulate spatial social systems. It is written in Java using RMI (Remote Method Invocation) for communication between clusters of machines. The system uses kD-trees⁸⁾ (k -dimensional trees) for the organization of the spatial decomposition of the simulation space between machines and for efficiently locating entities on a single machine. The initial construction of the system was begun because of certain requirements in the simulations that were not well met in other systems. The inherent nature of CASESim composed of large numbers of autonomous agents makes it well suited for social simulations. Of the existing MASs, such as RePast and Mason, few systems have encompassed the need to be both robust and distributed. The key to social simulations, and the part that makes them difficult for multiagent applications, is their need for scalable large systems. This translates to a need for serious computational power to handle the simulation. With these requirements only a highly distributed system would be able to handle running a simulation of the size required of this social research. The system would also need a sufficiently intuitive interface for the researchers as well. Since most social scientists do not have a background in computer science, they are unable to deal with large and esoteric systems. Thus, to allow them to effectively use a MAS as a tool for their research, a simple and user-friendly interface needs to be developed. An important component of this user interface is the visualization of the data. Social science results are often complicated and numerous; thus a tool to interpret and visualize these data would be required for the system to be effective as a social multiagent simulation.

The organization of CASESim is shown in Figure 10.8. The system is based on a service-oriented architecture design so that components can be easily added or swapped out. This figure shows that the system is organized in three different levels. The primary simulation framework sits in the middle level and communicates with the services above and the data stores below. On the middle level there are two primary components. A master server oversees the functioning of the simulation and controls the behaviors of a set of slave servers. Communication between these components occurs through RMI. Because the system is intended primarily for spatial simulations, the master machine uses a kD-tree in the x , y -plane to do a spatial decomposition of the simulation region. Each of the slave machines is referenced as a leaf in this tree. Organizing the slaves in this way allows for an appropriate combination of flexibility and efficiency. The way in which kD-trees can have variable split locations allows for the possibility of load balancing as well.

In addition to handling the synchronization between a master and multiple slave machines, the CASESim framework also abstracts the process of agents talking to one another. The message passing between agents is done through a class that

⁸⁾ A kD-tree is a space-partitioning data structure for organizing points in a k -dimensional space.

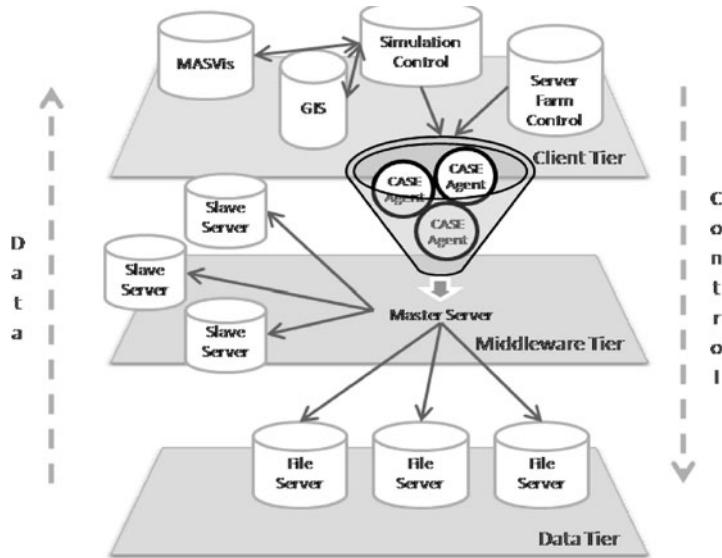


Fig. 10.8 The three-tier CASESim system architecture. (The simulation itself occurs on the middle tier, where a master server directs slave servers that control the agents. The top layer contains various services that constitute the user interface. The bottom tier has data resources for the system.)

locates agents. This allows the locator class to pull data from other machines in whatever way it wants. To optimize performance on the types of clusters that are common in educational settings, the CASESim system lumps machine communication at the beginning of each time step in an attempt to minimize the impact of network latency. For intermachine communications, this means that at the beginning of the step each machine will request the information that it expects will be needed during that step.

Currently agents for CASESim and the models they are part of are coded in Java. This provides complete flexibility, but also imposes certain restrictions on who can create new models. Work is under way to develop methods of building models and agents that do not require code-level modifications. What is discussed below will only focus on the current usage where programmers must be involved in the development process to get a new model running.

As a note on terminology, everything in a CASESim model is a subtype of entity. This includes agents and other elements of the environment. The system does not specifically distinguish between agents and other elements of the environment. The difference is in the logic behind them. The terms entity and agent are used in this section to reflect the idea that all things in the simulation are entities, but only agents are given significant logic in their behaviors.

In order to illustrate some of the capabilities of CASESim, an implementation of Sugarscape (Epstein and Axtell, 1996) will often be used as an example. Sugarscape

was first proposed by Thomas Schelling as a way of modeling what he thought was an inherent pattern within segregation (Schelling, 1969). The Sugarscape domain was later greatly expanded by Epstein and Axtell (1996) as a testbed for growing societies. The Sugarscape domain was chosen because it covered the necessary functionality and its results are also very well documented. This allowed us to create the simulation according to the specified approach and to easily and quickly validate the system's results. Because Sugarscape is also a well-known example with reasonably simple rules, it serves well for illustrative purposes.

10.6.1

Language

Observation. The first thing that happens in each discrete time step in CASESim is that the system asks each of the entities in the system for the size of the spatial region it should be able to observe. It then uses the local spatial tree to efficiently create a list of the entities within that distance. This list is given to the entity. The system also allows any entity to gather information on other entities based on a unique identifier. This allows agents to retain communication with elements of the environment after they have moved too far apart to be in the normal search range. The system handles the communication of entities between different machines in a manner that reduces the total number of communications in order to optimize the system for clusters with standard networking.

Updating KB. The agents in the system can use the information collected either from their spatial surroundings or from previously encountered more distant entities to update their KBs. There is a synchronization point in the code after the entities have updated their KBs so that none of the agents will begin changing their states until after all the information has been collected.

Decision. The decision process is written by the developer in Java code. As such, any method can be applied to the decision making. This is the primary area of work in CASESim right now. Tools are under development that will allow users without programming knowledge to put together agents of arbitrary complexity that incorporate different techniques for learning and adaptive behavior.

Communication. CASESim uses the KQML standard for agent communication. Every message includes information about the sender, the receiver, the content of the message, and the time step at which the message is sent.

Action. The last thing that the agents do in each time step is to update their internal state based on the action they are taking. This can include generically updating their position or doing other updates that are specific to the discipline of the simulation.

10.6.2

Environment

Simulated Entities. The fundamental type for simulations in CASESim is the entity. There are a number of methods that developers have to implement in order to have

a fully functional entity. For static elements of the environment many of these can be left empty.

- Rectangle2D getBoundingSpace();
- Point2D getPosition();
- int getSize();
- void gatherData(List<Entity> list, EntityFinder finder);
- double searchRadius();
- void update(int timeStep);
- int getID();
- List<Entity> getNewlyCreatedEntities();
- List<KQML> getMessages();
- void receiveMessage(KQML message);
- void receiveMessages(List<KQML> messages);
- void packageData(EntityData data);

The first three methods provide the spatial information that is required for operation in the CASESim environment. The `gatherData` method is called when the entity should update its KB. This method is passed information on all of the entities in its field of vision as well as an object that allows it to get information on other entities, including those that might be on other machines. The `searchRadius` method simply tells the system the range of vision for this entity.

The `update` method is called in the decision part of the step after all of the entities have updated their KBs. The entities also perform their actions and send any messages in the `update` method. The `getID` method should return a unique identifier for each entity in the system. This is what is used by the `EntityFinder` in the `gatherData` method to locate entities that are not near the current entity.

The rest of the methods are used by the system for carrying out the actions of the entities and performing communication between them. In many simulations, agents have the ability to alter the environment by creating new entities. Any agent doing this in CASESim will return the appropriate list of objects from `getNewlyCreatedEntities`. Those entities are then placed on the proper slave server. Three of the methods deal with communicating KQML messages between agents. The last method, `packageData`, is used for communicating to services. If the system determines that a particular agent is needed by a service during a particular time step, this method will be called to pack the data that that entity needs to send to services.

Agent Type. In CASESim an agent is simply an entity with a complete implementation of the logic for that particular agent type. What type of agent one is working with, reflex, model-based, goal-based, or utility-based, is determined by the code that is added to the `update` method. The Sugarscape example displays simple reflex agents. They look at the arrangement of sugar around them and move accordingly. As will be discussed below, this has also been augmented to a model-based agent type where agents communicate with one another and can decide to move to the location of one of several “friend” agents if their location appears favorable.

Goal- and utility-based agents can be coded as well given that the whole agent is written in Java and is therefore Turing complete inside the memory restrictions of

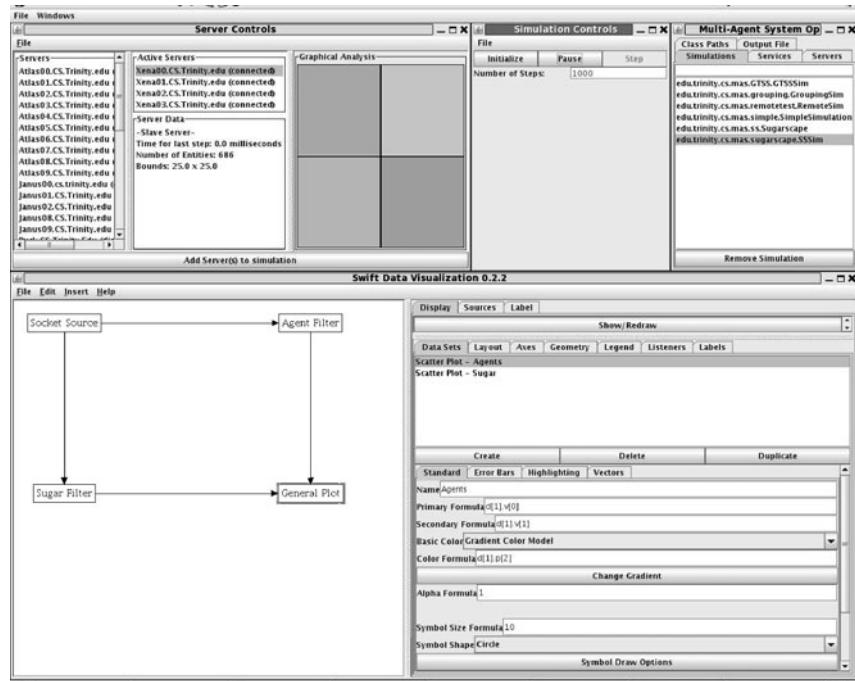


Fig. 10.9 User interfaces of CASESim. (Four windows that are part of the default user interface for CASESim. Along the top are windows for adding servers to the simulation, selecting and controlling simulation scenarios, and editing options. The large window at the bottom displays SwiftVis, the analysis and visualization service that is discussed in Section 10.6.3.)

the machine. However, the primary audience for CASESim is social science simulation and having them build up these complex agents in a standard programming language is less than ideal. In fact, even forcing them to code to create reflex or model-based agents is something that should be avoided if possible. For this reason, one of the active areas of research for CASESim is the creation of a graphical interface that will allow nonprogrammers to create domain-specific agents of any type without having to write the code for them. The graphical interface for this system will be modeled after the analysis and visualization system that is used with CASESim. This system, SwiftVis, is discussed below in the services section.

Simulation Environment. The organization of the CASESim environment was described above. A sample view of the default user interface is shown in Figure 10.9. When CASESim is started, the user gets to select the machines that are participating in the simulation from a list that can be modified in an options dialog. Once the machines have been added to the simulation, the user selects the simulation that they wish to run. The list of possible simulations is also provided in the options dialog and can be edited. The dynamic loading capabilities of Java make it

easy for users to create whatever simulations they wish and add them to the list of options. Once selected, the simulation can be initialized and started. While the simulation is running, the user has the option of pausing, resuming, or stopping the simulation.

When the simulation is initialized, the master machine will ask the simulation for the spatial extent that it will cover and then distribute that area to the other machines involved in the simulation using a kD-tree. The master places each machine in a manner that keeps the tree as a whole balanced. Once a slave instance has been started on each of the slave machines, the master asks the specific simulation code to generate entities that are sent to the individual slave machines. The generation of entities is done on the master instead of on the slaves to keep life simpler for those writing code for new simulations. Writing decentralized code for an unknown number of machines with unknown boundaries is much harder than writing centralized code, and it is easy for the master to use the kD-tree to hand each entity to the slave server that it belongs to. While there is significant network overhead in this process, it only has to happen once when the simulation is initialized, and for that reason this is tolerated to keep life simpler for those developing domain-specific codes.

Once a simulation has been initialized, the user can choose to bring up the interfaces for other services that have been registered with CASESim. Registering services is just like adding simulations. It is handled in the options and involves dynamic class loading. Services that are not written in Java and are not integrated into the environment can also be used. They will simply have to be started separately and told to make a socket connection to get the data. Even the services that can be integrated into the environment and are normally brought up by selecting a menu option can be connected in this way. One advantage of this is that those services can be brought up on computers other than the one serving as the master for the simulation.

Communication. After making a decision, the agents can communicate to other entities by sending messages. These messages are queued between machines for efficiency and communicated in bulk at the end of the step. They are not used by the receiving entity until after all actions have completed. They will become part of the observations in the following time step. This prevents them from causing race conditions in the parallel environment.

Migration. In many distributed MASs, the agents are allowed to migrate from one machine to another. This is often done to gain access to certain resources that are available on that machine. The CASESim system is different, but migration is still a very significant aspect. In CASESim the agents are completely unaware of the machine they are running. This allows the simulation to be run across any number of machines without impacting the code for the agents. However, agents can move from one machine to another. Because the machines are allocated agents by spatial decomposition, normal agent movements can cause an agent to switch machines. The CASESim framework does this completely behind the scenes so the agents have no knowledge that the move has taken place.

Another feature of CASESim that is under development and leads to the migration of agents between machines is load balancing. One of the advantages of kD-trees is the flexibility in the placement of the dividing line at each node. These boundaries can be moved around to try to make the loads on different machines more equal. When boundaries between the machines move, they can change which machine should hold a particular agent. This type of migration is also handled by the framework in a manner that is completely hidden from the agents.

Adaptivity. Work is currently under way to add different learning models to CASESim. Neural nets and Q-learning have basic implementations at this point. One of the more interesting goals of the nonprogramming agent creation task is to bring these learning models to bear in a graphical interface that will allow users to have complex agents with adaptive behaviors without ever writing a line of code.

10.6.3 Service

CASESim is built with a service-oriented architecture style in mind. The primary simulation code does not provide user interface code. The features that the user interacts with are provided by other code elements that connect to the main simulation program and exchange data with it. This makes it easy to vary the services that are provided by the system. Figure 10.9 showed the default user interface for CASESim along with a window for one of the standard services in a sample simulation of Sugarscape. This interface is complex but flexible. Simplified interfaces could be built to provide entry points into CASESim when users needed fewer options. For example, one could picture an interface written for those situations where only a single machine is to be used in the execution of the simulation.

One of the primary services that is already in use is a data analysis and visualization system called SwiftVis⁹⁾. The SwiftVis system was originally written to work with N-body planetary dynamics simulations done using the Swift integration package. However, the design has proven flexible enough to be used with multiple other numerical integrators. At their heart, multiagent simulations are extremely similar to N-body dynamics simulations. The only differences are the rules that the bodies in the simulation follow. For this reason, SwiftVis works well for doing the analysis and visualization for CASESim as well.

Before looking at the capabilities of SwiftVis and how it integrates with CASESim, there are a few details of the way in which CASESim gives data to services that are worth exploring. These features have been added in order to expand the flexibility of the services that CASESim can integrate with as well as to improve efficiency. Recall that one of the primary objectives of CASESim is to enable large-scale, distributed multiagent simulations. When CASESim is doing what it does best, no single computer should be able to hold the entire data set of agents that are distributed across multiple machines. For this reason, users need to be able

9) More information on SwiftVis can be found at
<http://www.cs.trinity.edu/~mlewis/SwiftVis/>.

to have some control over what data are sent to each service. The determination of what data are packaged from each entity happens in the code for the entity. By default they will package their location and ID number, but additional values can be added.

More significant is the choice of which entities to do the packaging and sending of data for. To provide full flexibility in this, the service can send CASESim a string that is evaluated as a boolean expression. The string can include method invocations that will be made on the entity using Java's reflection capabilities. Only the entities for which this expression evaluates to true will have their data packaged and sent across the network to the service. In this way, users have complete flexibility in what entities go to the service and they can optimize the data that are sent across by pairing the data down to only what is needed.

CASESim also provides flexibility in how the data are packaged and sent across the network. When a service connects, it tells CASESim what type of data packaging it wants and how often it wants simulation updates. The most efficient data packaging method is a binary format, but data can also be packaged in a standard text format or using XML. In this way there is greater flexibility with the services that are allowed.

To illustrate the capabilities of some of the services included with CASESim we use a Sugarscape simulation with a 50×50 grid world and 250 agents. At the beginning of the simulation, each agent is randomly assigned sugars that can afford them to move 5 to 15 spaces without sugar before dying. Vision and metabolism endowments are randomly distributed in the population. Therefore the population of agents is heterogeneous (i. e., not all agents are alike). Each agent's vision varies from one square ahead (bad vision) to six squares ahead (good vision). Each agent's metabolism is from one unit of sugar per step (slow and good) to four units of sugar (fast and bad). Finally, as sugar is eaten, it grows back in the landscape at the rate of one unit per time step.

Visualization. One of the first things that a researcher needs to be able to do when running a MAS simulation is to see what the agents are doing. For this reason, CASESim has integrated support to work with SwiftVis as a service. SwiftVis allows users to quickly plot the locations of entities in the world using numerous plot styles. Many of these plot styles were originally developed to work with N-body systems, but they function nicely with MAS data as well.

Figure 10.10 shows the SwiftVis Sugarscape visualization at four different time steps. At time step $t = 0$, the simulation begins with 250 agents randomly distributed on the Sugarscape. The color of the dot indicates the agents' original wealth (dark color means rich and light color means poor). There are two sugar lands in the world. Each sugar land has four layers; the outermost layer contains only one unit of sugar for each cell, and then two, three, and four, toward the innermost layer (the center of the mountain). Some agents happen to land on the rich sugar mountains and thus are born into sugar wealth, while others have bad luck and are born in the poor area of the badlands (the blank space around the two sugar lands). At the beginning of the simulation, things are a bit chaotic as the agents rush around looking for sugar, and many of them who were born in badlands die of starvation.

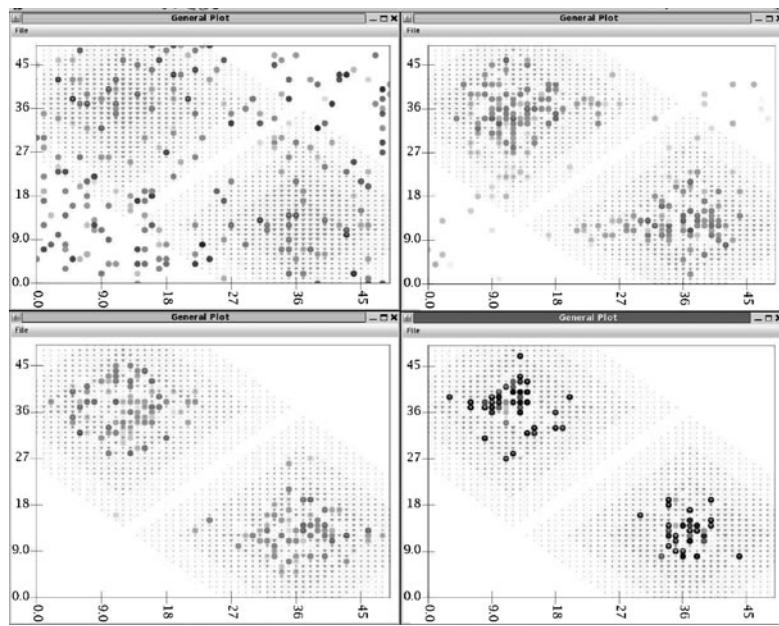
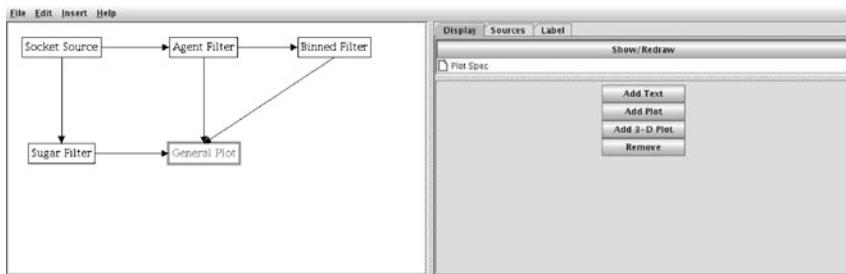


Fig. 10.10 CASESim screenshots of Sugarscape. [The simulation at four different times during a simulation (step $t = 0$, $t = 10$, $t = 22$, and $t = 93$ in clockwise order.) The sugar is drawn as triangles with the size indicating how much sugar is at that location. The agents are circles and their color is an indication of how much wealth they have accrued.]

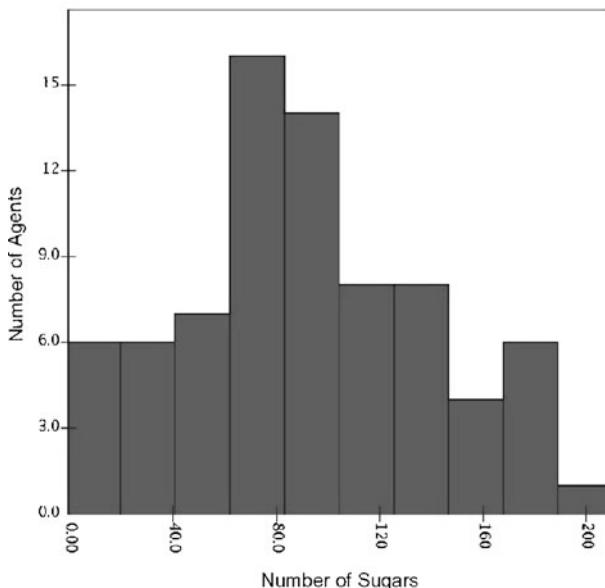
Pretty quickly, however, a pattern begins to emerge. At time step $t = 10$, the agents discover the two sugar-rich mountains and begin to cluster around them. This geographic density is getting more visible right away. At time step $t = 22$, the structure of the two sugar lands leads to a splitting of the agents into two groups. When the simulation goes on, at time step $t = 93$ and longer, we see a stable form of two self-developed organizations. Agents are able to stay on the two sugar mountains and get richer and richer.

These plots use the scatterplot option in SwiftVis. It currently also includes the ability to do surface plots, histograms, and barplots in addition to a set of other more specialized plots. One of the primary advantages of the SwiftVis architecture is that it is highly extensible. This includes the ability to add in new plotting styles for both 2-D and 3-D plotting. If it is found that different types of plots are helpful for MASs, those styles of plots can be added with little effort. Indeed, thanks to the dynamic capabilities of Java, users can write their own plotting styles if they have specific requirements.

Data Analysis. The true power of SwiftVis lies not in the plots that it can generate, but in the way it incorporates data analysis along with plotting capabilities in a manner that requires no programming. The main SwiftVis interface shows a data flow diagram where the information from the simulation, or some other source such



(a) A Swift Vis Configuration



(b) A Histogram Result of Sugarscape

Fig. 10.11 A SwiftVis configuration and a histogram of wealth produced by binning the data from the agents.

as a file, is passed through various filters before reaching a sink such as the plots. The filters can transform data in whatever way the user wants and output altered or completely new data elements. The flow of data does not have to be a simple linear configuration either as all sources and filters can output to multiple elements and many filters and sinks can accept input from multiple sources. A simple example of this is seen in the bottom window of Figure 10.10. Here the socket source, which pulls data from the simulation, is connected to two different filters that separate the two types of entities so that they can be processed separately.

Figure 10.11 shows SwiftVis being used to plot the environment in a Sugarscape simulation as well as a histogram of the wealth distribution of the agents in that

simulation. The selection filters are used to separate the agents from the sugar entities. The binned filter can bin data in arbitrary dimensions and arbitrary ways. In this case it is simply used to produce the distribution of wealth values. All of this information is fed into the plot element, which draws the graph that is shown. This plot can be viewed while the simulation is running, and it will be automatically updated as new data are received from the simulation.

More complex analyses can be performed as well. There are currently more than 20 different types of filters in SwiftVis. Some of these are very specific to certain tasks. Others are quite general. For users who can program, there is a filter, a source, and a plot style that allow scripting in any of the languages that have support for the Java 6 scripting engine. As with the plots, it is also easy to code up new filters that can be used for specific reasons. Filters that specifically support the needs of MAS researchers will be added as the need for them is discovered.

10.6.4

Application

To date CASESim has been used for doing social simulations and in the development of cognitive agents. The most significant social simulation work has involved simulating the mortgage market to uncover reasons behind the clustering of foreclosures seen in various neighborhoods. Instead of going into the details of those models, let us instead look at an enhancement to the Sugarscape model in which agents begin to utilize their communication capabilities.

Figure 10.12 displays two Sugarscape simulations with minor variations. One includes the capability of communication among agents, and the other is standard Sugarscape, which restricts the information individual agents can gather to that which is observable. Both of these simulations differ from the earlier Sugarscape model in that two extra groupings of sugar have been added. These groupings cover a smaller area but contain more sugar at each location and have higher regeneration rates. For this reason, they should be the most favorable spots on the map assuming there aren't too many agents there.

The top plots in the figure show the standard implementation without communication at the beginning and end of a simulation. In this model the agents are basically hill climbers who move around to find the best pile of sugar that they can see. Because of this, the only agents who get to the taller piles of sugar are those who were randomly created on them or near enough to them that they wandered onto them before dying. Basically, the hill-climbing agents are good at finding local maxima but have problems finding global maxima.

The bottom plots of the figure show the version with communication. At the beginning of the simulation each agent is assigned three random "friends". Note that this means that while one agent may have three friends, that doesn't necessarily mean that those friends are also friends with each other, and friendships are defined as one-directional, so that an agent's friends may not have the original agent as their friend as well, thus ensuring that a given agent's friends are as unique and varied as possible. At each step, agents collect information about their observable

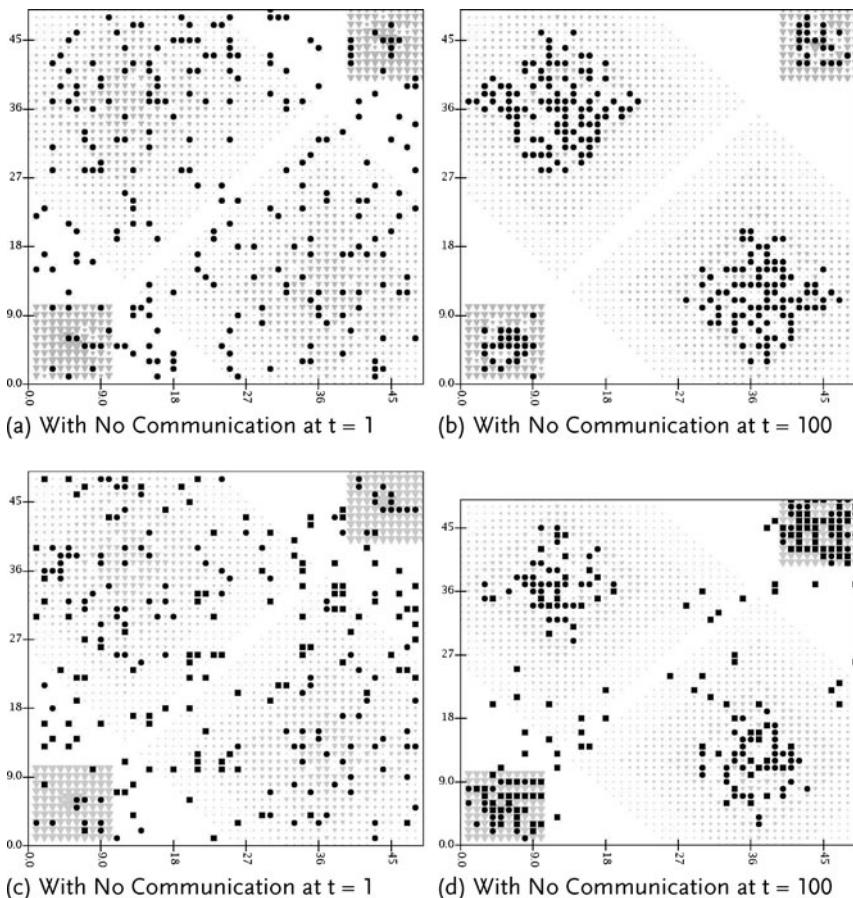


Fig. 10.12 Two versions of a modified Sugarscape simulation at the beginning, left and end, right of the simulation. (The top plots show a standard Sugarscape where only the distribution of sugar has been modified from the original form. The bottom shows a simulation where agents can communicate through message passing and will move toward their friends if the friend appears to have a superior location.)

environment and inform each of their friends about where they are and what the density of the sugar is across their range of vision. Each friend may then later make individual decisions based upon the information and may decide to start moving toward the foreign location if they deem it worth the trip. This action is portrayed by the shape of the agent's representation in Figure 10.12, such that circles describe agents wandering in their local environment and squares describe agents that are moving toward a foreign location.

The distance between the smaller mountains and the larger mountains is sufficiently large that it is improbable for most agents that make it to one of the larger

mountains to ever stray to the smaller, yet more dense, areas. However, once communication is added in, then many agents who have friends in the high-density regions begin to move away from their original locations and toward the smaller mountains. However, this can get a little out of hand, as most agents will want to get to that area, and some discover once they reach it that it isn't large enough to support the entire population. As such, once the smaller mountains begin to get overcrowded, some agents will begin to move back toward the larger and less populated mountains (note that without giving memory to agents, this can lead to a potential semicycle, in that after friends leave the smaller mountains, it then becomes less crowded and contains more sugar, which may draw them back). The final image of the communication run of the simulation demonstrates a much higher number of agents in the smaller mountains than will happen in a typical simulation without communication and shows a number of agents moving toward and leaving the mountains. Thus, this simulation demonstrates some of the ways in which communication across agents may be able to affect simulations.

10.7

Conclusion

The platforms reviewed in this chapter suggest that agent-directed simulation (ADS) seems a promising approach and is being increasingly used to identify and analyze the properties of complex systems. This benefits both the MAS community and the simulation community, as well as the social simulation community.

- For the MAS community, ADS has been applied to a wide range of MAS research and design problems, from models of complex individual agents employing sophisticated internal mechanisms to models of large-scale societies of relatively simple agents that focus more on the interactions between agents.
- For the simulation community, ADS provides a new way of organizing and managing large-scale systems by utilizing the capabilities of networked machines to power these systems. On the other hand, agent-based simulation presents several challenging new problems requiring the development of new theory and technologies to the traditional simulation community. Such challenging problems include the scalability of the system, the mobility of agents, the load balancing among machines, and the adaptivity of agents.
- For the social simulation community, ADS is an approachable methodology as it provides social science researchers with highly modular interfaces with data visualization tools built in. This eases the technical barriers bringing by the need of programming and computer networking skills that most social science researchers do not have. With ADS, social science researchers can understand and analyze real social systems with properties of self-organization, robustness, and openness. In addition, agent-based social simulation supports modeling and implementation of proactive behavior, which is important when simulating humans and animals. After all, it is

often more natural to model and implement humans and animals as agents than objects.

Though very helpful, ADS has a disadvantage in needing more computing resource. Also for any ADS platform, there is a need to improve the tradeoff between ease of use and generality of platforms. Therefore developing ADS for effective and robust simulation is a long-term goal (Macal and North, 2006).

- An ADS system normally uses more resources, both from computation within individual agents and communication between agents (Davidsson, 2000). This may lead to less efficient simulations. Compared with traditional discrete-event simulation, ADS is a time-driving simulation, and this often requires multiple synchronization points within a single simulation time step. For example, the simulation probably needs to be synchronized to allow every agent to finish its action before they all move to the next time step. Agents' observations may also need to be synchronized in order to have them all observe the same environmental state. The synchronization would slow down the simulation considerably.
- Any ADS platform needs to look for ways to improve the tradeoff between ease of use and generality of platforms (Railsback et al., 2006). One way to do so is by making common tasks (e.g., displaying spaces and agents, adding and removing agents from the scheduled list, looping through lists) easier to program, for example by using higher-level (NetLogo-like) code. High-level tools that hide some functions are very helpful and still allow models to be reproducible and flexible – if the tools are thoroughly documented (so users know how they work in full detail, which they do not with NetLogo) and can be overridden. Another technique is better use of standardized design patterns: making as many methods as familiar as possible to users. Graphical or menu-driven tools for adding observer capabilities can be powerful, as illustrated by NetLogo and platforms such as Borland Delphi.

Acknowledgements

This work was supported in part by the US National Science Foundation under Grants IIS 0755405 and CNS 0821585.

References

- F. Padberg. A discrete simulation model for assessing software project scheduling policies. *Software Process Improvement and Practice*, 9, 127–139, 2002.
- A. Cabrera, F.E. Cabrera, and S. Barajas. The key role of organizational culture in a multi-system view of technology-driven change. *International Journal of Information Management*, 21, 245–261, 2001.
- A. Acquisti and M. Sierhuis. A Work Practice Model of a Day in the Life Onboard the International Space Station. *Computational and Mathematical Organization Theory Conference (CASOS 02)*, 2002.

- M. Baba. Anthropology and Business. *Encyclopedia of Anthropology, Birx JH (Ed.)*, Sage Publications, pp. 83–117, 2006.
- M. Bell. *Service-Oriented Modeling: Service Analysis, Design, and Architecture*, Wiley, New Jersey, NJ, 2008.
- F. Bousquet and C.L. Page. Multi-Agent Simulations and Ecosystem Management: A Review. *Ecological Modeling*, 176(3), 313–332, 2004.
- E. Bonabeau. Agent-Based Modeling: Methods and Techniques for Simulating Human Systems. *Proceedings of the National Academy of Sciences*, 2002.
- J. Bradshaw, A. Uzzok, R. Jeffers, R. Suri, P. Hayes, M. Burstein, A. Acquisti, B. Benyo, M. Breedy, M. Carvalho, D. Diller, M. Johnson, S. Kulkarni, J. Lott, M. Sierhuis, and R. Van Hoof. Representation and Reasoning for DAML-Based Policy and Domain Services in KAoS and Nomads. *Proceedings of the Second International Joint Conference on Autonomous Agents and Multi-Agent Systems*, 2003.
- G. Bruinsma and R. de Hoog. Exploring Protocols for Multidisciplinary Disaster Response Using Adaptive Workflow Simulation. *Proceedings of the Third International Conference on Information System for Crisis Response and Management (ISCRAM'06)*, 2006.
- G.N. Buchler, L. Light, and M.L. Reder. Memory for Items and Associations: Distinct Representations and Processes in Associative Recognition. *Journal of Memory and Language*, 59, 183–199, 2008.
- F. Buschmann, J.C. Kel, R. Meunier, H. Rohnert, and H. Stahl. *A Pattern-Oriented Software Architecture – A System of Patterns*, Chichester UK, 1996.
- K.M. Carley, M.J. Prietula, and Z. Lin. Design Versus Cognition: The Interaction of Agent Cognition and Organizational Design on Organizational Performance. *Journal of Artificial Societies and Social Simulation*, 1(3), 1998.
- K. Carley and C. Schreiber. Information Technology and Knowledge Distribution in C3I Teams. *Proceedings of the Command and Control Research and Technology Symposium*, Naval Postgraduate School, Monterey, CA, 2002.
- K. Carley, D.B. Fridsma, E. Casman, A. Yahja, N. Altman, C.L. Chen, B. Kaminsky, and D. Nave. BioWar: Scalable Agent-Based Model of Bioattacks. *IEEE Transactions on Systems, Man and Cybernetics, Part A*, 36(2), 252–265, 2006.
- A. Casal, C. Sumen, T.E. Reddy, S.M. Alber, and P.P. Lee. Agent-Based Modeling of the Context Dependency in T Cell Recognition. *Journal of Theoretical Biology*, 236(4), 376–391, 2005.
- D. Chen, B. Jeng, W.P. Lee, and C.H. Chuang. An Agent-Based Model for Consumer-To-Business Electronic Commerce. *Expert Systems with Applications: An International Journal*, 34(1), 469–481, 2008.
- W.J. Clancey, P. Sachs, M. Sierhuis, and R. van Hoof. Brahms: Simulating Practice for Work Systems Design. *International Journal of Human-Computer Studies*, 1998.
- W.J. Clancey and M. Sierhuis. Multi-Agent Work Practice Simulation: Progress and Challenges. *NATO Future Modeling and Simulation Conference*, 2000.
- W.J. Clancey. Simulating Activities: Relating Motives, Deliberation, and Attentive Coordination. *Cognitive Systems Research, Special Issue on Situated and Embodied Cognition*, 3(3), 471–499, 2002.
- W.J. Clancey, M. Sierhuis, C. Seah, C. Buckley, F. Reynolds, T. Hall, and M. Scott. Multi-Agent Simulation to Implementation: A Practical Engineering Methodology for Designing Space Flight Operations. To appear in Artikis A., O'Hare G., Stathis K., and Vouros G. (eds), *Engineering Societies in the Agents' World VIII*, Lecture Notes in Computer Science Series, Volume 4870, Heidelberg Germany: Springer, 2008.
- P. Davidsson. Multi-Agent Based Simulation: Beyond Social Simulation, in Moss S. and Davidsson P. (eds), *Multi-Agent Based Simulation, LNAI 1979*, pp. 97–107, 2000.
- P. Davidsson, S.J. Johansson, and M. Svahnberg. Characterization and Evaluation of Multi-Agent System Architectural Styles, in *Proceedings of the Fourth International Workshop on Software Engineering for Large-Scale Multi Agent Systems (SELMA'05)*, pp. 497–503, 2005.
- P. Davidsson, J. Holmgren, H. Kyhlback, D. Mengistu, and M. Persson. Applications of Agent Based Simulation, in Antunes L.

- and Takadama K. (eds), *Multi-Agent Based Simulation, LNAI 4442*, pp. 15–27, 2007.
- H. Deguchi, M. Kobayashi, and H. Lee. Agent Based Modeling of National Economy – Centralized Mode Analysis of Virtual Economy by GA Simulation, in *Proceedings of IEEE International Symposium on Computational Intelligence in Robotics and Automation*, pp. 1317–1322, 2003.
- P. Domingos and M. Richardson. Mining the Network Value of Customers, in *Proceedings of the Seventh International Conference on Knowledge Discovery and Data Mining*, pp. 57–66, 2001.
- J.M. Epstein and R. Axtell. *Growing Artificial Societies*, The Brookings Institute, 1996.
- T. Erl. *Service-oriented Architecture: Concepts, Technology, and Design*. Prentice Hall, 2005.
- FIPA Repository. <http://www.fipa.org/repository/aclspecs.html>, accessed on September 19, 2008.
- T. Finin, R. Fritzson, D. McKay, and R. McEntire. KQML as an Agent Communication Language. *Software Agents*, 1997.
- R. Goldstone, A. Jones, and M.E. Roberts. Group Path Formation. *IEEE Transactions on Systems, Man, and Cybernetics, Part A: Systems and Humans*, 36(3), 611–620, 2006.
- R. Hirsh, J. Graham, K. Tyree, M. Sierhuis, and W. Clancey. Intelligence for Human-Robotic Planetary Surface Robots, in Howard AM and Tunstel EW (eds), *Intelligence for Space Robotics*, pp. 261–279, 2006.
- M. Hoggendoorn. Adaptation of Organizational Models for Multi-Agent Systems Based on Max Flow Networks. *Proceeding of International Joint Conference on Artificial Intelligence (IJCAI'07)*, pp. 1321–1326, 2007.
- K. Hwang and Z.W. Xu. *Scalable Parallel Computing: Technology, Architecture, Programming*, McGraw-Hill, 1998.
- N.R. Jennings. On agent-based software engineering. *Artificial Intelligence*, 117(2), 277–296, 2000.
- M. Kafeel and I. Ahmad. Optimal Task Assignment in Heterogeneous Distributed Computing Systems. *IEEE Concurrency*, 6(3), 42–52, 1998.
- R. Khosla, N. Ichalkaranje, and L. Jain. *Design of Intelligent Multi-Agent Systems: Human-Centredness, Architectures, Learning and Adaptation*, Springer, 2005.
- L.S. Mason. <http://cs.gmu.edu/~eclab/projects/mason/publications/SwarmFest04.pdf>, accessed on September 9, 2008.
- M.C. Macal and M.J. North. Tutorial on Agent-Based Modeling and Simulation Part 2: How to Model with Agents, in *Proceedings of the Winter Simulation Conference*, pp. 73–83, 2006.
- H. Mostafa and R. Bahgat. The Agent Visualization System: A Graphical and Textual Representation for Multi-Agent Systems. *Information Visualization*, 4(2), 83–94, 2005.
- N. Minar, The SWARM Simulation System: A Toolkit for Building Multi-Agent Simulations, Swarm Development Group Wiki. http://www.swarm.org/index.php/main_page, accessed on September 6, 2008.
- N. Netten and M. van Someren. Identifying Segments for Routing Emergency Response Dialogues, in *Proceedings of the Fifth International ISCRAM Conference*, 2008.
- A. Newell. *Unified Theories of Cognition*, Harvard University Press, 1990.
- T.I. Ören, S.K. Numrich, A.M. Uhrmacher, L.F. Wilson, and E. Gelenbe. Agent-Directed Simulation: Challenges to Meet Defense and Civilian Requirements, in *Proceedings of the 2000 Winter Simulation Conference*, pp. 1757–1762, 2000.
- X. Pan, C.S. Han, K. Dauber, and K.H. Law. A Multi-Agent Based Framework for the Simulation of Human and Social Behaviors during Emergency Evacuations. *AI & Society*, 22(2), 113–132, 2007.
- H.V.D. Parunak, R. Savit, and R.L. Riolo. Agent-Based Modeling vs. Equation-Based Modeling: A Case Study and Users' Guide, in Sichman J.S., Conte R., and Gilbert N. (eds), *Multi-Agent Systems and Agent-Based Simulation*, Springer, 1998.
- D. Perry and A. Wolf. Foundations for the Study of Software Architectures. *ACM SIGSOFT Software Engineering*, 17, 40–52, 1992.
- Repast Symphony. <http://repast.sourceforge.net/docs/tutorial/SIM/index.html>, accessed on September 16, 2008.
- S.F. Railsback, S.L. Lytinen, and S.K. Jackson. Agent-based Simulation Platforms: Re-

- view and Development Recommendations. *Simulation*, (82)9, 609–623, 2006.
- S. Russell and P. Novig. *Artificial Intelligence: A Modern Approach*, Prentice Hall, second edition, 2002.
- D. Sallach and C.M. Macal. The Simulation of Social Agents: An Introduction. *Special Issue of Social Science Computer Review*, 19(3), 245–248, 2001.
- T.C. Schelling. Models of segregation. *The American Economic Review*, 59(2), 488–493, 1969.
- C. Schreiber and K. Carley. Going Beyond the Data: Empirical Validation Leading to Grounded Theory, Computational and Mathematical Organization Theory, 10, 155–164, 2004. C. Schreiber and K. Carley. Going Beyond the Data Empirical Validation Leading to Grounded Theory, Computational and Mathematical Organization Theory, 10, 155–164, 2004.
- J. Searle. *Speech Acts*, Cambridge University Press, 1969.
- T. Serzykko, M. Ganzha, M. Gawinecki, P. Kobzdej, and M. Paprzycki. Introducing Commodity Flow to an Agent-Based Model E-commerce System, in *Proceedings of the 2007 IEEE/WIC/ACM International Conference on Intelligent Agent Technology*, pp. 294–298, 2007.
- M. Shaw and D. Garlan. *Software Architecture – Perspectives on an Emerging Discipline*, Prentice Hall, 1996.
- Y. Shoham. *An Overview of Agent-Oriented Programming*, Software Agents, MIT Press, pp. 271–290, 1997.
- M. Sierhuis, W. J. Clancey, R. van Hoof, and R. de Hoog. Modeling and Simulating Human Activity, in Freed M (eds). *AAAI Fall Symposium on Simulating Human Agents*, pp. 100–110, 2000.
- M. Sierhuis. Modeling and Simulating Work Practice; Brahms: A Multi-Agent Modeling and Simulation Language for Work System Analysis and Design. *Ph.D. dissertation*, SIKS Dissertation Series No. 2001–10, University of Amsterdam, 2001.
- M. Sierhuis, W.J. Clancey, and M. Sims. Multi-Agent Modeling and Simulation in Human-Robot Mission Operations Work System Design, in *Proceedings of the 35th Annual Hawaii International Conference on System Sciences (HICSS'02)*, 2002.
- M. Sierhuis and W. Clancey. Modeling and Simulating Work Practice: A Method for Work Systems Design. *IEEE Intelligent Systems*, 17(5), 2002.
- M. Sierhuis, W.J. Clancey, C. Seah, A. Acquisti, D. Bushnell, B. Damer, N. Dorighi, L. Edwards, L. Faithorn, L. Flueckiger, R.V. Hoof, D. Lees, A. Nandkumar, C. Neukom, M. Scott, M. Sims, R. Wales, S.Y. Wang, J. Wood, and B. Zhang. Agent-Based Mission Modeling and Simulation. *Spring Simulation Multi-conference (SpringSim 2006)*, Huntsville, AL, 2006.
- M. Sierhuis, T.E. Diegeman, C. Seah, V. Shalin, W.J. Clancey, and A.M. Selvin. Agent-Based Simulation of Shuttle Mission Operations, in *Proceedings of the 2007 spring simulation multi-conference (SpringSim'07)*, pp. 53–60, 2007.
- L. Suchman. *Plans and Situated Actions: The Problem of Human–Machine Communication*, Cambridge University Press, 1987.
- K.P. Sycara. Multi-Agent Systems. *AI Magazine*, pp. 79–92, Summer, 1998.
- S.R. Wolfe, F.Y. Enomoto, P.A. Jarvis, and M. Sierhuis. Comparing Route Selection Strategies in Collaborative Traffic Flow Management, in *Proceedings of the 2007 IEEE/WIC/ACM International Conference on Intelligent Agent Technology*, pp. 59–62, 2007.
- S.R. Wolfe, M. Sierhuis, and P.A. Jarvis. To BDI, or not to BDI: Design Choices in An Agent-Based Traffic Flow Management Simulation, in *Proceedings of the 2008 Spring Simulation Multi-Conference*, pp. 63–70, 2008.
- M. Wooldridge. *An Introduction to Multi-Agent Systems*, Wiley, New Jersey, NJ, 2002.
- L. Yilmaz. Validation and Verification of Social Processes within Agent-Based Computational Organization Models. *Computational and Mathematical Organization Theory*, 12, 283–312, 2006.

11

Simulation for Systems Engineering

Joachim Fuchs

11.1

Introduction

The role of simulation in the support of the systems engineering process requires a good understanding of that process. Especially complex dynamic systems benefit from the use of simulation in the engineering process. The support to this process can fundamentally be split into two distinct phases. In the first phase the main support is to predict the behavior and/or performance of a system before it is built, and in the finishing phase to support the verification of the product, especially in cases where the operational environment is not accessible for testing (e. g., for space-based systems). This article will highlight the relevant steps of the systems engineering process, identify the different steps where simulation can play a role, describe possible facilities, and discuss some relevant issues related to their implementation. A discussion on process management, responsibilities, and its impact on the testing and verification effort concludes the chapter.

11.2

The Systems Engineering Process

In order to understand the possible contribution of simulation to the systems engineering process, it is important to understand the main processes and its steps, as well as the functions a systems engineer has in this process. There are many different definitions of systems engineering, and in what follows an additional view might be added, although it is inspired by the process used in the European space industry (ECSS-E-10, 2004). However, the main common characteristics of all definitions are elements such as multidisciplinary, end-to-end, teamwork, or completeness. Keywords are as well architecture and interfaces. The importance of systems engineering grows with the complexity of a system. The fundamental steps to be covered are the following:

1. Requirements analysis;
2. Specification;
3. Design;
4. Analysis;
5. Performance estimation/design verification;
6. Development/production;
7. Assembly, integration, test/verification, and validation;
8. Operation/training/maintenance.

These steps need to be done in an iterative manner in order to find an optimal solution for the system to be built. The complexity of a system is determining to a large extent the need for formalism and procedures in this iterative process: a simple system that can be handled by a small team of engineers can cover these steps in close collaboration without too much formalism, and the interaction is limited to a few interfaces. The more interfaces between elements and/or domains are needed, the more importance is the integration work, and the more important it becomes to formalize the steps taken in procedures and communications between the partners.

Another important issue is the programmatic context in which a system is built (INCOSE, 2007). The objectives for systems engineering can be quite different if they address a single unit development, a small series, or a mass product. The needs for verification of design and performance as well as the final product are quite different in these cases. They are very often linked to characteristics such as “performance driven” or “production driven”.

A further element of distinction is the use context/usage of a system, which can range from research to operational uses. This dimension normally has a direct impact on the external interfaces of a system. In the former case it is likely that during use there will be changes expected from the system – the end user (i. e., scientist) will want to interact with system elements on a different level compared to an operational system, where the implementation should be irrelevant and hidden from the end user. The steps identified above can fundamentally be mapped into the preproduction phase, the production/development phase, and the postproduction or use/operational phase of a system. In each phase simulation plays a different role, as explained below.

11.3

Modeling and Simulation Support

At the risk of stating the obvious, it is important to stress that simulation should never be an end in itself. It should be a derivation of needs on the engineering level and provide answers to specific questions. In addition, the simulations discussed further address system issues and by their nature are not meant to answer domain-specific questions (such as: what is the mechanical load on bolt 15 in assembly XYV). The art of system engineers is to formulate relevant questions according

to the process and work together with a simulation expert to translate them into requirements for a simulation system.

Based on the steps described above, some typical questions are outlined below to illustrate the starting point for the definition of a required simulation system:

1. *Requirements analysis*: The system engineer needs to understand what the user wants to achieve with a system. Simulation can help to formalize and quantify these needs by providing a simulated function and/or product of the system as a basis for discussion. The focus is on the what and not the how.
2. *(System) Specification*: In this phase the user's needs are translated and quantified to engineering parameters (830-1998, 2004). Mainly the external interfaces of a system are formalized, and simulation can support this process by providing a formal description of these interfaces, allowing a dynamic verification at a later stage of system design.
3. *Design*: This is the crucial phase in the process, setting the boundary conditions of all subsequent elements. Important questions include: What is the best distribution/allocation of the required performances in my system? Based on an initial architecture, different scenarios can be simulated to optimize the breakdown. Where are my design drivers? In conjunction with the question above, simulation can help to identify the requirements that have a sizing effect on major system elements.
4. *Analysis*: In iteration with the previous step, individual parts are analyzed and further detailed. In general, critical elements (affected by system drivers) are simulated in further detail, providing insight into element performance, allowing further trade-offs and optimization of allocation of performance parameters to system elements. In this phase, interaction between system-level and domain-specific analysis is important, and the level of abstraction between these disciplines needs to be considered carefully to ensure correct abstraction for the different levels. Simulation is the communication tool between engineering disciplines.
5. *Performance estimation and design verification*: During the design process, system performance needs to be estimated to ensure that the envisaged design is fulfilling the user's requirements. Simulation can provide a virtual representation of the system in form and function and can therefore serve in this phase as a communication tool between the system engineer/designer and the user. It is crucial in this phase to ensure that the system representation by the simulation (as modeled by the simulation engineer) is consistent with the system design at any time. The underlying representation of the design and related data requires the necessary attention (Rainey, 2004).
6. *Development and production*: In the process of production, elements will be produced in isolation, due to distributed production facilities, out of sync due to planning inconsistencies, and developments at different level of maturity due to necessary elementary research and development. Simulation can provide the environment for the verification of elements. The interfaces

of single developments can be verified against the simulated interfaces of the remainder of the system.

7. *Assembly, integration, test, verification, and validation:* When putting the system together, the simulated environment of the system context is replaced step by step by the real elements. If the system is to be operated in an environment that is not readily available (such as, e.g., a space system), this environment remains simulated to provide the appropriate stimuli for the integrated system. The main effort has to be put into the proper interfacing of the simulated environment to the integrated system for proper testing of the system.
8. *Operation, training, and maintenance:* When a complete system requires the intervention of an operator, (s)he needs to be trained on the behavior of the system. To that end, simulation can provide the dynamics of the system in all conditions, including nonnominal behavior requiring specific procedures to recover. Training simulators enable the exploration of the full state space of a system without causing potential damage to the system to be operated. The challenge is to ensure that all possible reactions of the system on user/operator level, including the failure cases, are modelled. If maintenance is possible for a system (as for most systems containing software elements), it is important to enable the appropriate interaction with the system (visibility within the system). Testing of maintenance actions needs to be done before applying any modifications (patches) to the final system, in order to ensure that the system will not be damaged by the operations.

The art of the simulation engineer encompasses the foresight to anticipate future questions and to make sure that the simulation system built allows an evolution to address these questions and needs at a different time in the development process of a system. On the other hand, it is important not to overengineer any development to remain efficient.

11.4 **Facilities**

In order to fulfill the needs outlined above, the requirements of the simulation facilities vary. One of the challenges in an industrial setup is the consistency of these facilities along the different steps of the project. Whereas in small projects it can often be ensured that the development activities for simulators are coordinated by the same partners and in close collaboration with the systems engineering team, this is not guaranteed in the context of a large project with distributed partners. A clear definition of responsibilities and standards helps in that case to increase the efficiency of the development.

A clear separation of a framework/infrastructure and the modeling part is a precondition to achieving this efficiency. An appropriate interface definition needs to be agreed to allow the modeling effort to be reused in different contexts. This is par-

ticularly important if industrial processes make use of proprietary infrastructure that is not shared by all partners. Concerning the services/functionalities required by different infrastructures, they differ mainly w.r.t. the phase in which they are used. In the early phases a pure functional simulation may just require the management of time, without the notion of linking the simulated time to any real or wall-clock time. This will change in later phases, when links are established to the real world. This link can be in terms of equipment that is integrated in a simulation environment, or in terms of a link to physical time through interaction with a user. In both cases synchronization and response time need to be set in a hard real-time way, guaranteeing response times and latency on a time scale compatible with the equipment/user processes. In the former case (hardware in the loop) this is usually characterized by parameters such as guaranteed response time, latency, and jitter. In the latter case (person in the loop) this becomes much more linked to the channels that are used for communication. If just numerical/symbolic in-and output devices are used (such as synoptic displays, buttons, and text entries in fields), the main criticality lies on the side of the interface reaction to interaction and is less dependent on the process time itself. If the interaction is, however, based on graphical and haptic principles (usually referred to as virtual reality VR), the requirements for the simulation is usually the response through the simulation loop.

A further difference is the need to simulate either discrete or continuous processes. Whereas in the first case it is natural to refer to a digital representation, in the case of continuous (physical) processes it is normally required to consider the effects of discretization, sampling, and the problem of handling events/discontinuities. One specific case is the need to provide a simulated environment for the execution of software. Software plays an important role in modern systems, since a lot of functionality is provided by a software implementation, and there is the need to provide the capability of digital accuracy concerning the execution of software code (especially if it is an embedded application), and on the other hand the possibility to control this execution without effecting the environment. For that situation the term *real time* has a specific meaning because a simulated environment for a software can be stopped (compared to the physical time), but for the execution of the software no difference from a real processing environment can be seen. Emulators (SW or HW based) are used to achieve this.

11.5

An Industrial Use Case: Space Systems

In the context of (European) space projects efforts are ongoing to increase the efficiency of the development of simulators (ETM-10-21, 2007). The following table gives an impression of the mapping of phases and SE objectives to particular facilities that can support these objectives:

	Prephase A	Phase A	Phase B	Phase C	Phase D	Phase E	Phase F
Feasibility and Performance Analysis/Trade-Offs	Concurrent Design Activities						
Requirements Specification	Concurrent Design Activities System & Mission Analysis						
Design and performance Verification		System Interfaces and End-to-End Design Trade-off					
Subsystem & Payload V&V			Interfaces and End-to-End	AIV OBSW			
Spacecraft Qual. and Acceptance				Virtual AIV	AIV SVTs		
Ground Segment Qual. and Acceptance				MCS Testing	Operations Procedure Validation SVTs		
System Qual. and Acceptance					AIV SVTs	System Maintenance (e.g. S/W)	
Training & Operations					Mission Control Team Training	Ongoing Mission Control Team Training OBSW Patch and Ops Procedure Validation Anomaly investigation and resolution	Investigation of Disposal options

Fig. 11.1 Life-cycle and SE objectives in the space domain.

Although the objectives differ considerably, as shown in Figure 11.1, some development effort between the facilities may be reused. To ensure this, the following steps are required:

- Definition of a top-level architecture for simulators and facilities, identifying the major building blocks required for the different functions.
- Specification of a detailed architecture with clear identification of the interfaces, their semantics, and the functions of their modules.
- On the implementation level, a clear identification of the syntax used to implement simulation models and infrastructure.

For the context outlined above, a high-level generic architecture has been described (Figure 11.2). This architecture has (in principle) all the elements which can occur in the different simulation contexts, but not all will be instantiated at any given moment, and not all will be the same in all instances.

The virtual system model comprises the simulation infrastructure and the models of the space system, including the ground segment and space environment. For different systems, the ground model can be replaced by any other control infrastructure, and the space environment obviously by the relevant environment of the system. It is important to note the clear distinction between the simulation infrastructure and the models integrated in this infrastructure. It supports also external interfaces to the equipment under test and external monitoring and control facilities. The facility monitoring and control may also monitor and control the virtual

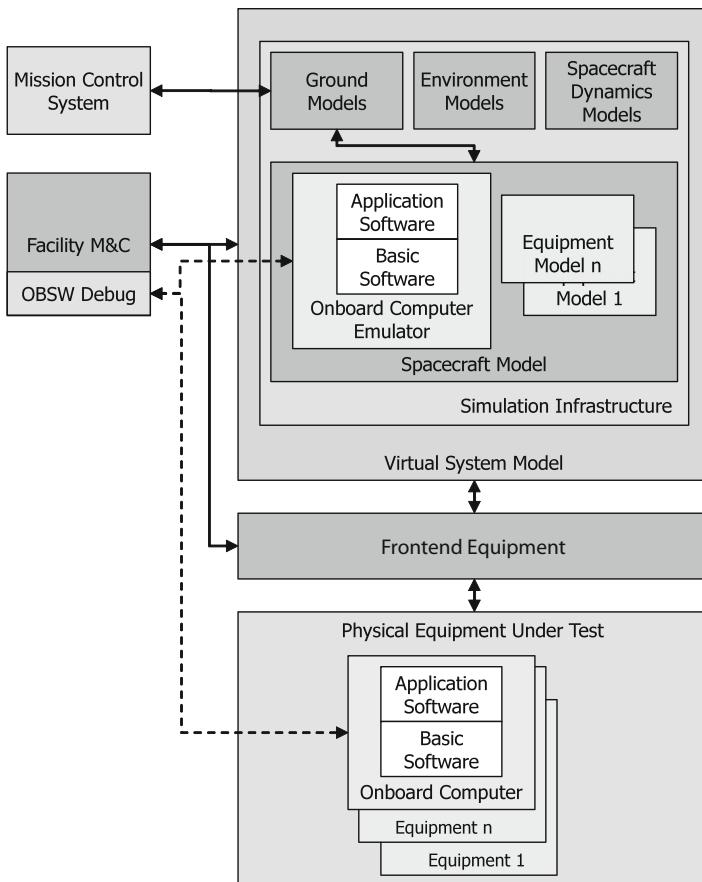


Fig. 11.2 Generic architecture for simulation facilities with embedded software.

system model as well as having normal telemetry and telecommanding functions. In what follows, the instances of this architecture in the space context are described.

11.5.1

Simulators for Analysis and Design

During the early phases of the development process identified above, basically three different classes of simulators are developed, supporting the analysis and trade-off aspects of the design phase. These classes are, on the one hand, domain or subsystem specific (such as data handling, control system, power, thermal, and structures) and only of limited interest to system-level considerations in their full complexity and specificity. However, the resulting designs from these detailed analyses are then integrated into a system simulator to address issues critical to the complete (integrated) system/mission (e.g., landing on a planetary surface). These

simulations are typically used to support the mission analysis and the definition of an operational concept.

In addition to these analyses, it is crucial to get a sound analysis and prediction of the performance at the overall mission level. Dedicated simulators focusing on the mission product (instrumentation focused) help to support the specification and early consolidation of science and user requirements. The further evolution of these simulators enables the proper preparation of processing and service definitions for a particular mission.

Due to the specific character of such developments, no uniform methodology has yet been defined. In the future, however, it is expected that a methodology for system simulation in the early phases will be established in order to properly support system engineering trade-offs.

The earliest of these simulators is the *system concept simulator* (SCS), typically running non-real-time mathematical models that support the specific needs of the engineering disciplines and allowing the rapid evaluation of system design and operational concepts. Addressing the mission aspects are the mission performance simulators, allowing the establishment and verification of the overall performance of the baseline mission from the user point of view, including adequate payload models.

With a baseline system being consolidated and verified on the system level by the aforementioned simulators, critical elements of this baseline (such as data handling, control algorithms) may need to be prototyped and functionally validated in a system context. The simulator supporting this is the *functional engineering simulator* (FES). It is recommended to maximize the reuse of the mathematical models (or parts thereof) between this and the earlier system-level simulators to prepare the basis for building the real-time simulators that are exploited in the subsequent design/test phases.

In a further step and representing the first step toward the actual implementation, it might be necessary to actually build/breadboard some equipment/subsystem and validate the performance of the design again in a system context. A *functional validation testbench* (FVT) is used for this purpose, and the scope of this simulation is again the complete system, with a focus on the identified critical and prototyped/breadboarded elements. As outlined above, software plays a major role in the system management of a spacecraft and therefore occupies a specific position in the overall system verification and validation approach. The onboard software interacts with the spacecraft system evolving in space under the control of operations and environmental inputs and constraints. The onboard software is itself commanded from ground via a TM/TC interface. The validation of the onboard software needs to be performed in a context that is representative of the spacecraft system in space and with ground interfaces. This is done in a *software validation facility* (SVF).

In the SVF, this context is simulated and contains a fully functional and performance representative simulation model of the spacecraft hardware and its dynamic behavior in space (the dynamics/kinematics models). Conceptually the SVF is a functional validation facility dedicated to the software as the item under test. In

this respect this facility bridges the gap to the system qualification and acceptance facilities since there is no difference between the SW in the development context and in the target context.

11.5.2

Facility for Spacecraft Qualification and Acceptance

During the spacecraft qualification and acceptance a simulator replaces missing equipment and also simulates the environment and the dynamics of a space vehicle, that is, calculates, for example, position and attitude, or provides simulated input to other sensors as required. This allows real-time, closed-loop tests in which the response of the spacecraft to commanding is taken into account, including the response to the simulated environmental stimuli to which the equipment is subjected. In such a spacecraft AIV facility the simulation of missing equipment can be implemented as part of this central simulator function or can be embedded in the FEE/SCOE components. In either case it must be possible to replace missing equipment by simulated equipment models. The simulator is therefore an integral part in most configurations of the spacecraft AIV facility.

11.5.3

Facility for Ground System Qualification and Testing and Operations

A space segment simulator (real-time behavioral model of space segment, ground stations, and interfaces to the mission control system) is used for validation of the ground segment and training of ground operators. The simulation models of the *Operations simulator* shares many characteristics with the models used in the SVF and the spacecraft AIV facility, and the operations simulator should, where practicable, be derived from them. These simulators are routinely used in operations to validate flight procedures and to assist the analysis of telemetry and potential troubleshooting.

In order to ensure the technical verification and validation of the ground segment, an extensive verification of its compatibility with the space segment is required. This is supported by using a space segment simulator including realistic software models of the space segment elements (e.g., emulators for complex software elements) and in some cases real spacecraft components (hybrid simulator). The ground station(s) and its (their) interfaces to the MCS are usually part of the simulation to allow closed-loop tests to be run from the MCS.

11.6 Outlook

It is recognized as a good practice to procure simulation products in a consistent manner across a project. In this way, commonalities between the different areas

can be reinforced and may be exploited to reduce costs if the different requirements can be adequately met.

Modeling and simulation capabilities are expected to grow in the future as organizations embrace it further in their processes, build experience, and build up a reusable base of models and infrastructure. This evolution is expected to include:

- Linkage with systems engineering data (candidates could be SysML for systems engineering data, and STEP data to configure the simulation and allow simulation results to feed back into the systems engineering process);
- Better integration within the systems engineering process (e.g., have a formal notion of a virtual system);
- Modeling used to predict the behavior of the system during the design process (e.g., due to failures) to allow the design to be modified accordingly;
- Development of models by subcontractors and delivery to primes become mandatory the use of established model exchange standards (e.g., Simulation Model Portability and common architecture)

The approach is to develop a digital or virtual model of the system already in early phases to support the specification and definition activities, and to evolve it to support design, testing, qualification, and operations phases. In order for this approach to provide its full potential, the use of digital (virtual) models of the system has to be an integral part of the specification, design, and verification process, in full coherence with the development of hardware models, thereby optimizing cost and schedule.

This innovative approach has the potential to increase the cost-effectiveness and quality of the development process because it allows for the optimization of the design at the system level and to increase the coherence between analysis, design, testing, and operations. It allows one to close some required iteration loops earlier in the life cycle and therefore has the potential to reduce the effort in integration and testing. In particular, improvements are expected in the following areas:

- Requirements management and verification,
- Design trade-offs at the system level,
- Analysis of system operability issues,
- Assessment of engineering margins,
- Coherence between analysis and testing,
- Preparation and execution of AIV,
- Transition from AIV to operations.

Model-based data sharing will be a major contribution to ensure the coherence of the different facilities in the life cycle. A systematic approach to formalizing all engineering (and management) data in a project will enable an automatic configuration of simulators with up-to-date data, representing at any time a correct and consistent set of engineering information, where any simulation results actually match the system analyzed and/or tested. This will also allow a closer interaction of different domains. It is, however, important to ensure a principle of single data single source, in order to avoid conflicts in design iterations.

In a further step, systems engineering will require more formal support than just simulation. The business process needs to be modeled, and the complex system of systems will need additional layers of interaction, with a large stress on nonengineering aspects and programmatic boundaries.

11.7 Conclusions

The support of simulation for the systems engineering process has been highlighted in an approach driven by practical experience rather than academic considerations. It is illustrated by an outline of facilities used in the space industry, highlighting the critical areas in the simulator domain. The difference between system simulation and domain/detailed simulation needs to be understood to ensure that there is no confusion of simulation objectives. For system simulation the complex interaction of all system elements needs to be captured, which might be at the expense of details in some cases. Reuse drives many concepts in the industrial context, and the need for data exchange models is highlighted, enabling further consolidation of system-level interactions, in particular between different disciplines.

References

- IEEE 830-1998 (2004) *IEEE Guide for Software Requirements Specification, IEEE 830-1998*.
- ECSS-E-10 (2004) *ECSS-E-10 Part 1B, System Engineering: Requirements and Process*.
- ECSS ETM-10-21 (2007) *ECSS ETM-10-21, Technical Memorandum on System Modeling and Simulation*.
- INCOSE (2007) *Systems Engineering Handbook*, INCOSE.
- Rainey, L.B. (2004) *Space Modeling and Simulation*, Aerospace Press.

12

Agent-directed Simulation for Systems Engineering

Philip S. Barry, Matthew T.K. Koehler, and Brian F. Tivnan

12.1

Introduction

The development of systems of systems (SoS) is becoming commonplace; the government and private industry are increasingly faced with the problem of investing large amounts of money in SoS that cannot be fully specified in a requirements document, and cannot be fully tested or in many cases even prototyped. Compounding this problem is that SoS are not simply very complicated systems like a race car or a communications satellite. While complicated systems have many parts that interact with each other in nontrivial ways, we can describe the interaction using well-understood laws of mechanics and physics. SoS, on the other hand, present a constantly changing topology, evolving over time and space. Furthermore, an important, if not the most important, component is the human element; who interacts with whom, as well as the technology to provide new systemic capabilities are often impossible to define *a priori*. This differs from the classical view of systems, where the human element was considered outside the system, and then only as a user.

With this expansion of scope has come a recognition that new approaches are needed (e.g., Maier (1998); Carlock and Fenton (2001); Sage and Cuppan (2001); Kriegel (1999)). There has been good progress in developing paradigms that provide relatively static descriptions of the SoS, hierarchically decomposed views of the enterprise with robust descriptions of the interfaces. While providing a valuable base for analysis, these approaches do not provide a clear path for gaining insight into the dynamic and evolutionary behavior of SoS.

To engineer a SoS we propose that additional techniques are required. The sheer number of components in fielded SoS and their associated complex behaviors argues strongly against using closed forms of analysis. However, systems engineers are often required to advise in areas such as portfolio management that require a quantitative understanding of the possibility space of complex interactions. We suggest the employment of agent-directed simulation (ADS) alongside rigorous, proven systems engineering (SE) techniques and enterprise architecture development to gain insight into the specification, design, and evolution of SoS. Wegner

(1997) and Wegner and Goldin (1998) showed that, as powerful as closed form algorithmic analysis is, it essentially represents the system as a Turing machine, whereas allowing the algorithms to interact (e.g., an interaction machine) is a far more powerful analytic representation. Simon (1999) argued that complex systems can be meaningfully represented as collections of subsystems, hierarchies of nearly decomposable systems. Within these collections the state of each subsystem through time is only weakly influenced by the other subsystems. The ability to represent these complex systems (or SoS) in this way makes SE of them possible and creating ADSs of them meaningful and useful. Csete and Doyle (2002) stresses not only the importance of modules in complex SoS but also the criticality of common interfaces between the components. These interfaces are critical for large collections of systems to function properly. Schelling (1978) argued that the interaction of subcomponents of a system, without meaningful centralized control, can have enormous impact on the behavior and performance of the system as a whole.

Moreover, Buss et al. (1991) proved that under ideal conditions a model of SoS as a collection of homogeneous automata with a global control rule that is independent of the states of any of the automata the system will be predictable (i.e., the future state of the system is computable) in polynomial time; if, however, the global control rule is not independent of the states of the automata, then the prediction of the system becomes PSPACE-complete. This means that you can do no better to understand the future states of the system than to simulate it. More recently, Epstein and Hammond (2002) have argued, and demonstrated, that it can be far more insightful to examine the dynamics of a system as it moves through time than to solve for the system's equilibrium. It is trivial to specify a system that may not be able to obtain its equilibrium from plausible initial conditions or that it may take longer than the universe has been in existence for the system to achieve an equilibrium state. This being the case, simulations can prove far more useful from a SE perspective to gain insight into the actual, or meaningful, performance of the system in question. Finally, Epstein and Axtell (1996) and Epstein (2002) argue for a generative nature to our understanding of social systems – if we didn't grow it, we didn't explain it. They argue that understanding the emergent properties of a SoS (the macrodynamics) comes from specifying the components and then allowing the system to move forward in time. In this way one generates a sufficiency theorem – this specification with this set of input is sufficient to generate this output. Again, this argues that ADS for SoS engineering is not only useful, but may be one of the only ways to meaningfully understand and design complex SoS.

The chapter will discuss how ADS can be used in conjunction with traditional SE techniques to provide the professional in this field with additional insights into the design and eventual performance of these SoS.

12.2

New Approaches Are Needed

The International Council on Systems Engineering (COS) defines SE as an interdisciplinary approach and means to enable the realization of successful systems. It focuses on defining customer needs and required functionality early in the development cycle, documenting requirements, then proceeding with design synthesis and system validation while considering the complete problem. COS (2008) has had a long and successful record in providing a quantitative approach to realizing systems that meet user requirements in a timely fashion. Key to this success is the fundamental assumption of a bounded system, where the boundaries of the system are well defined.

The Vee model (shown in Figure 12.1) is one of several canonical SE life cycles (Osborne et al., 2005). The first part of this Vee model develops progressively more detailed descriptions of the item to be built, beginning with a high level concept of operations and ending with a detailed design of individual subcomponents. The decomposition and consequent design stops after a complete understanding of the system to develop has been derived. At this point, individual subcomponents are implemented and are subsequently assembled and tested into larger and larger components until the aggregate system is complete.

We assert that this approach does not scale for SoS specification, design, and implementation. The reasons for this are traced to the differences between SoS and traditional systems. As described by Sage, these differences can be categorized into five fundamental areas (Sage, 2005). First, the SoS is composed of systems that are independent and useful in their own right. Second, there is managerial independence of the component systems. Third, there is geographic distribution of the component systems. Fourth, the SoS evolves; the configuration of the SoS is not frozen in time. Fifth, there is emergent behavior that is both expected and unexpected from the complex interaction between component systems.

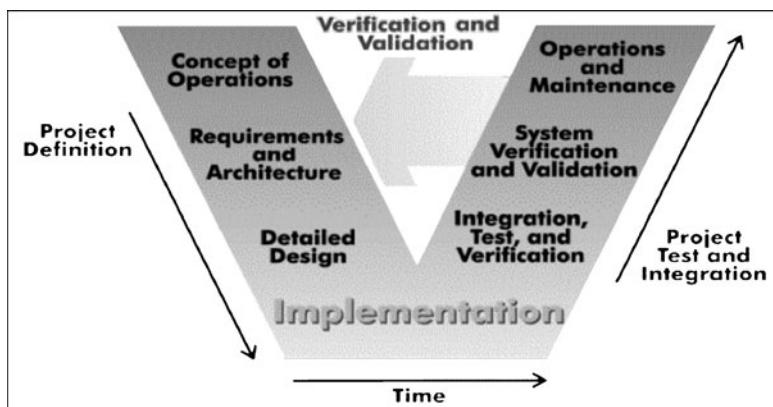


Fig. 12.1 The Vee model for system development.

As an illustration consider the 2008 failures in the international banking system. Among many other components, the international banking community contains individual banks that operate independently but also have a relation to one another in the global credit market. The banks are managed by a myriad of concerns and are distributed all across the world. The complexion of the banking system and its aggregate capabilities evolves, as evidenced by the easy credit environment in the early years of the 2000s and the drastic meltdown that occurred in 2008.

A model of the banking system that has served the community has been shown to be inadequate to explain the behaviors witnessed as well as to identify fixes. The 2008 meltdown can be traced in part to the rise of credit default swaps (CDSs), a derivative virtually unheard of in the 1990s (Editorial, 2005; van Duyn, 2008). The creation and proliferation of CDSs was an emergent behavior of human behavior interacting with the financial system.

In the midst of the turbulence it became clear that the previous models of the banking system were inadequate and that the SoS defied exhaustive behavioral specification. Our research has indicated that this phenomenon is typical for SoS and led us to the realization that ADS is an essential tool for the systems engineer. ADS can help throughout the SE process; as a thought tool for generating requirements, a prototyping and functional analysis tool, and even a visualization platform [see generally Axtell (2000)]. Further, ADS can allow one to represent various components of the SoS at whatever level of verisimilitude is required for the current task, including adaptive and goal-directed human behavior as well as to experiment with a number of operational contexts and complex interactions.

12.2.1

Employing ADS Through the Framework of Empirical Relevance

The progressive development and maturation of using an ADS in the SE process can be categorized as the migration of the ADS through Axtell's Framework of Empirical Relevance (FER) (Axtell, 2005). The FER relates a model's input and output data to the phenomena under study. There are four levels to FER. Level 0 is, essentially, a well-functioning program that is bug free. Level 0 simulations have qualitative correspondence at the agent level. This means that the agents behave in a manner that is logically consistent with the system being simulated. Level 1 is macro-level, qualitative correspondence to the dynamics of interest. At this level, the agent activity generates dynamics, as a whole, that relate to the actual phenomena. For example, a group of agents trading with each other may produce a clearing price for the artificial market. This clearing price might not relate to a real-world clearing price but one was found. Level 2 simulations have macro-level, quantitative correspondence with the real-world phenomena being modeled. These models produce the correct distributions within their output. For example, Axtell's model of firm size produces a power-law distribution that is the same as empirical data on the distribution of firm sizes in the USA (Axtell, 2001). The final level of this framework is level 3. At this level, the simulation not only has macro-level quantitative correspondence but also micro-level correspondence. In general, very few

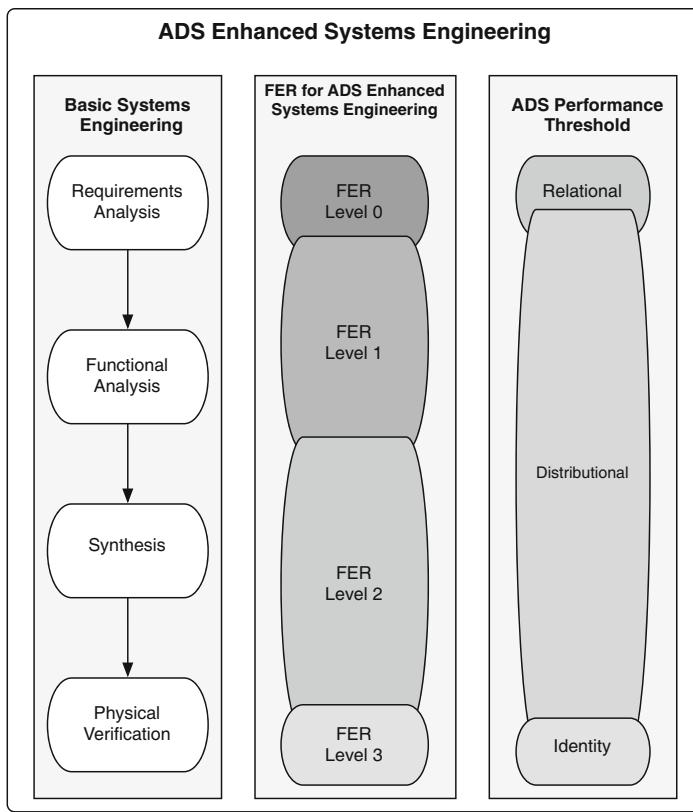


Fig. 12.2 Framework for empirical relevance.

ADSs achieve this level of empirical relevance. This is because it is difficult to specify such a simulation and even more difficult to obtain the empirical data at the necessary fidelity to conduct the comparative analyses.

Figure 12.2 depicts the relationship between the FER and the ADS used to enhance the basic SE process. As one begins to design the SoS and to prototype an ADS, one has two main concerns: (a) producing a model formulation that includes the most important features of the system and (b) implementing the formulation so that the simulation functions properly. As the requirements take shape, one could begin to ask more of the ADS, namely, that it behave appropriately at a distributional level, FER level 1. Depending upon the system and the data available, this may be as far up the FER scale as the ADS will be able to go. Depending upon the use of the ADS, this may be completely adequate and can still provide a great deal of insight into the system in question. This is especially true if the ADS is being used as a way to present (visualize) the SoS or as a way to begin to determine what system-level metrics can be used to evaluate and monitor the behavior and performance of the SoS (a critical step in evaluation and testing). As the SE process begins to create more detailed artifacts, the utility of the ADS will rise concomitant-

ly if it progresses up the FER scale. Specifically, if the ADS is to be used to create or evaluate requirements, produce functional evaluations, or design prototypes, then one should ensure FER level 2 for the ADS. Ideally, the ADS should reach FER level 3, but this is not always feasible for the reasons identified above.

The fact that an ADS is unlikely to reach FER level 3 has implications for verification, validation, and accreditation (VV&A) of both the ADS and the design produced by the ADS-enhanced SE process. Loosely speaking *verification* is determining whether or not you built the model correctly, *validation* is determining whether or not you built the correct model, and *accreditation* is determining if the model is good enough to use for its intended purpose (Hartley, 1997). The VV&A process largely grew up as a methodology for determining how well a simulation replicated physical phenomena, such as the performance of an airplane wing. This being the case, the simulation was built upon well-understood laws of physics and could be tested against experimental results. If the simulation was not accurate enough, more detail was added until the required level of correspondence with reality was reached. However, when the human behavioral aspects of a system of system are considered, the VV&A becomes very difficult. This does not mean that VV&A cannot be performed on an ADS; it means, simply, that the VV&A process is different.

12.2.2

Simulating Systems of Systems

As we move into a SoS context, we may be using a collection of simulations that are not at the same level on the FER scale. This is to be expected as the SoS is an evolving system, where many of the components are not at the same level of maturity and may not be fully developed or understood. Consider the illustration in Figure 12.3. Below the dashed line we see the familiar V-model for system development. As discussed above, we can use FER as a framework for ensuring accurate system-level modeling. However, as shown in Figure 12.4, the problem becomes more complex in that for a SoS our development environment needs to consider the parallel construction of a myriad of systems, frequently in various stages of maturity. Further, as SoS engineers our focus is on developing aggregate capabilities for the SoS. Consequently, we need the ADS to provide insight and guidance for all phases of the development effort, from high-level architecting through integration and testing.

Using the framework outlined in Axtell et al. (1996) for docking ADSs to each other, we can build a framework for VV&A of ADS for SoS engineering. We contend that accreditation of ADS when used in this manner will be an ongoing process performed by the systems engineer(s) and the relevant stakeholders at particular points in time. What is necessary for an ADS to be accredited, or be deemed useful, will change as the SE process progresses. At first, the ADS may only need to have the major components of the SoS and their most basic interconnections and functionality to be useful. At this stage the ADS likely only needs to achieve level 0 or 1 on the FER scale, which implies the ADS and the formulation it implements (i.e., the current SE plan) must only be of relational equivalence to reality.

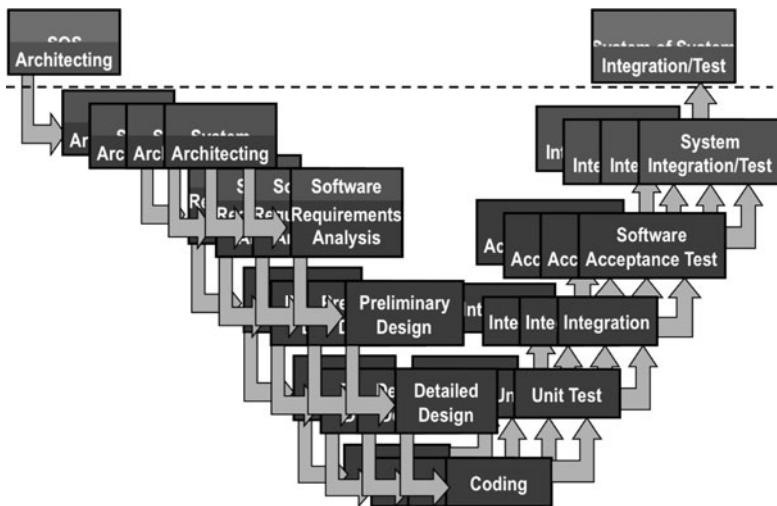


Fig. 12.3 SoS architecting and development.

As the plan becomes more specific and the SE process moves into functionality testing, then the level of detail necessary for the ADS to maintain its accreditation (while continuing to support the emergent properties of interest) must increase. This implies that the ADS must now also move higher in the FER framework, to level 1 or 2. This, in turn, implies that the ADS must now have distributional equivalence to the system in question and/or external data sources or results from other analytic tools.

During this process the requirements will likely be used to create a set of measures of effectiveness (MOEs). However, it is possible that the requirements will be stated as a series of component-level performance specifications. One could use ADS to create a system of these components, with their specified measures of performance (MOPs), and then run the simulation to generate the performance of the system and test ideas for the SoS-level MOEs. This ability, mapping MOP-level data into MOE-level metrics, is extremely important and very difficult to achieve by any other method, be it closed-form analysis; deterministic, physics-based simulation; or expert consultation.

Considering that SoS have so much randomness inherent in their construction, trying to provide a priori proof of the behavior of the SoS along any metric is unlikely to succeed. Clearly, in this environment gaining insight into a set of representative behaviors is the only reasonable goal. Given this randomness in the SoS and, by implication, in the ADS, one must use some sort of Monte Carlo method in conjunction with the ADS to truly understand it and the behavior of the system it represents. Each run of the ADS can be considered a deduction, and the aggregation of runs viewed as a collection of deductions that provides the observations necessary to make inductive inferences about the behavior of the SoS in question. Though not a formal proof, much can be done with inductive inference [see Hol-

land et al. (1986); Polya (1954); Shum (1994)], for example, Bayesian reasoning has proven useful (Gamerman and Lopes, 2006).

While understanding a complex technical system can be challenging, perhaps the most significant aspect of a SoS might be the human systems and the organizations that depend upon and interact with the technical systems. SE has historically spent less time modeling human systems than technical systems, although the rise of enterprise architecture (e.g., Maier et al. (2004); Zachman (1987)) and concerns about interoperability have recognized the importance of representing the human component in large systems (FEA, 2007; Dept. of Defense, 2007) and operational interoperability (Ford et al., 2007). However, these approaches are generally static models with limited simulation support.

12.3

Agent-Directed Simulation for the Systems Engineering of Human Complex Systems

In contrast to the static models described above, simulation often represents the method of choice for researchers (Carley and Svoboda, 1996; Epstein and Axtell, 1996; Levinthal, 1997; March, 1991; McKelvey, 1999a,b) to explore complex dynamics often found in human complex systems (HCSs) (Tivnan, 2005). HCSs are SoS that include active, human participants beyond simple roles of systems operator (e.g., a large, public venue including crowd and security personnel; a metropolitan area experiencing a pandemic; and a financial exchange including traders and regulators). The subsequent discussion of ADS for the design and engineering of HCSs begins with a general overview and then briefly describes some landmark, simulation-based studies that were influential to HCS research. This discussion continues with the call for a specific class of simulation for HCS (McKelvey, 1999b,c, 2002) followed by a brief description of some influential ADS in HCS research. This section concludes with a brief epistemological discussion of a model-centered science with ADS at its core.

While simulations applied to the study of HCSs first occurred as much as forty years ago (e.g., Cyert and March (1963)), only recently has it begun to generate a broader acceptance (Dooley, 2002). Not only special issues but entire journals are now dedicated to simulation and its application to the science of HCSs (e.g., Carley (1995), Lissack (1999), and Gilbert (1998)). This acceptance stems from two critical aspects of simulation research: (a) simulation allows researchers to explore the inherent complex dynamics of HCSs (Dooley, 2002; Dooley and de Ven, 1999), hence (b) simulation research allows for the conduct of experiments that would typically be impossible or impractical in the physical world (Gilbert and Troitzsch, 1999).

Stressing the value of simulations for theorizing, Weick (1995) and Axelrod (1995) argue that simulation offers a new vehicle for conducting scientific research that differs from induction (i.e., the discovery of patterns in empirical data) and deduction (i.e., specifying a set of axioms and proving consequences that can be derived from those assumptions). On the one hand, simulation research resembles deduction in that simulation starts with a set of assumptions. On the other hand,

the simulation generates data to be inductively analyzed. Axelrod (1995) refers to simulation research as thought experiments since the assumptions might seem simple but the results are often counterintuitive (i.e., the nonlinear, macro-level effects of interacting agents known as emergent properties).

Axelrod (1995) provides further support for simulation as an alternative to the rational actor/choice assumptions. Because the rational actor/choice assumption allows for deductive, closed-form analysis, researchers are willing to overlook the boundedly rational limitations of their actors (Simon, 1976). The primary alternative to the rational actor/choice assumption lies in some form of adaptive behavior. Due to the complex effects of social interactions, Axelrod (1995) asserts that ADS offers the only vehicle to study sets of actors who possess an adaptive capacity.

Recognizing these inherent strengths of simulation, James March proved to be one of the earliest pioneers of simulation in the study of HCSs and consequently has produced some of the most influential research in the field. As one of the first simulation-based studies in the field of HCS, Cyert and March (1963) advanced the organizational theories of Barnard (1938) and Simon (1955) to demonstrate that managers are rational in their pursuit of their personal goals while attempting to satisfy various stakeholders and avoid uncertainty. In Cohen et al. (1972), the Garbage Can model is used to demonstrate the path-dependent nature of organizational issues and structure as well as their effect on the performance of the HCS. Particularly relevant to HCS research, multilevel ADS linking individual learning and adaptation within an HCS supports the assertion that microcoevolutionary order within an HCS emerges in the context of macrocoevolutionary selection and competitive pressure (March, 1991; McKelvey, 1997). As two of the first studies in the field of HCS to use an ADS; Cohen et al. (1972) and March (1991) proved to be significant contributions to simulation-based research.

12.3.1

A Call for Agents in the Study of Human Complex Systems

With the growing acceptance of simulation in the design and engineering of HCS, several leading scholars have called for the formal use of ADS (e.g., Anderson (1997); Axelrod (1995); Dooley (2002); McKelvey (1999a,b,c)). As the primary tool of complexity theorists, ADS assume that agents behave in a stochastic, nonlinear manner and that agents possess a capacity to adapt over time.

This stochastic, nonlinear behavior of agents is consistent with the stochastic, idiosyncratic microstates of HCSs. That is, despite institutional influences (Zucker, 1988; Scott, 1995), strong forces remain to idiosyncratically steer both the behaviors of individuals and the conduct of aggregate processes (McKelvey, 1997). Among others, such forces might include unique organizational cultures, the unique set of suppliers and customers (i.e., organizations are each embedded within a unique social network), and the unique interaction network of different individuals each with his/her own personal history in different contexts. Therefore, agent activity in an ADS can offer an excellent representation of the adaptive and idiosyncratic behavior of an HCS and that of its human agents.

12.3.2

Noteworthy Agent-Directed Simulations in the Science of Human Complex Systems

Some of the most common uses of ADS perspective in the study of HCSs include cellular automata (Toffoli and Margolus, 1987), the NK model (Kauffman, 1993), simulated annealing (Aarts and Korst, 1989), and genetic algorithms (GAs) (Holland, 1995). Cellular automata consist of identical cells, usually arranged in a grid pattern, that interact locally according to some homogeneous rules [see Epstein and Axtell (1996)'s extension of the cellular automata model to explore the emergence of social networks, markets, and cultural differentiation]. The NK model uses the concept of rugged landscapes where ruggedness is determined by the number of components in a system, N , and the interdependence between those components, K [see Levinthal (1997); Levinthal and Warglien (1999); McKelvey (1999a,b); Rivkin (2000); Sorensen (1997) for applications of the NK model]. Analogous to the physical process of annealing, a solid, simulated annealing provides a heuristic solution to combinatorial, optimization problems [see the microcoevolution study linking individual learning and organizational adaptation (Carley and Svoboda, 1996)]. Analogous to evolutionary theory and natural selection, GAs consider the fitness of each agent in a population and breed the agents with the highest fitness [see seminal research on cooperation in the Prisoner's Dilemma (Axelrod, 1997)].

12.4

A Model-Centered Science of Human Complex Systems

Inspired by the merits of ADS to capture the stochastic idiosyncrasy often found in HCS, McKelvey (1999c, 2002) calls for a model-centered science of HCSs. That is, he supports the use of stylized models to further the study of HCSs by adhering to the semantic conception for scientific inquiry.

The semantic conception, first introduced by Beth (1961) and later advanced by Suppe (1989), is a normal science, postpositivist epistemology. It contends that scientific theories relate to models of idealized systems, not the complexity of real-world phenomena and not necessarily to self-evidently true, root axioms (McKelvey, 1999c). Fundamental to the semantic conception is a model-centered view of science that uses models as an intermediary between theory and phenomena to both represent theoretical relationships and predict fundamental, phenomenological behavior. This model-centered view provides a useful bridge between scientific realism and the use of computational experiments as a basis of truth tests of complexity-theory-rooted explanations (McKelvey, 1999c) in the study of HCSs.

12.5

An Infrastructure for the Engineering of Human Complex Systems

ADS typically represents a viable approach to experimenting with and testing a given SoS. With that in mind, we developed the Infrastructure for Complex Systems Engineering (ICE). With its model-centered core, the ICE is a collection of software tools, computational hardware, and methodologies that allow one to move from abstract thought experiments to operational testing and optimization of an HCS.

12.5.1

Components of the Infrastructure for Complex Systems Engineering

Consistent with solid SE practices, an application of the ICE starts with an assessment of what information about the HCS in question is currently known; this includes subject matter expert understanding of the components of the HCS and how they are interconnected, MOP-level data on the performance of the individual components (usually in isolation), and so on. Once enough information is gathered about the HCS in question, one moves to the prototyping stage. What constitutes an adequate prototype will be driven largely by what the HCS is and the questions to be asked about it. For example, if the prototype does not need to be overly large in scale (less than 10000 entities), we can effectively use tools such as NetLogo (Wilensky, 1989). However, there are times when even the prototype must be very large scale. In those cases, we move to scalable tools (Tatara et al., 1961). Eventually, as the prototype stabilizes we typically port the NetLogo simulation to Repast for deployment on a high-performance cluster computer. Though Repast is more tightly integrated with our cluster computer, NetLogo can be run on it also. Therefore, if scale or high performance is not a driving concern, we need not move away from our prototype. We use NetLogo and/or Repast to handle the representation of the HCS as a whole. Usually, however, there are components of the HCS that are of particular importance to the functioning and performance of the HCS. These components are handled specially in the ICE framework. These high-importance components are modeled separately at as high a resolution as possible. The family of simulations is now run together to represent the functioning of the system.

Use of a cluster computing system allows us to scale very large if necessary and also to run many replicates of the simulation to perform the aforementioned Monte Carlo analysis of the modeled HCS. The intelligent design of experiments is very important here as the parameter space associated with these models can be nearly infinite. As part of the cluster computing system we have an automated genetic optimization framework. Optimization over a complex space requires a reasonable degree of verisimilitude. Consequently, our methodology employs detailed physical models and vetted behavioral models where possible. We also seek to provide realistic physical models of the geometry where that level of detail is important to the question at hand.

Concurrent with the model development is the development of a preference model. Parallel model development is important in systems where emergence is

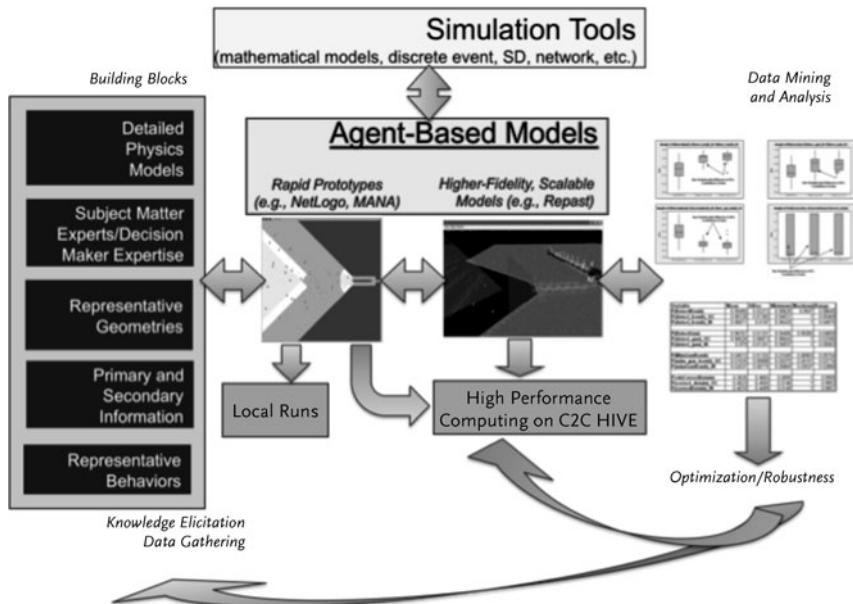


Fig. 12.4 The infrastructure for complex systems engineering.

a feature of study. This is the case because once the scalable model is developed, we can compare its dynamics with those of the prototype and determine their similarities (Axtell et al., 1996). If both models demonstrate the same dynamics, then we can be more confident that these results are a true attribute of the system rather than a bug in our code or an artifact of the modeling framework we chose. Most decisions involve a number of criteria, frequently competing. Our methodology characterizes the goodness of a given decision by developing a preference structure and an overall utility equation through interaction with subject matter experts. This step is described in detail in the next section.

To explore a variety of system responses, formal design of experiments (Kleijnen et al., 2005) is created to use statistical techniques to manage the inherent combinatorics. These experiments are then run in a high-performance computing environment to investigate a wide variety of behaviors that are implied from the stochastic behavior of the agents. As the simulation runs are completed, two types of analyses are conducted. The first uses optimization techniques such as GAs to drive to areas of optimal parameter combinations, optimality being defined using the utility analysis described below. The second conducts statistical analysis to characterize the relationship of the various parameters within the simulation. While the optimization algorithms will generally find families of solutions, the statistical analyses will provide additional insight into areas of concern as well as statistical anomalies that might have catastrophic consequences. This overall methodology is illustrated in Figure 12.4.

12.5.2

Modeling Goodness

The key to using a model to find an optimal solution or families of optimal solutions is to define the MOPs and from that the MOEs. In many cases the MOPs are readily identified and often easily measured. Consider two MOPs related to sensors: the accuracy and range of detection. To compare the different quantities, the MOPs are mapped to a preference space from zero to one using straightforward utility functions. The mapping can take virtually any mathematical form such as step functions, linear functions, polynomials, and so on. In the current example we will assume the preference structure for the probability of detection as 0 for $P_d < 0.7$ and then exponentially increasing to 1 for $0.7 < P_d < 1.0$. In other words, any solution with $P_d < 0.7$ would not be acceptable.

Once the MOPs have been mapped to the preference space, they are combined into MOEs. For the purposes of this paper, we define MOEs as consisting of a weighted average of one or more instantiated utility functions. An issue often mentioned when using utility functions is that of correctness. How can one determine that the equations being used are an accurate representation of the user preferences? While this question is frequently asked, it is misguided as we seek to measure the qualitative goodness of set solutions. Our experience has shown the effectiveness of standard user elicitation techniques (Teknomo, 2006; Verma et al., 1998) to quantitatively model qualitative preferences.

The utility framework provides a straightforward path to utilize enterprise architecture products. For example, the Federal Enterprise Architecture Framework (as described above) provides a performance reference model (PRM). The PRM defines a hierarchical set of metrics that describe the performance of the enterprise. We suggest that these metrics are used to score the configuration for an enterprise that we can represent as a simulation. Specifically, individual metrics are assessed through simulation, mapped to MOEs as described, and then used to rate the goodness of a given approach. This score can be used to guide the search space of configuration possibilities for the enterprise. In the next section we describe the use of GAs to evaluate a large number of configurations and drive toward optimality.

12.5.3

The Genetic Algorithm Optimization Toolkit

Since their formal introduction in 1975, and even before, by Holland (1975), GAs have been applied to a variety of fields – from medicine and engineering to business – to optimize functions that do not lend themselves to optimization by traditional methods (Back, 1996; Friedberg, 1958). More recently, the study and practical development of the GA by Goldberg (1989) and Jong (1975) has resulted in great growth in the application of GAs to optimization problems. As succinctly stated by Goldberg, GAs are search procedures based on the mechanics of natural selection and natural genetics.

GAs apply random choice as a tool to guide a global search in the space of potential solutions. Rather than focusing on a single-candidate solution (a point in design space), GAs operate on populations of candidate solutions, and the search process favors the reproduction of individuals with better fitness values than those of previous generations (optimal individuals). Whereas calculus-based and gradient (hill-climbing) methods of solution are local in the scope of their search and depend on well-defined gradients in the search space, GAs are useful for dealing with many practical problems containing noisy or discontinuous fitness values. Enumerative searches are also inappropriate for many practical problems as they exhaustively examine the entire search space for solutions; they are only efficient for small search spaces, while the global scope of the GA makes it suitable for problems with large search spaces. Thus, GAs not only differ in approach from traditional optimization methods but also offer an alternative method for cases in which traditional algorithms are inappropriate. Moreover, GAs offer a way to derive populations of robust solutions rather than a single optimal solution that may be very fragile to changes in conditions.

12.5.3.1 Our Application of the Genetic Algorithm Toolkit

We have implemented our optimization capability as a GA with the following characteristics:

- Individual parameterized solutions (the sets of x , y orientation values) are encoded as integers.
- Elitist selection is introduced enabling the retention of the current M best solutions in a population of N individuals, driving convergence and preserving multiple potential optima.
- Elitist selection is introduced enabling the retention of the current M best solutions in a population of N individuals, driving convergence and preserving multiple potential optima.
- Parallel execution is achieved on a 128-processor Linux cluster using a customized version of the Unified Search Framework (Tao and Kalyanaraman, 2003).
- An evolutionary algorithm (EA) capability, modeled after Back (1996), enables normally distributed changes in parameters, versus the uniformly distributed random changes characteristic of GAs.

The EA capability is important in that it enables continuous-valued optimization in the context of a discrete GA. The algorithm is expressed in pseudocode as follows:

```

gen = 0
initialize Population
do while not terminated:
    apply crossover to Population(gen) giving Children(gen)
    apply mutation operators to Children(gen)
    decode and evaluate Children(gen)
    Population(gen+1) = select from(Children(gen)+  

                                    Population(gen))
    gen = gen + 1

```

Crossover and mutation are applied uniformly, and termination is forced after a fixed number of generations. The decode and evaluate step is executed in parallel, where each individual solution in the new Child population is evaluated simultaneously.

12.5.3.2 Finding Solutions

As discussed above, the utility framework can be used to create a fitness function that can be used within our GA toolkit. A key assumption for our approach is that the solution can be parameterized as a function of a number of input variables. We further assume that within the constraints of the solution space these variables describe a multidimensional space. Within this multidimensional space we assert that there are areas of stability and areas of optimality; often these areas are not the same. However, we believe that for a reasonable number of parameterized input variables we can search the space and find these interesting areas.

Once the individual preference curves have been derived, these can be captured by our utility function tool shown in Figure 12.5. For each MOP, the user can specify the type of curve (and tweak its shape via sliders) as well as the weighting constant. Each of these is aggregated into an MOE. These solutions are then translated into binary and encoded in a string that represents a solution. A population of possible solutions is then developed.

Each of the solutions is then translated to an initialization file for a simulation of the phenomena of interest. As the simulation often has a number of stochastic elements, it is run a number of times. MOPs are gathered from the simulation both during and at the completion of each run and, using an equation developed as described above, the candidate solution is scored. The best combination of parameters is used to develop new simulations using standard GA operators such as replication, crossover, and mutation (as discussed in greater detail above). This process is continued until a stable population is obtained or it becomes apparent that there is too much noise in the system and a stable region in solution space is not likely to be found. The simulation provides a straightforward evaluation function, and the GA codifies a heuristic search. Further, the utility framework provides a rich way to quantify the goodness of any solution.

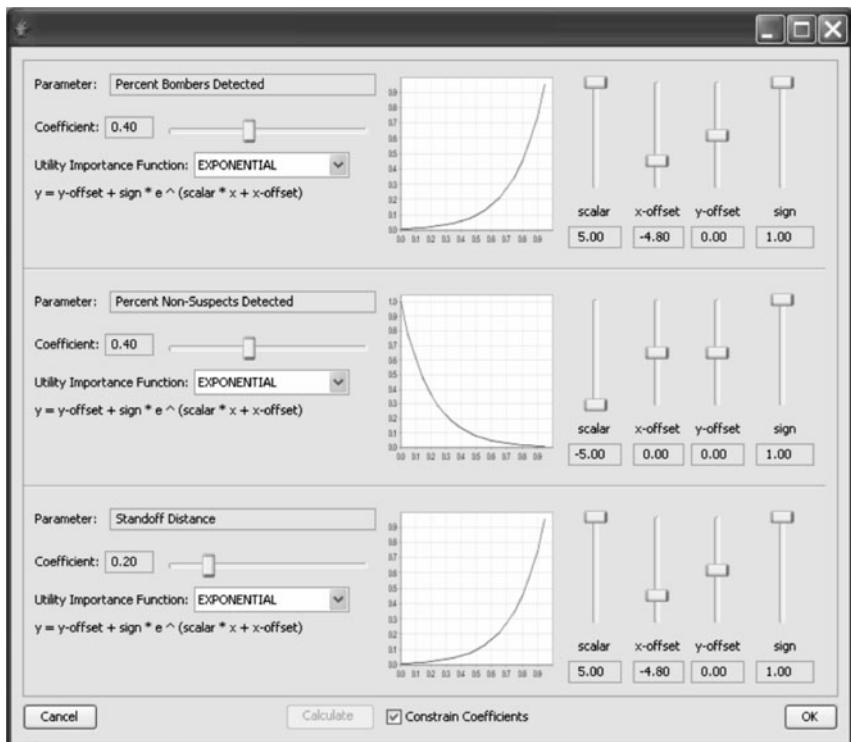


Fig. 12.5 Utility function GUI.

12.6

Case Studies

We developed the aforementioned tools and techniques over a number of years and have applied them to a wide variety of SE problems. Some of these problems include placement and employment techniques for sensors at large venues (Barry et al., 2007a,b), the impact of warfare on civilian populations, and the impact of changes to weapons systems and tactics, techniques, and procedures (Johnson et al., 2007; Koehler et al., 2004, 2007; Meyer et al., 2005). In the interest of space we have chosen to focus on two case studies. The first case study examines how to use a combination of technology and tactics to defend a stadium against terrorists. The second case study deals with the secondary effects of an influenza pandemic and how various systems (such as the US telecommunications infrastructure) and mitigation strategies (such as social distancing) interact to produce unintended outcomes.

12.6.1

Case Study 1: Defending The Stadium

ADS-enhanced SE has been used to explore the best ways to best defend a stadium against possible terrorists. As shown in Figure 12.6, there are three types of agents in the model: Security, Civilians, and bad actors. Bad actors model terrorists – agents that have as their goal getting through the turnstiles undetected carrying explosives. Bad actors must traverse a corridor from left to right where they are likely to encounter sensors as well as security guards. If a bad actor successfully crosses a turnstile, he is considered successful.

Bad actors will take evasive measures to avoid security guards. If the security guards close off potential escape routes and the bad actor considers the situation hopeless, the bad actor will exhibit satisficing behavior and detonate the explosives. In this simulation, bad actors do not act collaboratively.

Security agents have as a primary goal to interdict the bad actors. Security agents are throughout the corridors as well as around the turnstiles. Security agents use fused data from the sensors to evaluate whether a passenger agent is a bad actor or not. As false alarms are possible, nonbad actors may be targeted as bad actors and stopped by the security agents.

The remainder of the agents are innocent civilians that are instantiated as a rate per unit time. The number created each time step is drawn from a random-exponential distribution. Upon instantiation a small percentage of the passengers (0.005%) may be further instantiated as an individual with a bomb or a bad actor.

12.6.1.1 Behavioral Modeling

Extending the work done in modeling stability and support operations (Barry et al., 2004), we refer to the propensity for an individual to carry out an action as a function of a term called activation energy. Activation energy is modeled as a logit function, whose general form is described by (12.1). If an individual's activation

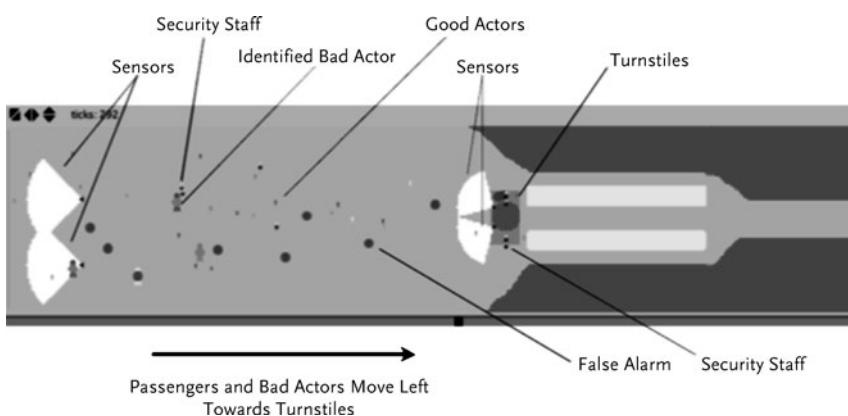


Fig. 12.6 Simulation Setup and Agent Environment.

energy for a given situation exceeds an endogenous threshold, then an action is taken.

$$f(x) = Lb \frac{(Ub - Lb)}{1 + (1 + e^{-\text{slope}(k-m)})} , \quad (12.1)$$

where Lb and Ub are the lower and upper bounds of the function, respectively. Slope is a parameter that adjusts the slope of the function. In this framework, we suggest that one can view the slope parameter as a proxy for volatility; an agent with a large slope is considerably more unpredictable than an agent with a smaller slope. The mean of the function m is used as a proxy to model how close the context must be to the ideal value to increase activation energy. The values on the x -axis represent k , the average distance of the actual context parameters from the ideal values as measured in percent. (Negative values of k indicate that the average of the parameters is better than the context for activation.) Unlike other approaches, this framework does not create a detailed goal hierarchy (Johns and Silverman, 2001) but assumes the bad actor has a single goal of some terrorist action.

We can also use activation energy as an external indicator of the agent's tendency for action. Agents that have subthreshold but measurable activation energy can be considered to be telegraphing their intention to act. Consider the case of an agent that shows no external indicators of untoward behavior until the conditions are ripe for action. This agent would be modeled with a large slope. On the other hand, agents with smaller slopes show their intentions and build up to activation much more slowly.

12.6.1.2 Representing Satisficing Behavior

Satisficing is a behavior that attempts to achieve at least some minimum level of a particular variable, but that does not necessarily maximize its value (Simon, 1957). Often satisficing is used within a learning context as an alternative to strict optimization (Izquierdo et al., 2004). It is fully reasonable and expected that the agents being modeled would exhibit satisficing behavior. Consider the example above, where the agent is taking the decision to carry out the negative action. In the example, only when the agent is directly at the center of a crowded area will the activation energy be sufficient to trigger an action.

However, consider the case where the agent has determined that the optimal combination of parameters is too restrictive, in other words the allowable distance from the actual context to the ideal context is too small for activation. In this case the agent may satisfice, which is represented by a reduction in the activation energy threshold. In the example above, consider the case where the agent may decide that an activation energy threshold of 0.6 is adequate. This would then open two other combinations of k ; the activation energy threshold is reached in the situation where the agent is in the center of the area but the crowds are medium or the agent is 40 meters from the center of a crowded venue.

Satisficing behavior is observed in the model when the agent perceives no other options. For example, if an agent perceives impending arrest, the agent may trigger an explosive device even if the context is significantly far from optimal. This

aspect of behavior is a significant factor that contributes to the adversarial reasoning. To thwart the undesirable actions of a given agent, the opposition will attempt to increase the requisite activation energy. The form of this may be to increase the perceived distance between the actual and ideal context variables as well as to morph the shape of the activation energy curve.

12.6.1.3 Sensor Modeling

Sensors are categorized by one of three types: two types of passive millimeter wave (MMW) and one type of infrared (IR). The sensors are placed in fixed locations in the environment. The primary purpose of the sensors is the detection and classification of concealed weapons under a person's clothing at this public venue. It is often desirable to screen people from a standoff distance to reduce the chances of long lines or unnecessary crowding. However, the composition, placement, and tasking of semiautonomous sensor assets, the extraction and utilization of actionable information from these systems, and the generation of reasonable concepts of operation are important problems that have not yet been resolved.

The first stage of this effort involved the development of passive MMW sensor models that could be used to accurately predict the detection and classification probabilities of concealed weapons at range. In the MMW band, objects are described by their apparent temperature – a combination of the actual temperature and the temperature of the reflected background. For a metallic object, the apparent temperature is basically the background temperature. A passive MMW imager uses a radiometer (energy detector) to estimate the apparent temperature in a beam pointed at the object. After accounting for system noise and possible antenna losses, the effective contrast temperature between an object and its background across each beam can be determined as a function of the imager aperture size, focal length, detector time-bandwidth product, weapon size, sky and ground temperatures, and the corresponding weapon emissivities (the amount an object radiates). The detection probability is determined using the effective contrast temperature and the number of beams covering the object at a fixed false alarm probability. A similar procedure is used to determine the classification probabilities for various weapon sizes and shapes with possibly different material compositions.

12.6.1.4 VV&A of the Simulation

Consistent with the discussion, the VV&A process was ongoing from the very start. The initial agent terrain was developed by examining the general geometry of the entrances to large venues. Agent behavior (civilian, security, and terrorist) was determined in close collaboration with subject matter experts. As discussed above, as the overall process of determining the composition of the system moved forward, the verisimilitude of the simulation also needed to improve. Some of the secondary features that were added to the simulation to keep it in step with the details of the SE included sensor occlusion from other people and obstacles and evasive behavior for the terrorists.

12.6.1.5 Experimentation

The simulation was run for a wide variety of sensor placements, varying the x and y coordinates of two passive IR sensors and a passive MMW sensor as well as their respective angles with respect to the incoming traffic. Additionally, the amount of evidence necessary for the security guards to interdict a possible suspect was varied. In other words, varying the sensors directly affected the amount of information received. Changing the evidence threshold resulted in varying the likelihood of false positives or false negatives.

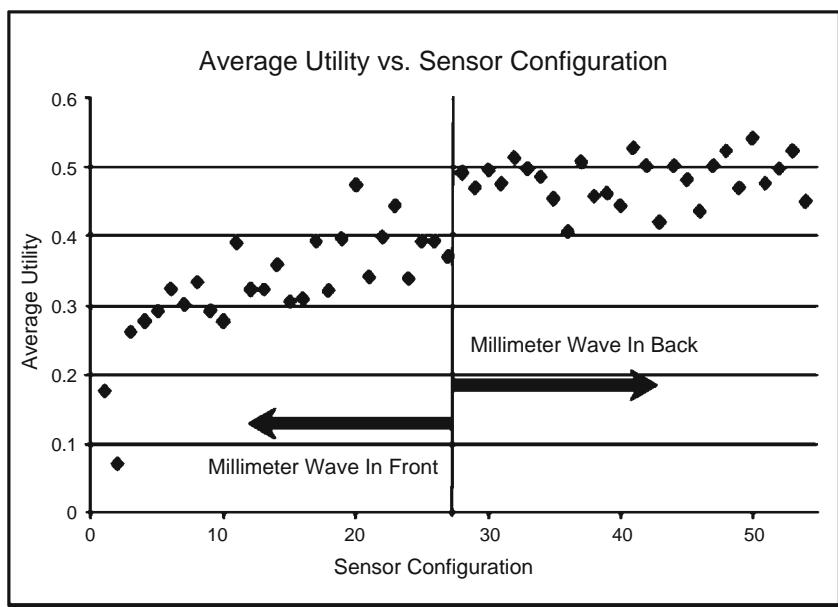
As discussed above, the goodness of a configuration of sensors in combination with the likelihood of interdiction was modeled as a utility function. Two primary measures of effectiveness were considered: for a given configuration the probability that an explosive would be detected and the probability that there would be a false alarm, meaning an explosive was indicated where no explosive existed. Two thousand possible combinations were modeled; each iteration was run 30 times as there were a number of stochastic features in the simulation.

The data indicated a number of interesting aspects. First, for any given configuration there was wide variability in the results. However, some general trends were observed. Figure 12.7a plots utility against numbered sensor configurations. From sensor configurations 0–26, the general trend is that the MMW sensor starts out significantly in front of one of the IR sensors and then progressively moves closer. At sensor configuration 27–53, the MMW sensor is behind the IR sensors. As a general practice, it would appear that placing the IR sensors behind the MMW sensor is preferred.

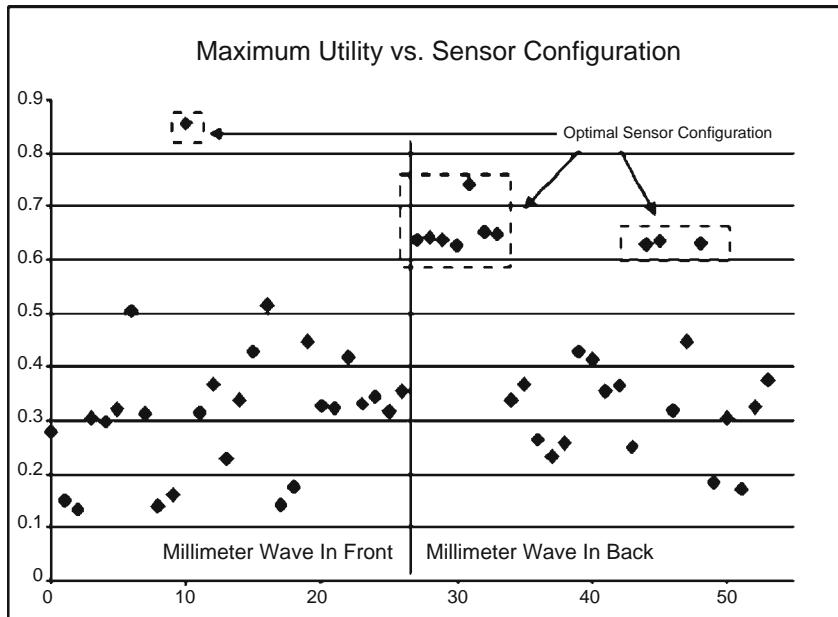
Figure 12.7b illustrates the maximum utility plotted against sensor configuration. Consistent with the indication from that shown in Figure 12.2, the best performing configurations from a maximum utility perspective are where the MMW sensor is behind the IR sensors. There are two groupings of similar configurations that performed roughly the same. However, the highest utility occurred during a run using sensor configuration 10, where the MMW sensor is in front of the IR sensors. This can be viewed as an anomalous result, possible but unlikely as the average utility is much less. For both the average utility as well as the maximum utility, the optimal temperament configuration for the security forces was a mildly but not overly aggressive profile. Overly aggressive security forces result in too many false alarms, whereas passive security forces miss too many threats. This result seemed relatively consistent across all of the sensor configurations.

12.6.1.6 Implications

This case study presented a SE approach to illustrate the use of simulation to optimize complicated systems consisting of both behavioral and technology components to illustrate using simulation to optimize the desired behavior of a HCS. We have compared these results to published theory (Kaplan and Kress, 2005) and found good agreement. The methodology appears to be extensible to larger and more complex models.



(a) Utility vs. Sensor



(b) Maximum Utility vs. Sensor

Fig. 12.7 Configurations.

12.6.2

Case Study 2: Secondary Effects from Pandemic Influenza

There is widespread concern among policymakers and public health experts about the possibility of a worldwide pandemic outbreak (Bauer et al., 2007). Although severe pandemics are rare events, they are a recurring phenomenon, with three pandemics in the 20th century. Today, scientists are particularly worried about H5N1, a strain that has spread through bird populations across Asia, Africa, and Europe. Pandemics cannot be accurately predicted and can occur at any time of the year. Health and Human Services Secretary Leavitt (2005) warned that when it comes to a pandemic we are overdue and underprepared. In addition to direct medical consequences, a pandemic is expected to have impacts across the economy. Travel restrictions and quarantines will limit mobility and may disrupt supply chains around the world. Additionally, the pandemic could potentially lead to up to 2 million deaths in the USA and a drop of close to 5% of Gross Domestic Product as reported by the Congressional Budget Office (2006).

Several national pandemic plans (Council, 2006; o. H. Security, 2006; o. H. a. H. Services, 2007) identify telecommuting as a key component of the national response to a pandemic influenza. Telecommuting is recommended as a social distancing mechanism that can limit disease spread while allowing businesses to continue to function. The Pandemic Severity Index defined by the Centers for Disease Control advises businesses to consider following telecommuting programs for Category 2 and 3 pandemics and recommends telecommuting programs as an important component of the planning strategy for Category 4 and 5 pandemics (o. H. a. H. Services, 2007). More specifically, the Homeland Security Council (HSC) has advised businesses to plan for up to 40% of their employees to be absent during the two-week peak of a six- to eight-week pandemic wave (Council, 2006).

12.6.2.1 Research Questions and Context

Enabling this telecommuting strategy to succeed will be key in mitigating disease spread while also facilitating businesses to continue operations and limit additional impact on the economy. Given the telecommuting strategy advocated in several national pandemic plans, the following questions remain:

- Is the telecommuting strategy technically feasible for an influenza pandemic?
- What planning can be done to better prepare for telecommuting during a pandemic influenza?

Spurred in part by the adoption of broadband Internet access, telecommuting is a growing trend, particularly in the services sector of the economy (e.g., approximately 8% of US workers have an employer that allows them to telecommute full-time and roughly 20% of the workforce engages in telecommuting at least once a month (WorldatWork, 2007)). Despite the growing telecommuting trend, the technical feasibility of widespread telecommuting during a pandemic scenario given the existing telecommunications infrastructure has not been established.

Two areas of particular concern are the ability of enterprise networks and residential Internet access networks to handle increased telecommuting traffic. Many businesses that have limited reliance on telecommuting today may not be capable of handling the anticipated increase in telecommuting traffic during a pandemic. Additionally, telecommuters could face congestion on the residential Internet access networks on which they connect. A surge in traffic on residential Internet access networks from users who are home due to illness or part of the worried well may affect the network performance for workers attempting to telecommute.

A recent war-game-style simulation of a pandemic scenario judged that the telecommunications infrastructure would be severely strained during a pandemic (Forum and Hamilton, 2006). In light of such predictions, this study addressed the above questions by analyzing the potential impact on the communications infrastructure in the event of a pandemic influenza in the United States. The study also identified steps that can be taken to better prepare the nation to support the pandemic telecommuting strategy.

12.6.2.2 Scope of the Study

The dynamics between pandemic spread and Internet user behavior were assumed to occur primarily at the scale of a major metropolitan area. The pandemic is anticipated to spread across the country due to people traveling between major cities. Within metropolitan areas, population densities and human-to-human interaction within the metropolitan area are anticipated to largely drive the pandemic spread. From the perspective of telecommuters, the majority of communications traffic is anticipated to be between telecommuters and an enterprise network and therefore mostly confined within the metropolitan area. Thus, the intrinsic scale of understanding the relationships between pandemic spread, Internet user behavior, and Internet performance is assumed to be at a metropolitan level.

12.6.2.3 Related Research

Leveraging an earlier large-scale model of a deregulated electricity market (Kortarov, 2004), we developed a modeling framework to better understand the relationship between pandemic spread, Internet user behavior, and Internet performance. Our modeling framework includes the following: a metropolitan-scale pandemic model based largely on seminal epidemiological research (Kermack and McKendrick, 1927), a network user model to depict various Internet user profiles, a metropolitan-scale model of the Internet, and a visualization tool based on Google Earth (Google, 2008). The three models (i.e., pandemic, network users, and Internet) were all instantiated in a large-scale, discrete-event simulation platform to leverage optimistic, parallel simulation techniques (Yaun et al., 2003).

12.6.2.4 Data

Using data from the Census Bureau and the Department of Transportation (o. T. Statistics, 2007), the pandemic model examined the potential movement of the Internet user population between home, office, and school locations during a pandemic. Using actual trace-route data (Spring, 2002) derived from leading Internet

mapping techniques (Spring et al., 2002), the network model depicted the impact of the change in Internet user behavior on residential Internet access networks. In particular, we examined the ability of telecommuters to perform their functions from residential Internet access networks when increased numbers of Internet users were online during a pandemic. The modeling framework was also used to evaluate the potential impact of phased preparations to improve network performance.

12.6.2.5 Model Structure

This section provides an overview of the modeling architecture constructed. The modeling architecture can be logically separated into three models: Pandemic Spread, Network Users, and Network. The basic premise behind the modeling architecture is that the Pandemic Spread model simulates the disease spreading through a metropolitan area. Based upon sickness and other factors, the Network Users component has people decide whether to remain at work/school or to go home. The Network Users component also assigns a traffic profile to each type of Internet user. When these people are at home, they put their traffic profile on the residential network. The Network model then simulates the conditions of a typical access network in response to the offered traffic load.

The Pandemic Spread model is an ADS intended to represent the spread of a pandemic across a metropolitan area. The purpose of the Pandemic Spread model in the overall modeling architecture is to move individuals through the different stages of the disease in order to cause individuals to change location. Since the objective of this study is to examine the impact on the communications network, the purpose of the Pandemic Spread model is to propagate the disease through the population via human-to-human interaction.

Several infectious disease models have been developed and there is ongoing debate on the representation of certain epidemiology parameters [for more detail on these open questions, refer to the Models of Infectious Disease Agent Study (MIDAS) (O. G. M. Sciences, 2006) project and the related modeling efforts]. For the purposes of this study, many of those debated details are not relevant, as they will not significantly affect the maximum number of people infected in a given area at a given time. Therefore, the Pandemic Spread model in this study was based largely on seminal epidemiological research (Kermack and McKendrick, 1927) and its extensions (Watts et al., 2005; Epstein et al., 2002). Spread of the disease follows the general Susceptible-Exposed-Infected-Recovered (SEIR) algorithm. In the SEIR algorithm, an individual contracts the disease through its interaction with other individuals and passes through the four stages of the disease based on a variety of disease parameters such as time of incubation, time of contagion, time of recovery, mortality rate, and so on.

The SEIR algorithm was implemented with human interaction in homes, offices, and schools in a metropolitan area. The home locations were derived from data on census tract populations from the Year 2000 census. Office and school locations were derived from data from the Department of Transportation that showed morning commuting patterns between census tracts. The Pandemic Spread model was

calibrated to the homogeneous interaction case to ensure proper implementation of the SEIR algorithm.

The Network Users model reflects two major aspects of the problem: (a) people deciding whether or not to leave their office or school and return home and (b) the users' network traffic profiles when accessing the Internet from home. People may decide to go home based on a number of factors. The primary driver is whether an individual is infected with the disease. People who begin feeling sick while at office or school return home. People may also decide to go home based on having to take care of a sick family member at home or in an attempt to avoid contracting the disease (i.e., the worried well).

Each type of person was assigned a user traffic profile. Four categories of users were created: At Home User, Telecommuter, Financial Telecommuter (representative of a superuser requiring a high quality of service), and Child User. Public source information on typical network traffic was used to construct a traffic profile for each category. The traffic profiles make assumptions about the type of applications run and how long to run each application. These assumptions were aggregated into an average desired throughput rate for each user category. Real-world data from a recent Federal Reserve Board telecommuting exercise was used to verify assumptions about telecommuter traffic profiles. All of these traffic profiles aggregate to generate a total traffic load within the Network model.

The Network model includes the user locations and corresponding traffic profiles as input to generate the traffic load for the network. The model then mapped a separate traffic flow to every network user. The status of the network is measured at various points in the Network model. For example, each router can report queuing delays at any port. Network user performance in terms of throughput can be measured for every user.

12.6.2.6 Findings

In order to further quantify the potential impact of a pandemic on the communications infrastructure, several pandemic scenarios were selected to analyze network performance. From the perspective of impact on the communications infrastructure, the home network user population during the pandemic is the most important factor. Stated differently, this study did not focus on the lethality of the pandemic per se, but instead on the secondary effects of the pandemic on network performance. In this study, the pandemic scenarios are characterized based on their effect on the home network user population. Three pandemic scenarios of particular interest include a low absenteeism pandemic, 40% absenteeism pandemic, and a high absenteeism pandemic. In this context, absenteeism refers to people absent from the workplace and therefore accessing the network from their homes. The low- and high-absenteeism pandemic cases give a lower and upper bound for reasonable pandemic parameters. The 40% absenteeism case corresponds to the Homeland Security Council guideline that businesses should plan for 40% absenteeism. Figure 12.8 below shows results for the low-absenteeism case; space limitations preclude the presentation of additional results.

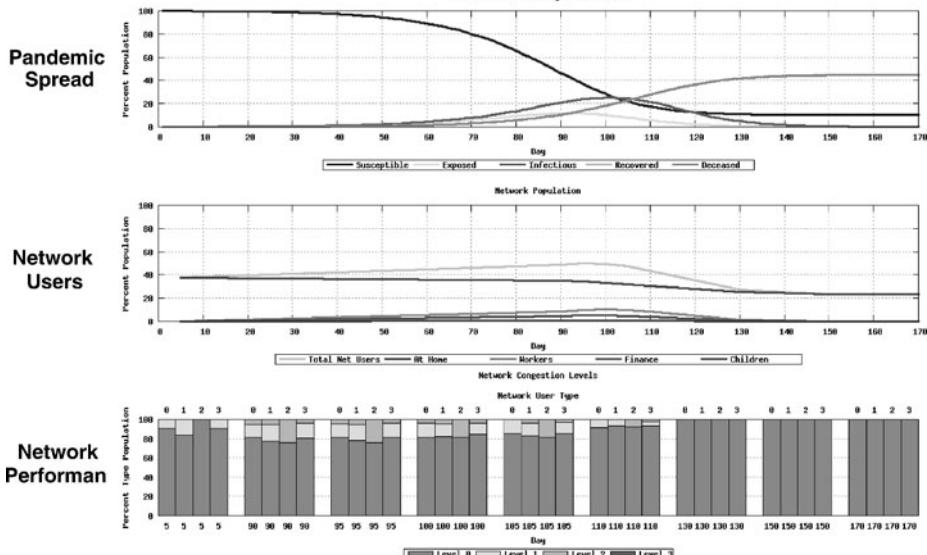


Fig. 12.8 Low absenteeism pandemic scenario.

For a given pandemic scenario, three time-series graphs are generated: Pandemic Spread, Network Users, and Network Performance. The Pandemic Spread graph shows the percentage of the population that contracts the disease as they progress from Susceptible to Exposed to Infected to Recovered/Deceased. The Network Users graph shows how the home network user population changes as the disease spreads across the metropolitan area. Home network users are categorized into four groups with different traffic profiles:

- Type 0 At Home – Network users who are normally at home;
- Type 1 Workers – Telecommuters working from home locations;
- Type 2 Finance – Financial industry telecommuters working from home locations;
- Type 3 Children – School-aged children home from school.

The Network Performance graph shows end-user network performance during the pandemic for each home network user category. For analysis purposes, network user performance is categorized into levels based on the ability of users to achieve their desired average bit rate.

- Level 0 – Network users are able to achieve desired bit rate;
- Level 1 – Network users achieve less than the desired bit rate but at least 1/2 the desired bit rate;
- Level 2 – Network users achieve less than 1/2 but greater than 1/10 the desired bit rate;
- Level 3 – Network users are unable to communicate due to TCP maximum backoff time.

In the low-absenteeism pandemic scenario displayed in Figure 12.8, the home network user populations increase slightly. This results in a small percentage of the population experiencing congestion during the peak of the pandemic. For the telecommuter populations (Type 1 and Type 2), over 80% do not experience any degradation in network throughput, even during the peak of the pandemic wave. Of the telecommuters that do experience congestion, those with lower bandwidth requirements (Type 1) are generally able to achieve at least one half their desired throughput. Telecommuters with higher bandwidth requirements (Type 2) experience slightly more congestion during the two-week peak of the pandemic. In the low-absenteeism pandemic scenario, the telecommuting strategy seems to be successful for the majority of telecommuters.

12.6.2.7 Implications of Secondary Effects from Influenza Pandemic

In extreme cases, an influenza pandemic is a rare but potentially catastrophic phenomenon. Some estimate a bird-flu pandemic could potentially lead to up to 2 million deaths in the US and a drop of close to 5% of GDP. Many federal agencies have advocated for social distancing in the event of a pandemic and hypothesized that telecommuting over the public telecommunications infrastructure presented a technically feasible option for business continuity planning.

This study used computational models to test this hypothesis for various pandemic scenarios for a realistic user population, heterogeneously distributed across a major metropolitan population. The findings demonstrated the various factors that policymakers must consider when evaluating the technical feasibility of the telecommuting strategy. Although not described in detail here, the findings also emphasized the critical importance of public compliance with various attempts to maximize the effectiveness of the public telecommunications infrastructure to support business continuity. The citizens of the area experiencing the pandemic are the critical actors in this HCS, and SE problems of this scale and complexity require ADS to reflect the inherent and interdependent dynamics.

12.7 Summary

As our development of systems becomes increasingly complex, the practice of SE must evolve. Our tools and techniques must expand to handle issues of combinatorial complexity, long-term system evolution, managerial independence, and emergent behavior. There is an increasing recognition of the fact that viable models of the organizational element of an enterprise require concordant maturation of our toolset. This phenomenon is a natural effect of engineering and provides interesting opportunities for engineering and research.

ADS can provide a valuable tool to gain insight into complex enterprises. The application of ADS builds on a long history of successful work in organizational theory married to traditional SE and enterprise architecture. Our experience indicates

that with significant computational capabilities we can find statistically interesting insights into HCSs, allowing us to provide quantitative engineering direction to all phases of the systems life cycle for an HCS. The use of ADS has been shown to provide significant value for the design and implementation of HCS and will continue to gain importance as our systems grow in scope and complexity.

References

- E. Aarts and J. Korst. *Simulated Annealing and Boltzmann Machines: a Stochastic Approach to Combinatorial Optimization and Neural Computing*. John Wiley & Sons New York, 1989.
- P. Anderson. Complexity theory and organization science. *Organization Science*, 10, 216–232, 1997.
- R. Axelrod. Advancing the art of simulation in the social sciences. *Simulating Social Phenomena*, 40, 385–390, 1995.
- R. Axelrod. *The Complexity of Cooperation: Agent-Based Models of Competition and Collaboration*. Princeton University Press, Princeton, NJ, 1997.
- R.L. Axtell. Why agents? on the varied motivations for agent computing in the social sciences, 2000. R.L. Axtell. Why agents? on the varied motivations for agent computing in the social sciences, 2000.
- R.L. Axtell. Three distinct kinds of empirically relevant agent-based models. *Social and Economic Dynamic*, (24), 2005.
- R.L. Axtell. Zipf distribution of US firm sizes. *Science*, 293, 1818–1820, 2001.
- R.L. Axtell, R. Axelrod, J.M. Epstein, and M.D. Cohen. Aligning simulation models: A case study and results. *Computational and Mathematical Organization Theory*, 1, 123–141, 1996.
- B. o. T. Statistics. *The Intermodal Transportation Database*. Bureau of Transportation, 2007.
- T. Back. *Optimization through Evolution and Recombination*. Oxford University Press, Oxford, UK, 1996.
- C.I. Barnard. *The Functions of the Executive*. Harvard University Press, Cambridge, MA, 3rd edition, 1938.
- P. Barry, M. Koehler, G. Jacyna, T. Bergen-Hill, and M. Tierneyr. M. Koehler, P. Barry, B. Widdowson, and A. Forsyt. *Proceedings of the Agent 2004 Conference*, 2004.
- P. Barry, G. Jacyna, and M. Koehler. Spy v. spy: A utility based approach to agent based adversarial reasoning. *Proceedings of the Agent 2007 Conference*, 2007a.
- P. Barry, M. Koehler, G. Jacyna, T. Bergen-Hill, and M. Tierneyr. Using generative analysis for homeland security: Modeling the possibilities and the probabilities. *Proceedings of the Agent 2007 Conference*, 2007b.
- D.W. Bauer, E.R. Beeker, J.W. Darkoch, M.J. Leamy, E. Page, and B.F. Tivnan. Pandemic influenza impact on communications networks study. *Department of Homeland Security Technical Report*, 2007.
- E. Beth. Semantics of physical theories. *The Concept and the Role of the Model in Mathematics and Natural and Social Sciences*, pp. 48–51, 1961.
- S.R. Buss, C.H. Papadimitriou, and J.N. Tsitsiklis. On the predictability of coupled automata: An allegory about chaos. *Complex Systems*, 5, 525–539, 1991.
- K.M. Carley. Computational and mathematical organization theory: Perspective and directions. *Computational and Mathematical Organization Theory*, 1, 39–56, 1995.
- K.M. Carley and D.M. Svoboda. Modeling organizational adaptation as a simulated annealing process. *Sociological Methods & Research*, 25, 138–168, 1996.
- P.G. Carlock and R.C. Fenton. Systems of systems (sos) enterprise systems engineering for information intensive organizations. *Systems Engineering*, 4(1), 242–251, 2001.
- M.D. Cohen, J.G. March, and J.P. Olsen. A garbage can theory of organizational choice. *Administrative Science Quarterly*, 17, 1–25, 1972.
- I. COS. What is systems engineering? *I. C. o. S. Engineering*, 2008.
- Congressional Budget Office (2006) A Potential Influenza Pandemic: Possible Macro-

- economic Effects and Policy Issues, revised July 27, 2006.
- H.S. Council. *National Strategy for Pandemic Influenza Implementation Plan*. D. o. H. Security, Ed., 2006.
- M.E. Csete and J.C. Doyle. Reverse engineering of biological complexity. *Science*, 295, 1664–1669, 2002.
- R.M. Cyert and J.G. March. *A Behavioral Theory of the Firm*. Prentice-Hall, Englewood Cliffs, NJ, 1963.
- D. of Defense. *Department of Defense Architecture Framework (DoDAF) Version 1.5 Volume II*. Department of Defense, 2007.
- D. o. H. a. H. Services. *Community Strategy for Pandemic Influenza Mitigation*. D. o. H. a. H. Services, 2007.
- D. o. H. Security. *Pandemic Influenza Guide for Critical Infrastructure and Key Resources*. D. o. H. Security, Ed., 2006.
- K.J. Dooley. Simulation research methods. *Companion to Organizations*, J. A. C. Baum, Ed., 2002.
- K.J. Dooley and A.H. Van de Ven. Explaining complex organizational dynamics. *Organization Science*, 10, 358–375, 1999.
- Editorial. Credit swap worries go mainstream. *in Naked Capitalism*, 2005.
- J.M. Epstein. Agent-based computational models and generative social science. *Complexity*, 4, 41–60, 2002.
- J.M. Epstein and R.L. Axtell. *Growing Artificial Societies: Social Science from the Bottom Up*. Brookings Institution Press, Washington DC, 1996.
- J.M. Epstein and R.A. Hammond. Non-explanatory equilibria: An extremely simple game with (mostly) unattainable fixed points. *Complexity*, 7, 18–22, 2002.
- J.M. Epstein, D.A.T. Cummings, S. Chakravarty, R.H. Singa, and D.S. Burke. Multi-scale, resurgent epidemics in a hierarchical metapopulation model. *Brookings Institution-Johns Hopkins University Center on Social and Economic Dynamics Working Paper*, (31), 2002.
- FEA. *Federal Enterprise Architecture (FEA) Consolidated Reference Model Document Version 2.3*. O. o. M. a. Budget, Ed., 2007.
- T.C. Ford, J. Colombi, S. Graham, and D. Jacques. A survey on interoperability measurement. *Proceedings of the 12th International Command and Control Research and Technology Symposium*, 2007.
- W.E. Forum and B.A. Hamilton. *Influenza Pandemic Simulation*. 2006.
- R.M. Friedberg. A learning machine: Part ii. *IBM Journal of Research and Development*, 3, 282–287, 1958.
- D. Gamerman and H. Lopes. *Markov Chain Monte Carlo: Stochastic Simulation for Bayesian Inference*. Chapman and Hall, 2006.
- N. Gilbert. Editorial. *Journal of Artificial Societies and Social Simulation*, 1, 1998.
- N. Gilbert and K.G. Troitzsch. *Simulation for the Social Scientist*. Open University Press, 1999.
- D.E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, Reading, MA, 1989.
- Google. *Google Earth*. 2008.
- D. Hartley. Verification & validation in military simulations. *Proceedings of the 1997 Winter Simulation Conference*, 1997.
- J.H. Holland. *Hidden Order: How Adaptation Builds Complexity*. Addison-Wesley, Reading, MA, 1995.
- J.H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, MI, 1975.
- J.H. Holland, K.J. Holyoak, R.E. Nisbett, and P.R. Thagard. *Induction: Process of Inference, Learning, and Discovery*. MIT Press, Cambridge, MA, 1986.
- L.R. Izquierdo, N. Gotts, and J.G. Polhill. Case-based reasoning, social dilemmas, and a new equilibrium concept. *Journal of Artificial Societies and Social Simulation*, 7, 125–146, 2004.
- M. Johns and B. Silverman. How emotion and personality effect the utility of alternative decisions: A terrorist target selection case study. *Proceedings of the 10th Conference On Computer Generated Forces and Behavioral Representation*, 2001.
- S. Johnson, M. Koehler, and D. Quinn. The importance of being docked. *Proceedings of the Agent 2007 Conference*, 2007.
- K.A. De Jong. *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*. University of Michigan, Ann Arbor, MI, 1975.
- E.H. Kaplan and M. Kress. Operational effectiveness of suicide-bomber-detector

- schemes: A best case analyst. *Proceedings of the National Academy of Science*, 7, 10399–10404, 2005.
- S. Kauffman. *The Origins of Order: Self-Organization and Selection in Evolution*. Oxford University Press, New York, 1993.
- W.O. Kermack and A.G. McKendrick. Contributions to the mathematical theory of epidemics. *Proceedings of the Royal Society*, 115A, 700–721, 1927.
- J.P.C. Kleijnen, S.M. Sanchez, T.W. Lucas, and T.M. Cioppa. State-of-the-art review: A user's guide to the brave new world of designing simulation experiments. *INFORMS Journal on Computing*, 17, 263–289, 2005.
- M. Koehler, P. Barry, and T. Meyer. Sending agents to war. *Proceedings of the Agent 2004 Conference*, 2004.
- M. Koehler, T. Meyer, A. McLeod, C. Burke, S. Johnson, and P. Barry. Visualization of systems of systems simulations: Density plots and trajectory storyboarding. *Proceedings of the 1st Annual IEEE Systems Conference*, pp. 1–7, 2007.
- V.S. Koritarov. Real-world market representation with agents. *IEEE power & energy magazine*, 2, 39–46, 2004.
- A.J. Kriegel. *Behind the Wizard's Curtain: An Integration Environment for a System of Systems*. CCRP, Washington DC, 1999.
- M. Leavitt. Pandemic influenza impact on communications networks study, 2005.
- M. Leavitt. Pandemic influenza impact on communications networks study, 2005.
- D.A. Levinthal. Adaptation on rugged landscapes. *Management Science*, 43, 934–950, 1997.
- D.A. Levinthal and M. Warglien. Landscape design: Designing for local action in complex worlds. *Organization Science*, 10, 342–357, 1999.
- M.R. Lissack. Editor's note. *Emergence*, 1, 3–4, 1999.
- M. Maier, D. Emery, and R. Hilliard. Ansi/ieee 1471 and systems engineering. *Systems Engineering*, 7(3), 257–270, 2004.
- M.W. Maier. Architecting principles for systems-of-systems. *Systems Engineering*, 1, 1998.
- J.G. March. Exploration and exploitation in organizational learning. *Organization Science*, 2, 71–87, 1991.
- B. McKelvey. Avoiding complexity catastrophe in coevolutionary pockets: Strategies for rugged landscapes. *Organization Science*, 10, 294–321, 1999a.
- B. McKelvey. Self-organization, complexity catastrophe, and microstate models at the edge of chaos. *Variations in Organization Science*, pp. 279–310, 1999b.
- B. McKelvey. Complexity theory in organization science: Seizing the promise or becoming a fad? *Emergence*, 1, 5–32, 1999c.
- B. McKelvey. Model-centered organization science epistemology. *Emergence*, 1, 752–780, 2002.
- B. McKelvey. Quasi-natural organization science. *Organization Science*, 8, 352–380, 1997.
- T.E. Meyer, M.T.K. Koehler, P.S. Barry, and B.F. Tivnan. How simple is simple enough? military modeling case studies. *Proceedings of the Agent 2005 Conference on Generative Social Processes, Models and Mechanisms*, pp. 193–202, 2005.
- N. I. o. G. M. Sciences. *Models of Infectious Disease Agent Stud.* 2006.
- L. Osborne, J. Brummond, R. Hart, M. Zarean, and S. Conger. *Clarus: Concept of Operations*. F. H. Administration, 2005.
- G. Polya. *Mathematics and Plausible Reasoning, Volume I: Induction and Analogy in Mathematics*. Princeton University Press, Princeton, NJ, 1954.
- J. W. Rivkin. Imitation of complex strategies. *Management Science*, 46, 824–844, 2000.
- A.P. Sage. Systems of systems: Architecture based systems design and integration. *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, pp. 123–129, 2005.
- A.P. Sage and C.D. Cuppan. On the systems engineering and management of systems of systems and federations of systems. *Information, Knowledge, and Systems Management*, 2, 325–345, 2001.
- T.C. Schelling. *Micromotives and Macrobbehavior*. Norton, New York, 1978.
- W.R. Scott. *Institutions and Organizations*. Sage, Thousand Oaks, CA, 1995.
- D.A. Shum. *The Evidential Foundations of Probabilistic Reasoning*. Northwestern University Press, Evanston, IL, 1994.
- H.A. Simon. *Administrative Behavior*. Prentice-Hall, New York, 3rd edition, 1976.

- H.A. Simon. A behavioral model of rational choice. *Quarterly Journal of Economics*, 69, 99–118, 1955.
- H.A. Simon. *The Sciences of the Artificial*. Cambridge, MA, Washington DC, 3rd edition, 1999.
- H.A. Simon. *Models of Man: Social and Rational; Mathematical Essays on Rational Human Behavior in a Social Setting*. John Wiley and Sons, 1957.
- O.J. Sorenson. *The Complexity Catastrophe in the Computer Industry: Interdependence and Adaptability in Organizational Evolution*. Department of Sociology, Stanford University, Palo Alto, CA, 1997.
- N. Spring. *Raw Traces*. University of Washington, 2002.
- N. Spring, R. Mahajan, and D. Wetherall. Measuring isp topologies with rocketfuel. *Special Interest Group on Data Communication (SIGCOMM)*, 2002.
- F. Suppe. *The Semantic Conception of Theories and Scientific Realism*. University of Illinois Press, Urbana-Champaign, IL, 1989.
- Y. Tao and S. Kalyanaraman. A Unified Search Framework for Large-scale, Black-Box Optimization. Rensselaer Polytechnic Institute, 2003.
- E. Tatara, M.J. North, T.R. Howe, N.T. Collier, and J.R. Vos. An introduction to repast modeling by using a simple predator-prey example. *Proceedings of the Agent 2006 Conference on Social Agents: Results and Prospects*, 1961.
- K. Teknomo. *Analytic Hierarchy Process (AHP) Tutorial*, 2006.
- B.F. Tivnan. Coevolutionary dynamics and agent-based models in organization science. *Proceedings of the 2005 Winter Simulation Conference*, pp. 1013–1021, 2005.
- T. Toffoli and N. Margolus. *Cellular Automata Machines*. MIT Press, Cambridge, MA, 1987.
- A. van Duyn. Cds sector weighs Bear Stearns backlash. *Financial Times*, 2008.
- D. Verma, R. Chilakapati, and W.J. Fabrycky. Analyzing the quality function deployment (qfd) matrix: An expert system based approach to identify inconsistencies and opportunities. *Journal of Engineering Design*, 9, 252–262, 1998.
- D.J. Watts, R. Muhamad, D.C. Medina and P.S. Dodds. Multiscale, resurgent epidemics in a hierarchical metapopulation model. *Proceedings of the National Academy of Science*, 102, 11157–11162, 2005.
- P. Wegner. Why interaction is more powerful than algorithms. *Communications of the ACM*, 40, 80–91, 1997.
- P. Wegner and D. Goldin. Computation beyond turing machines: Seeking appropriate methods to model computing and human thought. *Communications of the ACM*, 46, 100–102, 1998.
- K.E. Weick. What theory is not, theorizing is. *Administrative Science Quarterly*, 40, 385–390, 1995.
- U. Wilensky. *NetLogo*. Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL, 1989.
- WorldatWork. *Telework Trendlines for 2006*, 2007.
- G.R. Yaun, D. Bauer, H.L. Bhutada, C.D. Carothers, M. Yuksel, and S. Kalyanaraman. Large scale network simulation techniques: Examples of tcp and ospf models. *ACM SIGCOMM Computer Communications Review*, 33, 27–41, 2003.
- J. Zachman. A framework for information systems architecture. *IBM Systems Journal*, 26, 276–292, 1987.
- L.G. Zucker. *Institutional Patterns and Organizations: Culture and Environment*. Ballinger, Cambridge, MA, 3rd edition, 1988.

Part Four Agent-Directed Simulation for Systems Engineering

13

Agent-implemented Experimental Frames for Net-centric Systems Test and Evaluation

Bernard P. Zeigler, Dane Hall, and Manuel Salas

13.1

Introduction

This chapter is concerned with the testing and evaluation of net-centric systems using agent-implemented experimental frames. Such systems are characterized as compositions of nodes distributed over data networks, with information exchanges among nodes playing a role as essential as the processing within nodes. As with their conventional counterparts, the design of net-centric systems should follow a structured life cycle development methodology. In particular, we emphasize a methodology that underlines the role of requirements and the progression from requirements to both implementation and testing of implementations in an integrated manner. As illustrated in Figure 13.1, the development and testing methodology bifurcates the process into two main streams – system development and test suite development – that converge in the system testing phase, which includes testing and evaluation, validation, and verification. The methodology provides a process to transform a requirements specification to a Discrete Event System Specification (DEVS) (Zeigler et al., 2000) representation supporting evaluation and recommendations for a feasible design.

The bifurcated life cycle process combines systems theory (Wymore, 1993), a modeling and simulation (M&S) framework, and model-continuity concepts. The system development includes the definition of requirements, capture of specifications to map to formalized DEVS model components, and creation of a reference master model. Model continuity is exploited to execute the model using the DEVS real-time execution protocol. The test suite development includes execution of test models to run against the system under test whether the system is developed by leveraging models (i.e., via model continuity) or is a system developed from the system engineering requirements existing independently of the executable models. In particular, for net-centric systems, the test models constitute experimental frames (Oren, 2005; Oren and Zeigler, 1979), which can be distributed to observe the behaviors of, and information exchanges among, the nodes of the system. The process is iterative, allowing one to return to modify the reference master DEVS

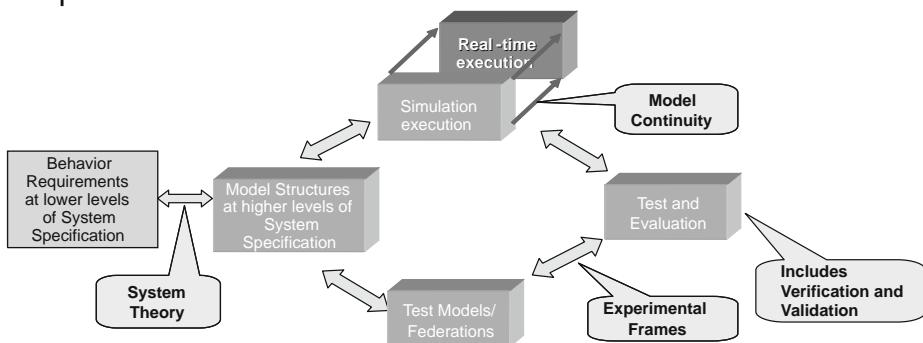


Fig. 13.1 Bifurcated development and testing methodology.

model and the requirements specifications. Model continuity minimizes the artifacts that have to be modified as the process proceeds and increases the coherence of the artifacts across development stages (Hu, 2004; Hu and Zeigler, 2005; Mittal, 2007).

In what follows, we elaborate on the role of requirements in the bifurcated methodology and the progression to development of test models for net-centric system testing. We then show how the test models are implementable using a concept of agent technology, where agents are DEVS models that can be distributed over an infrastructure that supports the DEVS simulation protocol. The appendix discusses a software tool that supports the methodology being discussed. More details can be found in Salas (2008).

13.2

The Need for Verification Requirements

Requirements are written to ensure that the process of design and implementation eventually provides the needed system. The plan for verification should be written with, or in, the requirements. Grady (2006) suggests the following:

Failure to identify how it will be verified that the product satisfies its requirements will often lead to cost, schedule, and performance risks that we gain insight into only late in the program as proven on many programs that the later faults are exposed, the more it costs in dollars and time to correct them. Alternatively, these failures may require so much cost to fix that it is simply not feasible to correct them, forcing acceptance of lower than required performance.

Further, verification of the requirements for components does not necessarily guarantee the next-higher-level item will satisfy its requirements. The Hubble Space Telescope will forever offer an example of this failure (Grady, 2006). This statement points out the need for verification of the integration of elements as the project proceeds toward system acceptance. Each requirement must be verified, and ver-

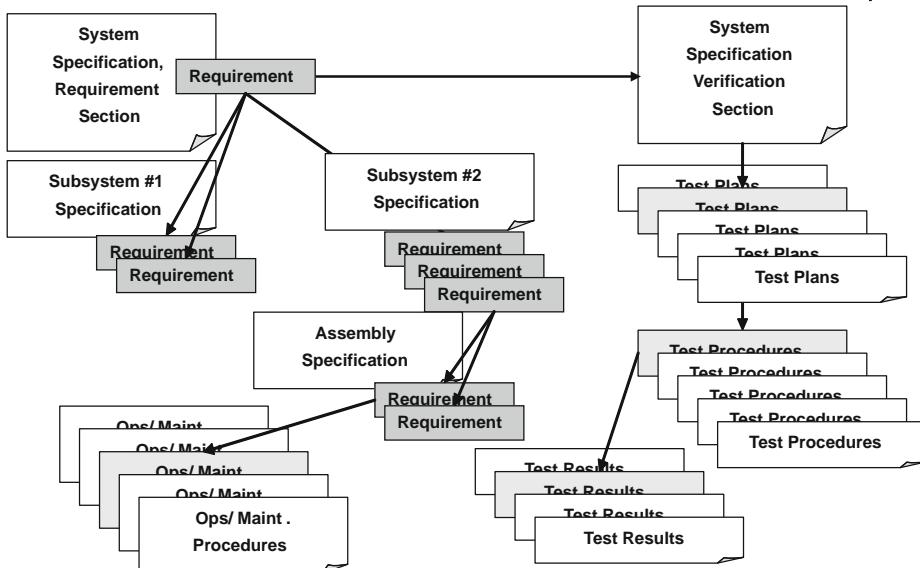


Fig. 13.2 Requirements branch to other documents [taken from Hooks (1994)].

ification may occur throughout the design, development, and operational phases. Any changes to requirements will cause reverification (Hooks, 1994).

Figure 13.2 shows the relationship of system-level specifications to other documents such as test plans and test procedures. Specifications for items have often had separate sections for verification requirements. Verification requirements need to state precisely how the system or subsystem will be verified, whether or not the design completely complies with the requirements.

Grady (2006) encourages having one or more verification requirements for each functional requirement in the requirements section. A better approach would be to have the requirements include the verification requirements, rather than having separate verification requirements. Table 13.1 shows a verification requirement under the behavioral requirement. This table suggests that writing verification requirements together with writing requirements can be an iterative process that can provide greater assurance that the requirements are adequate.

Conditions, behavior, and standards provide a comprehensive description of what is required. These elements map to a formal description of a system, such as is provided by the DEVS formalism. Basic DEVS models completely describe the discrete event behavior of a system. They may be coupled together to form larger models (e.g., assemblies, subsystems, and systems) in a hierarchical fashion.

Table 13.1 Functional requirement with verification requirement divided into conditions, behavior, and standards.

Case	Typical statement	Conditions	Behavior	Standards
Functional requirements	The item shall be capable of entering, residing in, and exiting a mode signaling North-South auto passage as appropriate as defined in <figure> for source and destination states and <table> for transition logic.		Entering, residing in, and exiting a mode signaling auto passage.	Transitions appropriate per <figure> source and destination states <table> transition logic
Verification requirements	The item shall be connected to a system simulator and stimulated sequentially with all input conditions that should trigger mode entry and mode exit and the desired mode shall be entered or exited as appropriate. While in this mode all possible input stimuli other than those that should cause exit shall be executed for periods between 100 and 1000 ms resulting in no mode change. Refer to <figure> and <table> for appropriate entry and exit conditions.	The item shall be connected to a system simulator and stimulated sequentially with all input conditions that should cause desired mode transitions. In desired mode all possible input stimuli other than those that should cause exit shall be executed.	Enter desired mode Exit mode	Enters mode per entry conditions in <fig> and <table>. Stays in mode for periods between 100 and 1000 ms Exits only per exit conditions in <fig> and <table>

13.3

Experimental Frames and System Entity Structures

A requirements specification is among the earliest models of a new system. “...a model is conceived as any physical, mathematical, or logical representation of a system, entity, phenomenon, or process” (5000.59, 1994). To describe the relation of models with systems, a framework for M&S was developed in Zeigler (1976) and is illustrated in Figure 13.3. A comprehensive structure for a requirement and for a requirement’s validation and verification should represent the entities of this framework: experimental frame, source system, model, and simulator, each of which can be described as a system in the underlying systems theory framework. An experimental frame is a system that interacts with a system to be tested under specific conditions to gather data to answer a question about the system. In tests, the model represents the system of interest.

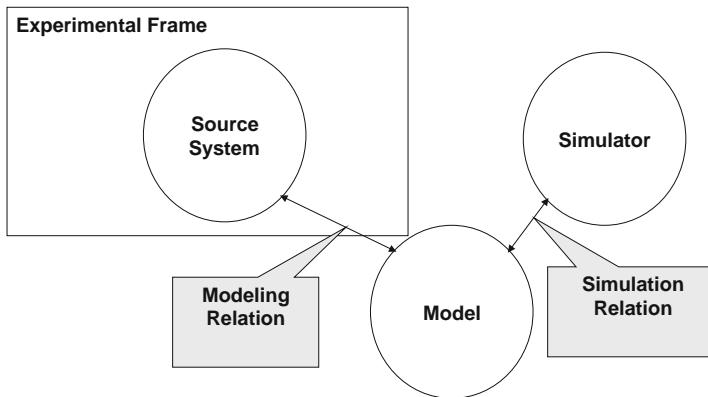


Fig. 13.3 Framework for modeling and simulation.

The simulator executes the model under the experimental frame's conditions to generate its behavior. At the genesis of a new system the only simulator available may be the minds of the designers, and the only model of the future system may be the set of system requirements (specification) under consideration. Grady (2006) strongly warns against failing to have good verification requirements corresponding to all requirements. He recommends appointing a lone, principal engineer, who should ensure that all of the verification requirements are prepared together with the item requirements. This emphasis on verification requirements emphasizes the importance of the experimental frame, which specifies the conditions under which the model will be exercised.

In the M&S framework (Zeigler, 1976), an experimental frame is the operational formulation of the objectives that motivate an M&S project. A frame is realized as a system that interacts with the system of interest to obtain the data of interest under specified conditions. Whether used in a simulation or not, an experimental frame is an important tool for both build-time verification that the system is being built to the specification and for design-time validation that the requirements for the system can be met and represent a system that will meet the need.

Experimental frames may be implemented in M&S in various ways. One way is to divide the work of the experimental frame among a generator, a model that generates inputs to the system, and another component, a transducer, which observes inputs to the system from the generator and observes and analyzes the system's outputs. A transducer can also inform an acceptor that determines whether the desired experimental conditions are met. Figure 13.4 shows such an experimental frame designed to measure the "what and how well" of the requirements for the system. The experimental frame and a representation of the system (model) are operated by the simulator to provide a prediction of how well the model meets requirements for the system.

Table 13.2 shows how the elements of the system formalism and M&S framework map to requirements and their conditions, behaviors, and standards. Although components of the experimental frame (being models themselves) can be

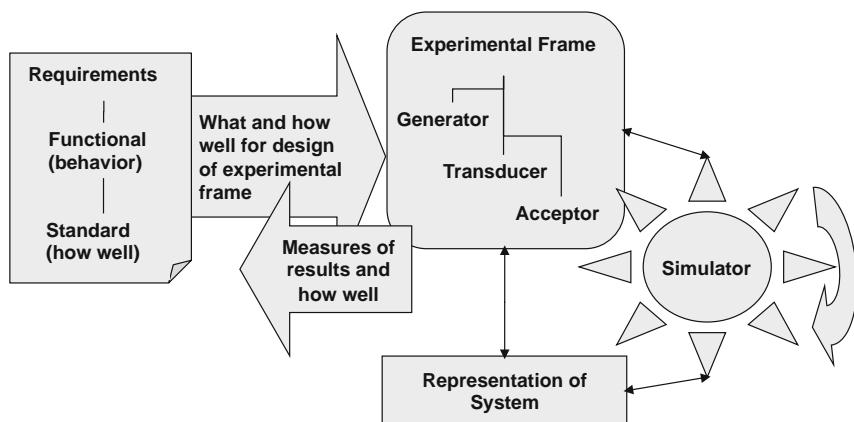


Fig. 13.4 Experimental frame and model of system driven by simulator.

Table 13.2 Mapping system formalism and M&S framework to conditions, behaviors, and standards.

Elements of system formalism and the M&S framework	Requirement element (i.e., conditions, behaviors, standards, and infrastructure)
Experiment Frame	Conditions and Standards
Generator – sends input values to the model	Conditions
Transducer – receives outputs of the model and receives copies of the inputs to the model; using these inputs provides calculations to the acceptor	Standards: Monitors inputs to the model and outputs of the model for the use of the acceptor.
Acceptor – monitors the experiment to see whether the desired experimental conditions are met	Standards
Source system	Realizes the set of all system requirements
Model	Behavior
Time Base	Behavior – time-indexed input/output data
Set of input values	– Contain inputs that the model must receive
Set of states	– Represent modes and states
Set of output values	– Contain outputs that the model must send
Transition function	Determines state transitions induced by inputs and passage of time
Output function	Behavior – generates outputs
Simulator	Infrastructure for executing the V&V

specified by a DEVS, the table is filled in from the perspective of the system representation being validated or verified.

The generator, transducer, and acceptor shown in Table 13.2 realize an experimental frame specification, which includes the following dimensions:

- *Input stimuli*: specification of the class of admissible input time-dependent stimuli. This is the class from which individual samples will be drawn and injected into the model or system under test for particular experiments.
- *Control*: specification of the conditions under which the model or system will be initialized, continued under examination, and terminated.
- *Metrics*: specification of the data summarization functions and the measures to be employed to provide quantitative or qualitative measures of the input/output behavior of the model. Examples of such metrics are performance indices, goodness-of-fit criteria, and error accuracy bound.
- *analysis*: specification of means by which the results of data collection in the frame will be analyzed to arrived at final conclusions. The data collected in a frame consist of pairs of input/output time functions.

An experimental frame can be viewed as a system that interacts with the system under test (SUT) to obtain the data of interest under specified conditions. As indicated earlier, a frame typically has three types of components (as shown in Figure 13.5a): a generator that generates input segments to the system; an acceptor that monitors an experiment to see the desired experimental conditions are met; and a transducer that observes and analyzes the system output segments.

Figure 13.5b illustrates a simple, but ubiquitous, pattern for experimental frames that measures typical job processing performance metrics, such as those that relate to round trip time and throughput. Illustrated in the web context, a generator produces service request messages at a given rate. The time that has elapsed between the sending of a request and its return from a server is the round trip time. A transducer notes the departures and arrivals of requests allowing it to compute the average round trip time and other related statistics, as well as the throughput and unsatisfied (or lost) requests.

When combined with the System Entity Structure (SES) ontology framework (Zeigler and Hammonds, 2007), the experimental frame provides a means to represent complete requirements, that is, it answers the question, what would a requirement in which all relevant features have been addressed look like? Figure 13.6 presents an SES for a behavioral requirement with verification (experiment) conditions and postconditions (standard for acceptable verification). This SES exploits a key feature of SES: specialization. Children of specializations are entities representing variants of their parents. The items with tildes (~) under the choices for metrics are variables, for example, type of units, and threshold (minimum acceptable) performance levels and objective (maximum desired) levels expressed in the units specified.

Since it can capture all the aspects needed for completeness, we propose that the experimental frame concept is a desirable way to express requirements. Such an SES can be expressed as an XML Schema using a tool such as the SESBuilder. Once

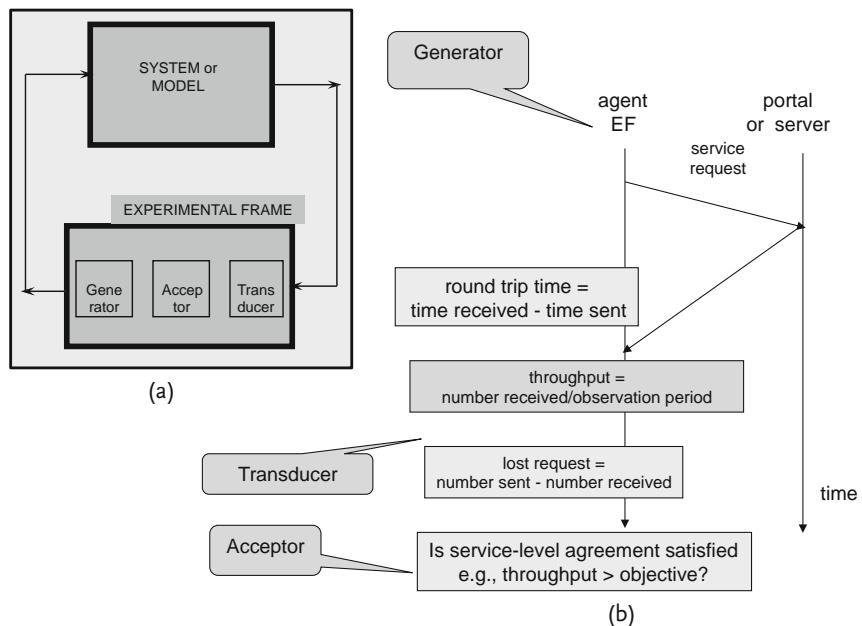


Fig. 13.5 Experimental frame and components.

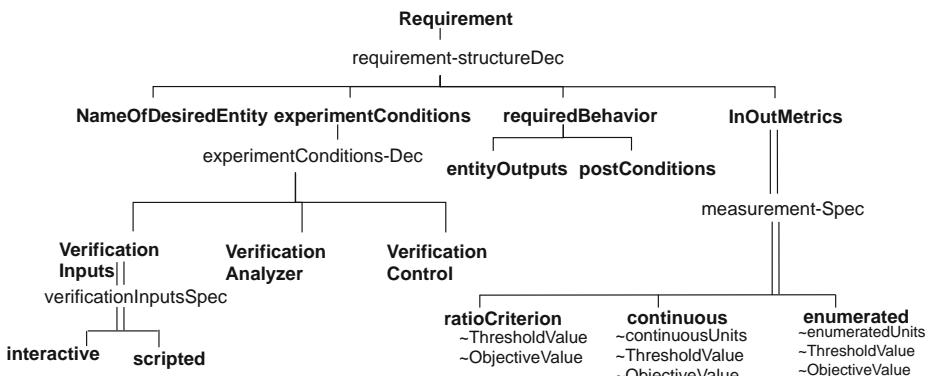


Fig. 13.6 SES for verification conditions and standards for verification.

adopted and standardized, this form can be widely promulgated within an organization or among a wider community of interest. Modern information-technology-based systems need to be represented by multiple, top-level requirements. These should be characterized by their respective experimental frames. Such a concept could enable vendors to describe their products' performance and interface specifications on the World Wide Web per the standard XML schema, bringing vendors and system architects together more effectively and efficiently. This marketplace would also improve the expression of requirements through the examples that requirement developers would find on the web. The greater effort and rigor needed

to comprehensively express requirements in this manner would be encouraged by such a marketplace and facilitated by being able to learn from requirements posted by others. Another advantage of this approach is that due to the “specialization” feature of the SES, a requirements SES for a generic product should facilitate designing product lines for it.

Since experimental frames are realizable as test systems and/or tools, a database of requirements expressed as experimental frames should be easily related to the set of test items required for a project. This should improve estimating of verification costs and the verification of requirements. Expressing requirements and experimental frames as XML instances complying with standardized XML schemata would mitigate against falling into the trap of unwittingly embedding specific solutions within the requirements (perhaps overconstraining the designer’s freedom). For this purpose software tools that support higher levels of validation and analysis of XML instances against standardized XML schemata should be developed.

13.4

Decomposition and Design of System Architecture

Requirements are derived through engineering processes that include design and analysis, trade studies, concept development, and other activities such as prototyping (Hooks, 1994). The difficulty with this major activity is shown in the following quote from Grady (2006):

Many organizations find that they fail to develop the requirements needed by the design community in a timely way. They keep repeating the same cycle on each program and fail to understand their problem. Commonly, the managers in these organizations express this problem as, we have difficulty flowing down system requirements to the designers. [T]he flowdown strategy is only good for some specialty engineering (e.g., reliability engineering and weight management) and environmental design constraints. It is not a good strategy for interface and performance requirements. Performance requirements are best exposed and defined through the application of a structured process for exposing needed functionality and allocation of the exposed functionality to things in the architecture.

Grady also discusses why flowing requirements down is difficult and often needs simulation. “There are many requirements that will not yield directly to the allocation process, however. And, these are commonly the most difficult requirements to quantify. Given that we have a requirement for aircraft maximum airspeed, how shall we determine the engine thrust needed? The mathematical relationships between parameters is complex and cannot accurately be resolved on the back of an envelope. Commonly, we have to apply some form of computer simulation to try various combinations of values working toward identification of a good and achievable mix of values.”

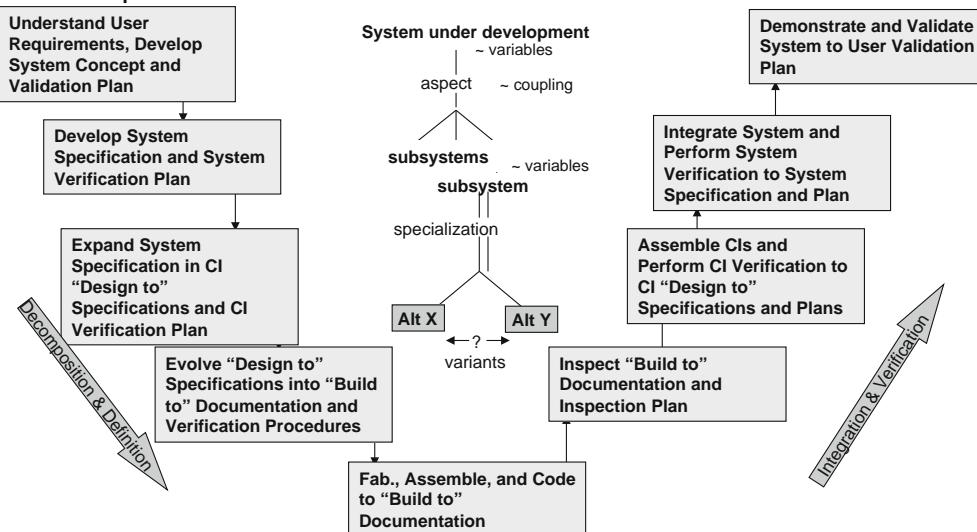


Fig. 13.7 SES-structured M&S superimposed on the center of the Vee model of systems engineering.

Figure 13.7 provides context for requirement decomposition and design. Activities step down and then up as a V-like structure. Decomposition into lower levels of detail and design is at left. At the right of the V, integration and verification step from component up to system level. While the Vee Model of systems engineering has been advocated by others, we augment it with the above Experimental Frame and SES concepts. In the center of the V is a SES representing the concept of a system under development built from smaller component systems. Such a hierarchical composition can employ levels such as assemblies, modules, packages, and so on.

Forsberg et al. (2005) suggest a Decomposition Analysis and Resolution Process as a framework for proactive requirements management to be applied at each level of the overall process of decomposing and defining requirements to provide the design and verification specifications for the next lower level, down to the lowest level of documentation.

Figure 13.8 shows the activities and results of Decomposition Analysis and Resolution.

Steps 2, 3, and 4 of Decomposition Analysis and Resolution are prime activities for the use and reuse of M&S. At each point in progressing down the left side of the V (refer to Figure 13.7), system engineers should discover and document how to verify later during the work leading up the right side of the V that the solution is being implemented as specified. Also, the project team should be validating that the solution will perform as required (i.e., the right solution being built). M&S can contribute by building a tree of experimental frame leafs for M&S as decomposition proceeds down the hierarchical levels. With the use of real time simulators,

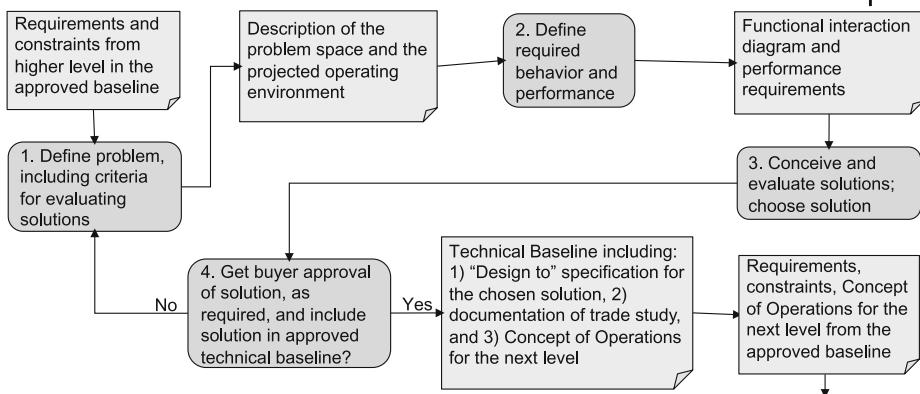


Fig. 13.8 Requirements decomposition analysis and resolution.

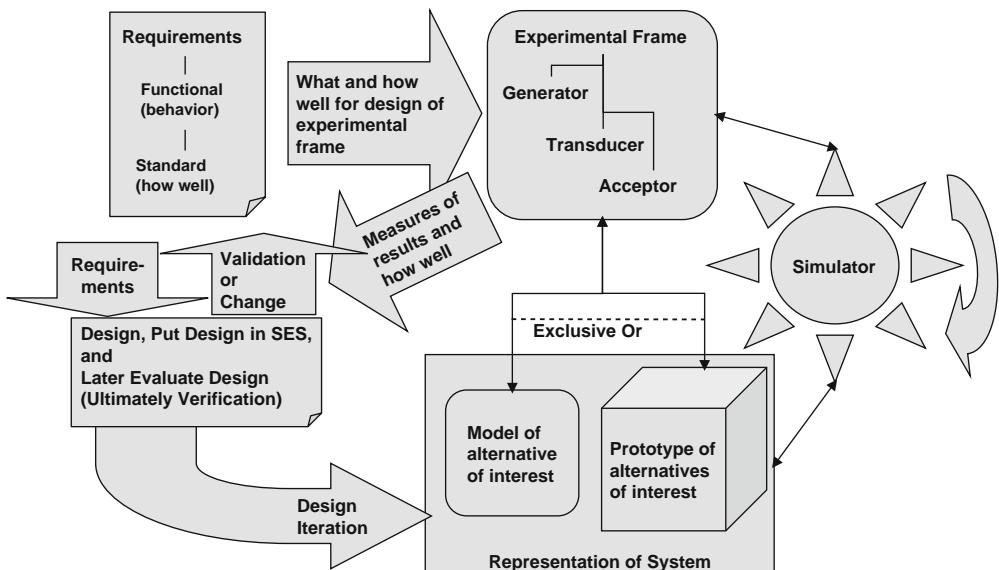


Fig. 13.9 Requirements management, validation, design, prototyping, and verification.

physical implementations can gradually replace models in the simulations. So in addition to being used for the validation of requirements and for choosing between alternatives on the decomposition and definition side of the V, the experimental frames may be reused on the right side of the V in stimulating and measuring the response of the components, assemblies, subsystems, and finally the system. Dual use of an experimental frame is shown in Figure 13.9, which shows M&S used in an integral way with requirement management, validation, design, prototyping, and verification.

So the process of examining the breakdown of the desired system into the best alternative subsystems, assemblies, and components can be viewed as resulting in

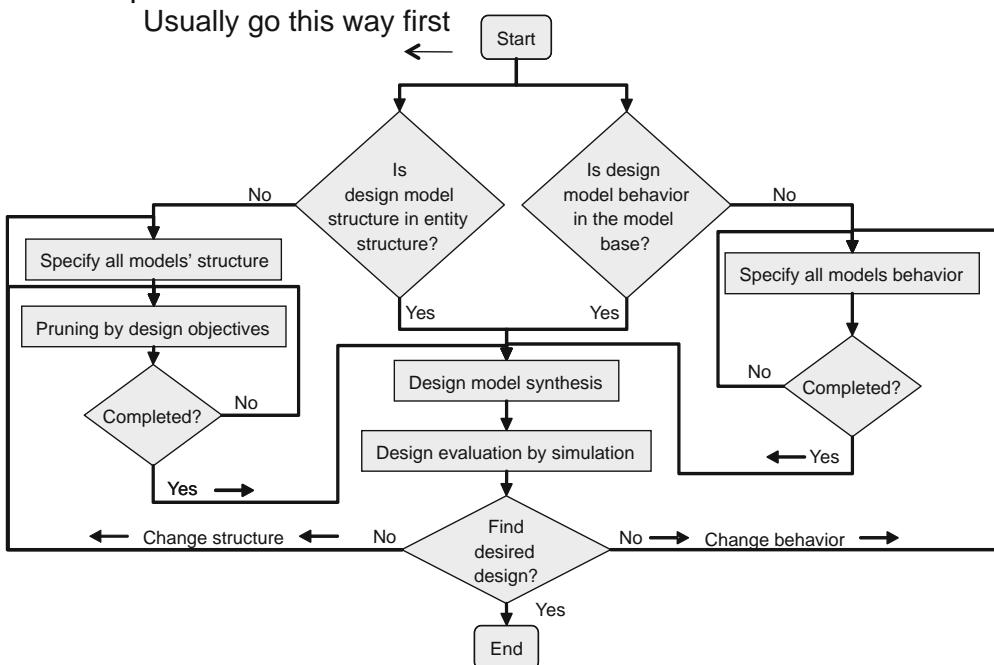


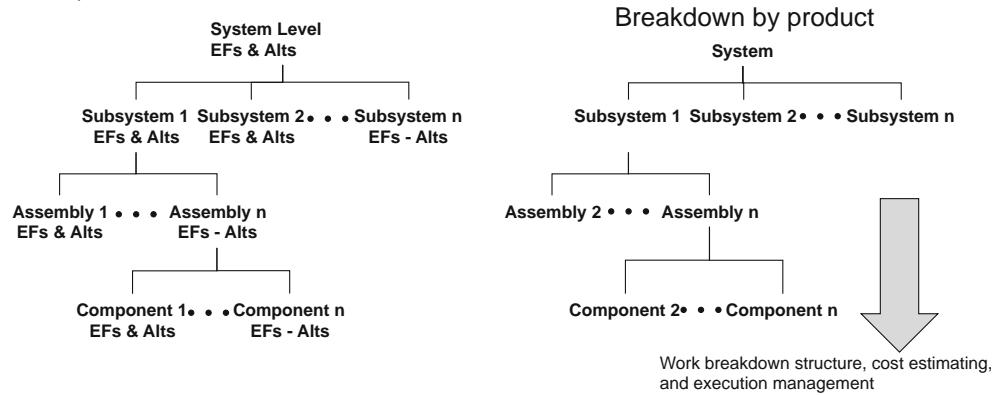
Fig. 13.10 System entity structure and a simulation ModelBase.

a tree of experimental frame – alternative choice pairs. The ability to express these pairs and to select best alternatives and know the cost and schedule to realize them is a measure of subject matter expertise. The SES and the consistent development of models can provide a significant part of a structured approach to capturing this expertise. A scheme that uses the SES for managing a collection of models for simulating and evaluating alternative configurations of systems as shown in Figure 13.10.

Special expertise is required to derive requirements for a set of component systems that will work together to satisfy a given requirement. For example, posed with a requirement to transport a 10000 pound payload 5000 miles away in 2 hours, expert designers would have to be able to derive requirements for components to provide propulsion, control, and navigation. In the case of the navigation system for example, the requirements for navigation would be transformed to an experimental frame specific to that subsystem, and navigation experts and/or navigation system vendors would contribute models and/or specifications representative of the state of the art. Figure 13.11 illustrates the hierarchy of experimental frames and alternatives juxtaposed with the product breakdown and work breakdown.

We now propose combining requirement management via M&S with the concept of web services facilitated by open standards. Consider a scenario in which already developed components are described with requirements written to an open standard for requirements. Then, with web service-enabled models of the

EFs = Experiment Frames



The EF and Alternatives (Alts) at the system level are used in M&S for selecting the configuration at each level. The product breakdown structure is directly related to the breakdown into EF and Alt groups.

Fig. 13.11 M&S for generating product breakdown structure.

components written to an open standard and a standard simulation protocol (Ziegler, 2008), designers could search and discover such models in an open registry. Vendors could have their products modeled in a ready-to-simulate form in a web-catalyzed marketplace. Vendors could register web service enabled models of their products in an on line registry, such as registries conforming to the open standard for Universal Description Discovery and Integration (UDDI). To facilitate registering their models, vendors would describe their web service enabled models using Web Service Description Language (WSDL) using the XML schema discussed above as the schema for expressing the specifications that their products meet. Models can use these WSDLs to develop simulations. Also under this concept, vendor web services would upon suitable request, return the specification for their product in XML, including interface requirements according to an open standard (i.e., the XML schema for requirements/experimental frame). Interface requirements are closely related to coupling of a representation (model) of a system of interest with an experimental frame, which would be required to demonstrate what the product will do and accomplish. This system of requirements and models should be supported by a standard lexicon or ontology. For example, ISO 9001, “Quality management systems – Requirements” is accompanied by ISO 9000, “Quality management systems – Fundamentals and vocabulary”.

This system should extend the idea of SES-enabled model base discussed above with Figure 13.12. Additionally, this system should capitalize on the technology that provides service oriented distributed M&S over the World Wide Web (Mittal, 2007; Mittal et al., 2009). This technology would enable designers to federate suitable vendor models with other models to validate the performance of a proposed architecture under consideration.

13.5

Employing Agents in M&S-Based Design, Verification and Validation

In this article, the term “agent” (or “mobile agent”) refers to a discrete thread of computation that is deployable in distributed form over computer networks where it interacts with local computations and communicate/coordinate with other such agents. The modular nature of software objects together with their behavior and interaction with other objects, led to the concept of agents which embodied increased autonomy and self-determination. A wide variety of agent types exists in large part determined by the variety and sophistication of their processing capacities – ranging from agents that simply gather information on packet traffic in a network to logic-based entities with elaborate reasoning capacity and authority to make decisions. The step from agents to their aggregates is natural, thus leading to the concept of multi-agent systems or societies of agents, especially in the realm of M&S. Distributed Artificial Intelligence systems (Gasser et al., 1987) integrate concepts from concurrent programming (Agha and Hewitt, 1985) and knowledge representation to coordinate agent ensembles in distributed problem solving. Cognitive agents represent beliefs and intentions and flexible mechanisms for communication among agents (Rao and George, 1987). Reactive agents employ procedural reasoning and automatic control theory to interact with their environments. Hybrid agents combine higher level goal-directed planning with lower level automatic control to integrate deliberative and reactive processes in real time (Firby, 1992). Endomorphic agent concepts, first defined in Zeigler (1990), provide a framework for including the distributed knowledge that arises from providing agents with internal models of the environment, themselves, and other agents. We will employ these concepts below.

Given this review of agent concepts and technology, we address the question of how to effectively employ agents in the M&S approach to requirements-based design, verification and validation of systems. There are various roles that agents could take in supporting the activities summarized in Figure 13.9. Here we focus on the possibility of employing agents to implement the experimental frames derived from requirements for a distributed system. Such agents will be expressed as DEVS models that may implement experimental frame components, may communicate and/or coordinate with each other using underlying DEVS-based distributed simulation infrastructure and may include endomorphic models of the system under test. We now discuss these dimensions of DEVS agent technology.

Starting from the top, the composition of a system and an experimental frame is represented in Figure 13.12a. Using a natural language specification (Zeigler and Hammonds, 2007), we can express this composition as:

From the structure perspective, *EFS* is made of *EF* and *System*.

For purposes of observing the interactive behavior of components, we can indicate that a *System* consists of components, each of which may be observable or not observable, as illustrated in Figure 13.12b and expressed as:

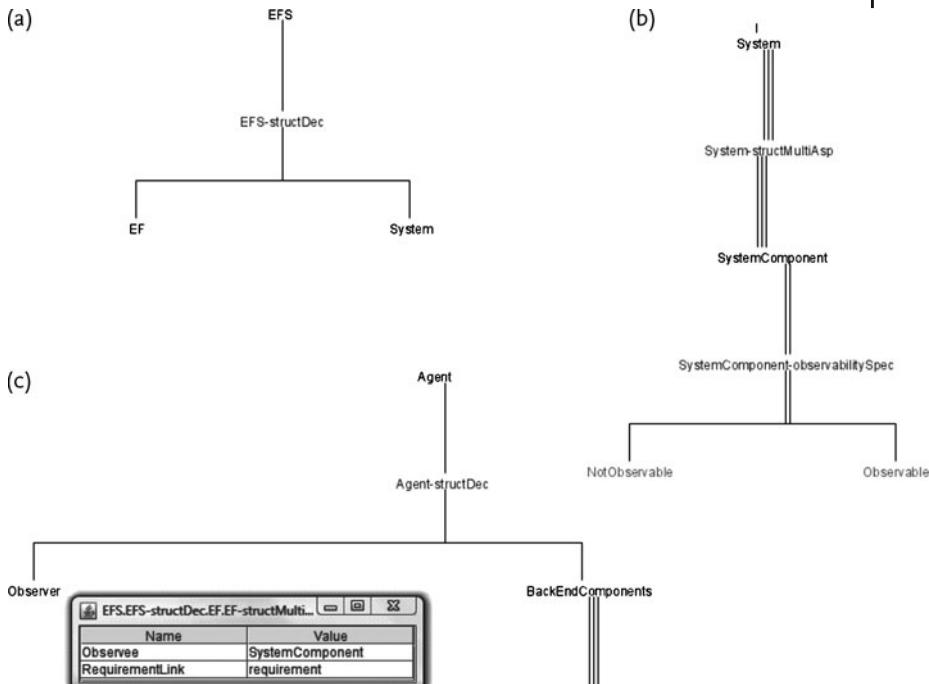


Fig. 13.12 Experimental frame, system, and agents.

- (1) From the observability perspective, a *System* is made of more than one *System Component*. (2) A *System Component* can be *Observable* or *Not Observable* in observability.

A system component is observable if it accommodates attachment of an agent to intercept its communications with other components. Indeed, an essential constraint on experimentation in net-centric systems is that components may not support agent attachment for various reasons including security, proprietary concerns, machine loading, and others.

On the other side, as illustrated in Figure 13.12(c), an experimental frame in this context will be composed of agents that carry out its distributed functionality. In the following description, each agent has an assigned component to observe and a link that traces back to one or more requirements (as described in Figure 13.11) from which its role in the testing and evaluation was derived. An agent is composed of an observer module and one or more “backend” components such as generators, acceptors and transducers that implement its functionality:

- From the structure perspective, EF is made of more than one Agent;
- From the structure perspective, Agent is made of Observer and BackEnd-Components;
- An Observer has an Observee and a RequirementLink;
- The range of Observer’s Observee is SystemComponent;
- The range of Observer’s RequirementLink is requirement;

- From the structure perspective, BackEndComponents are made of more than one BackEndComponent;
- A BackEndComponent can be Generator, Acceptor, or Transducer in function.

From this high top-level view we proceed to drill down to uncover some critical details in making the concept of agent-implemented frames operational.

13.6

Experimental Frame Concepts for Agent Implementation

In the realm of service-oriented architecture for web services, mission threads for a proposed system of services are intended to encapsulate how well the system's capabilities will be used in real world situations. The capability to measure effectiveness of missions requires the ability to execute such mission threads in operational environments, both live and virtual. For testing mission threads, Experimental Frame concepts offer an approach to enable simulated and real-time observation of participant system information exchanges and processing of acquired data to allow mission effectiveness to be assessed. Relevant Measures of Performance (MOPs) include quality of shared situational awareness, quality and timeliness of information, and extent and effectiveness of collaboration. Measures of Effectiveness (MOEs) are measures of the desired benefits such as increase in combat power, increase in decision-making capability, and increase in speed of command. Such metrics must be modeled with mathematical precision so as to be amenable to collection and computation with minimally intrusive test infrastructure to support rigorous, repeatable and consistent testing and evaluation (T&E).

An example of a joint mission thread is the Theater Strategic Head Quarter Planning at the Combatant Command Level. As illustrated in Figure 13.13, this thread starts when the Combatant Command HQ receives a mission which is delegated to HQ Staff. The staff initiates the Mission Analysis Process which returns outputs and products to the Commander. Then the Joint planning group and the development teams develop courses of action and perform effects analysis, respectively. The thread ends with the issue of operational orders. The MOP of interest might include various ways of measuring extent and effectiveness of collaboration, while the MOE might be the increase in speed of command. These measures might be collected in assessing the value added of a collaboration support system in an operationally realistic setting. An informally presented mission thread can be regarded as a template for specifying a large family of instances. As illustrated in Figure 13.13, such instances can vary in several dimensions, including the objectives of interest (including the desired MOP and MOE), the type of application, the participants involved, and the operational environments in which testing will occur. Furthermore, mission threads can be nested, so that for example, Mission Analysis is a sub-thread that is executed within the Theater Planning thread.

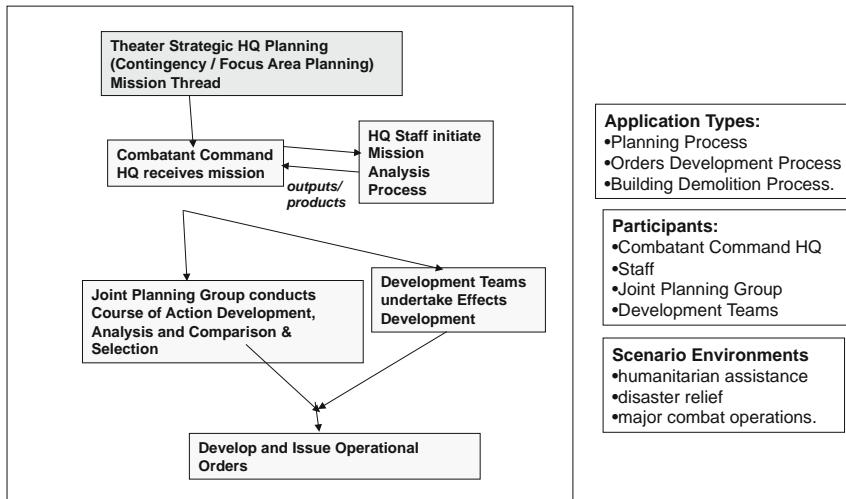


Fig. 13.13 Joint mission thread and its dimensions for variation.

The upper part of Figure 13.14 illustrates the process for deriving the types of experimental frame components and their parameter values. The test event objectives, as embodied in the MOEs and/or MOPs of interest in the mission thread to be tested, determine the experimental frame – this is a specification of the experimental conditions to be established in the network environment to support the test. The experimental frame consists of a selection of observers – the agents that listen to web service requests, generators that can simulate user actions or inject background web traffic, transducers that gather statistics and acceptors that check for conditions that must pertain for valid experimentation to proceed. Not all of these components need be present for a test – for example, generators would be absent in a test in which only live users participate without virtual counterparts.

Figure 13.14, shows how an experimental frame sets up the conditions for testing the effectiveness of a service oriented architecture (SOA) for mission threads involving multiple SOA services. An instance of collaboration is modeled as a coupled model in DEVS – a model which specifies components and how they interact by being coupled together by specified output to input connections. In a model of collaboration, the components are participants in the collaboration and couplings represent the possible information exchanges that can occur among them. This formulation offers an approach to capturing and implementing mission threads in the form of test model federations that can be deployed in a net-centric environment. The experimental frame, as derived from the test event objectives, determines which participants in the collaboration will be observed, the types of web service requests that will be listened to, and the particular field values in the request Simple Object Access Protocol (SOAP) messages that will be extracted and examined. The Web Service Description Language (WSDL) structures associated with such requests provide the basis for accessing these messages and their contents.

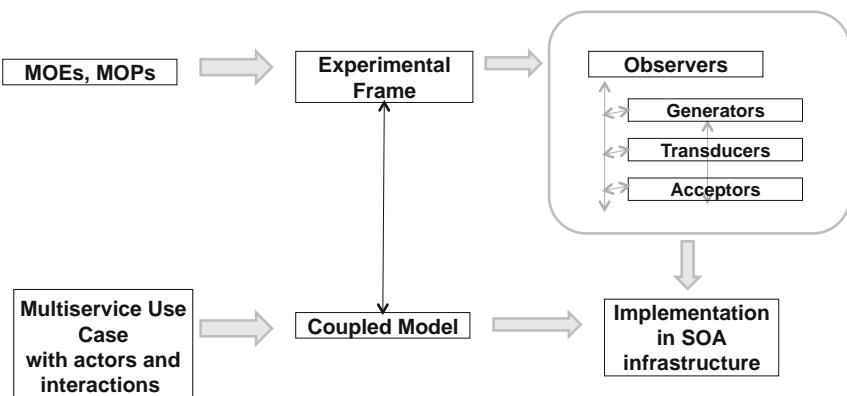


Fig. 13.14 DEVS concepts for mission thread-based testing.

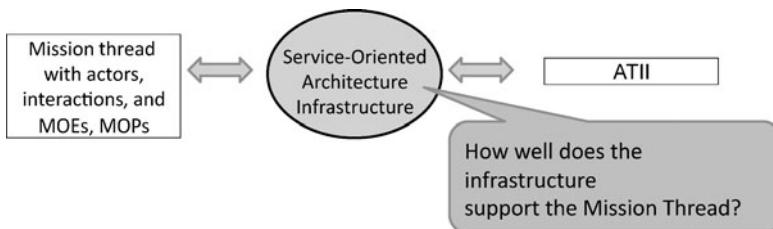


Fig. 13.15 Mission thread-based testing the net-centric infrastructure.

As shown in Figure 13.15, the design of an agent-implemented test infrastructure is aimed at the problem of assessing how well a service-oriented infrastructure supports the collaboration and information exchange requirements of a mission thread. Figure 13.15 depicts how a net-centric infrastructure offers a way to deploy the participants in a mission thread together with network and web services that allow them to collaborate to achieve the mission objectives. Formulation of a mission thread instance as a coupled model allows us to provide rigorous methods for realizing the use case within the SOA infrastructure. At the same time, the MOEs and MOPs that have been formulated for assessing the outcome of the mission execution are translated into an appropriate experimental frame, with observers of the participant activities and message exchanges as well as the generators, transducers and acceptors as discussed above. However in the context of net-centric operation, the distributed nature of the execution will require the experimental frame to be distributed as well. This distribution is implemented by deploying agents and their associated components among a selection of nodes of the network that is specified by the tester.

13.7

Agent-Implemented Experimental Frames

A DEVS distributed federation is a DEVS coupled model whose components reside on different network nodes and whose coupling is implemented through middleware connectivity characteristic of the environment, for example, SOAP for Global Information Grid (GIG)/SOA. The federation models are executed by DEVS simulator nodes that provide the time and data exchange coordination as specified in the DEVS abstract simulator protocol.

As discussed earlier, in the general concept of experimental frame (EF), the generator sends inputs to the system under test (SUT), the transducer collects SUT outputs and develops statistical summaries, and the acceptor monitors SUT observables making decisions about continuation or termination of the experiment (Zeigler et al., 2000). Since the application is composed of service components, the EF is distributed among application components, as illustrated in Figure 13.6. Each component may be coupled to an EF consisting of some subset of generator, acceptor, and transducer components. As mentioned, in addition an observer couples the EF to the component (web server/client). We refer to the DEVS model that consists of the observer and EF as a test agent.

Net-centric Service Oriented Architecture (SOA) provides a currently relevant technologically feasible realization of the concept. As discussed earlier, the DEVS/SOA infrastructure enables DEVS models, and test agents in particular, to be deployed to the network nodes of interest. As illustrated in Figure 13.16, in this incarnation, the coupling between observer and the service is facilitated by the interface provided by the Web Service Description Language (WSDL). The network inputs sent by EF generators are SOAP messages sent to other EFs as destinations; transducers record the arrival of messages and extract the data in their fields, while acceptors decide on whether the gathered data indicates continuation or termination is in order (Mittal, 2007).

Since EFs are implemented as DEVS models, distributed EFs are implemented as DEVS models, or agents as we have called them, residing on network nodes. Such a federation, illustrated in Figure 13.17, consists of DEVS simulators executing on web servers on the nodes exchanging messages and obeying time relationships under the rules contained within their hosted DEVS models.

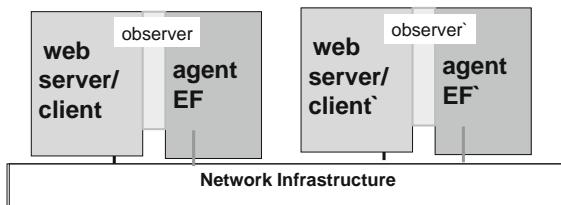


Fig. 13.16 Deploying experimental frame agents and observers.

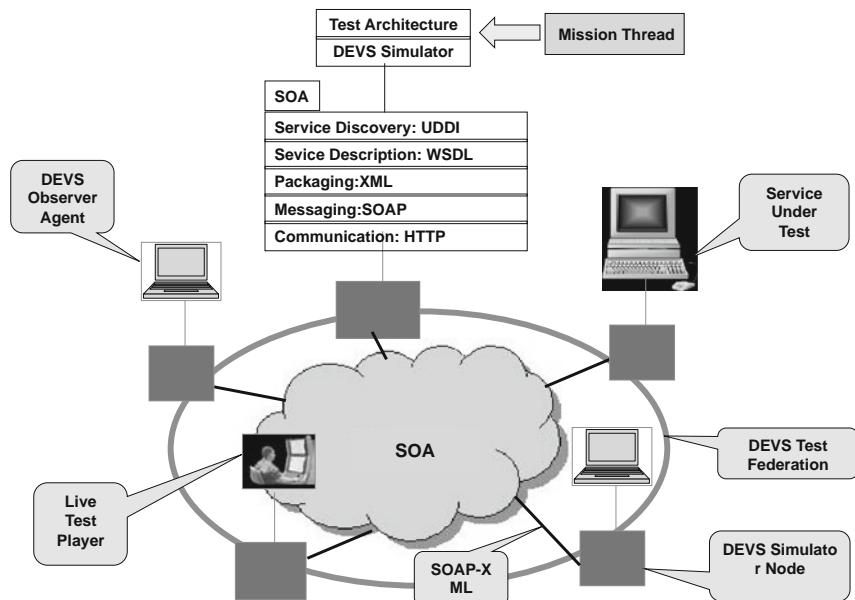


Fig. 13.17 DEVS test federation in GIG/SOA environment.

13.8

DEVS/SOA: Net-Centric Execution Using Simulation Service

The fundamental concept of web services is to integrate software applications as services. Web services allow the applications to communicate with other applications using open standards. We are offering DEVS-based simulators as a web service, and they must have these standard technologies: communication protocol (Simple Object Access Protocol, SOAP), service description (Web Service Description Language, WSDL), and service discovery (Universal Description Discovery and Integration, UDDI).

The complete setup requires one or more servers that are capable of running a DEVS Simulation Service. The capability to run the simulation service is provided by the server side design of DEVS Simulation protocol supported by the DEVS-JAVA. The Simulation Service framework has two layers. The top-layer is the user coordination layer that oversees the lower layer. The lower layer is the true simulation service layer that executes the DEVS simulation protocol as a Service (Mittal, 2007).

13.8.1

Automation of Agent Attachment to System Components

Having reviewed the underlying infrastructure, we return to the top level view of Figure 13.14 to consider how to automate the attachment of agents to observable

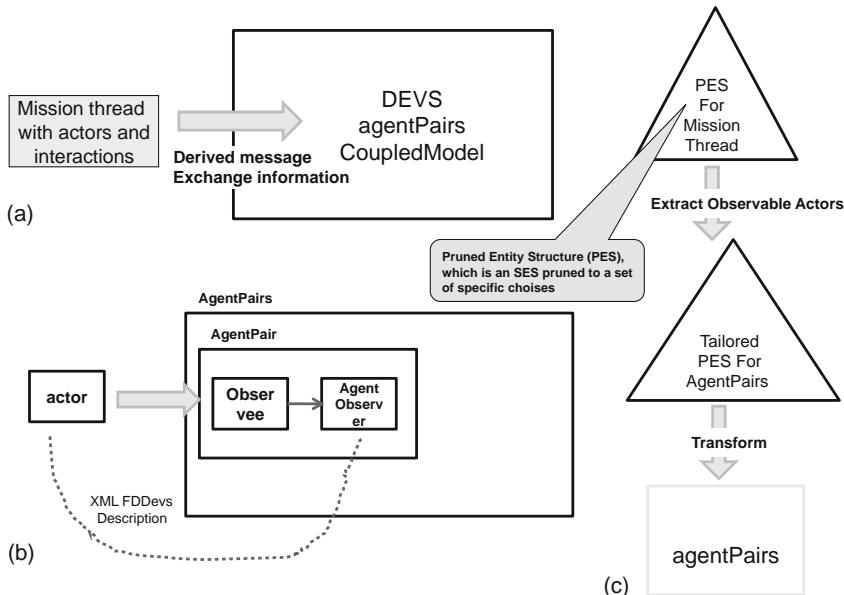


Fig. 13.18 Mapping mission thread coupled models to agent pair models.

components and their deployment in a net-centric environment. We continue by limiting our discussion to the case of mission threads as illustrated in Figure 13.13 and refer to Figure 13.18 to frame the problem of automating the attachment of agents to observable components. In Figure 13.18a, we are given a DEVS coupled model of a mission thread and consider mapping such a specification into a coupled model consisting of agents as defined in Figure 13.18. Such a mapping can include information concerning the message exchange paths among actors in the mission thread for runtime use by the observing agents.

To formulate the process that will attach agents to clients (as in Figure 13.16), we introduce the abstraction of an agentPair as shown in Figure 13.18b. Here the DEVS model for an observable actor in the mission thread is called an Observee. It is to be interfaced to an Observer to form an agentPair. These pairs form the components of the coupled model, agentPairs, a composition that will represent the final net-centric environment, as in Figure 13.17. Information concerning the structure and behavior of the actor can be extracted from its model to help its agent in its observation. Information of this kind, at the both the system and component model levels, endows the agents with endomorphic models of the system (mission thread) they are tasked with observing.

To construct agentPairs we write the SES specification:

From a structure perspective, agentPairs are made of more than one agentPair

From a message perspective, the agentPair is made of observer and observee

which is illustrated in Figure 13.19.

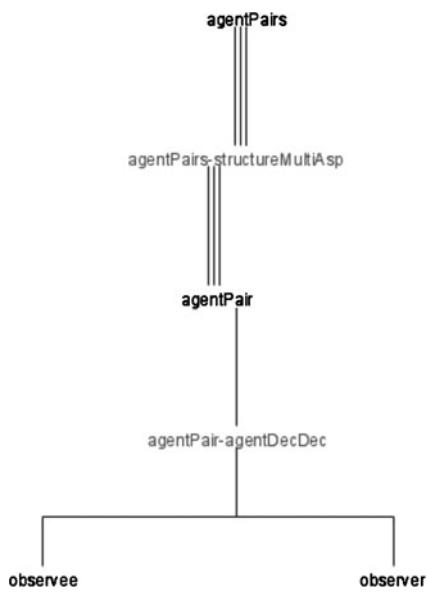


Fig. 13.19 SES for agent pair models.

13.8.2

DEVS-Agent Communications/Coordination

We indicated earlier that the mapping in Figure 13.18 can include information concerning the message exchange paths among actors in the mission thread for runtime use by the observing agents, endowing the agents with endomorphic models of the mission thread under observation.

We use a greatly simplified example of a mission thread to illustrate how this can be implemented.

Example 13.1 Mission Thread

In this thread, a truncated version of a much longer one, a commander, MajSmith, posts a request for intelligence on a target he has been assigned to attack. The request is posted using a web service hosted at a Net-Enabled Command Capability server farm, NECC, and relayed to an intelligence unit, IntelCell, which collects the needed data and returns its estimate to MajSmith via web services of the NECC.

From a message perspective, the MissionThread is made of MajSmith, IntelCell, and NECC

From the message perspective, IntelCell sends threatEstimate to NECC

From the message perspective, NECC sends threatEstimatePost to MajSmith
From the message perspective, MajSmith sends ThreatEstReqPost to the NECC

From the message perspective, NECC sends IntelReq to IntelCell

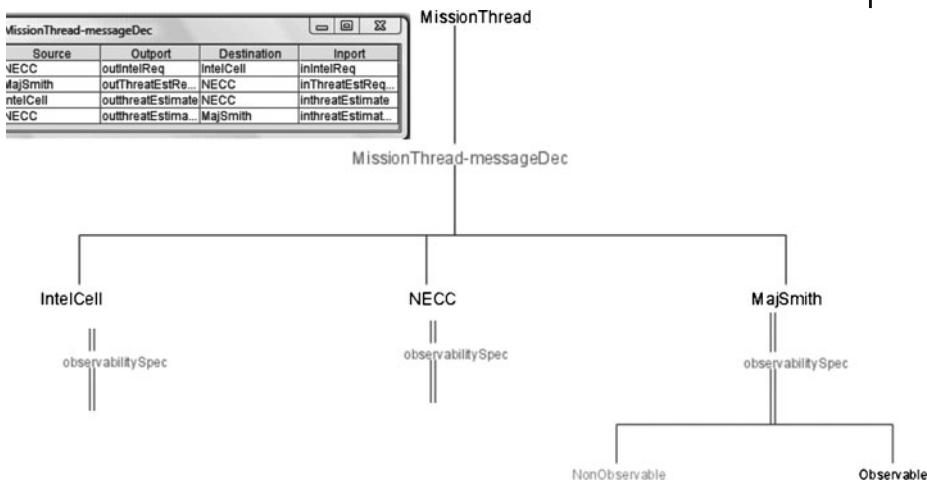


Fig. 13.20 SES for observability designations.

To describe the possibility of observing these actors we state that:

- MajSmith can be Observable or NonObservable in observability
- NECC is like MajSmith in observability
- IntelCell is like MajSmith in observability

This description is illustrated in Figure 13.20.

Consider the assumption that we can only attach DEVS agents to clients of the web services at the user sites not at the servers. Then only messages sent and received by the MajSmith and IntellCell can be observed in actual testing. This constraint can be expressed by selecting Observable from observabilitySpec for both MajSmith and IntellCell while selecting NonObservable for the specialization under NECC. Using the transformation process (Zeigler and Hammonds, 2007) this pruned entity structure is transformed to a DEVS coupled model of the example mission thread illustrated in Figure 13.21. The component names, Observable MajSmith, NonObservable NECC, and Observable IntellCell carry the information about observability that was determined in the pruning process.

While the gray lines in Figure 13.21 indicate couplings prescribed by the SES of Figure 13.20, the arrows are superimposed to indicate web service requests that can be observed by agents to be coupled to the observable components. For example, when MajSmith posts an intelligence request an agent connected to the attached client can view this request. Similarly, when IntellCell requests the details of the posting via a web service, this request can be observed by the associated agent. However, because the NECC server is not observable, agents cannot directly watch for arrivals or departures of request messages at this server. This means that in order to verify that MajSmith's request for intelligence actually reached IntellCell, we have to make it possible for MajSmith's agent to inform IntellCell's agent that

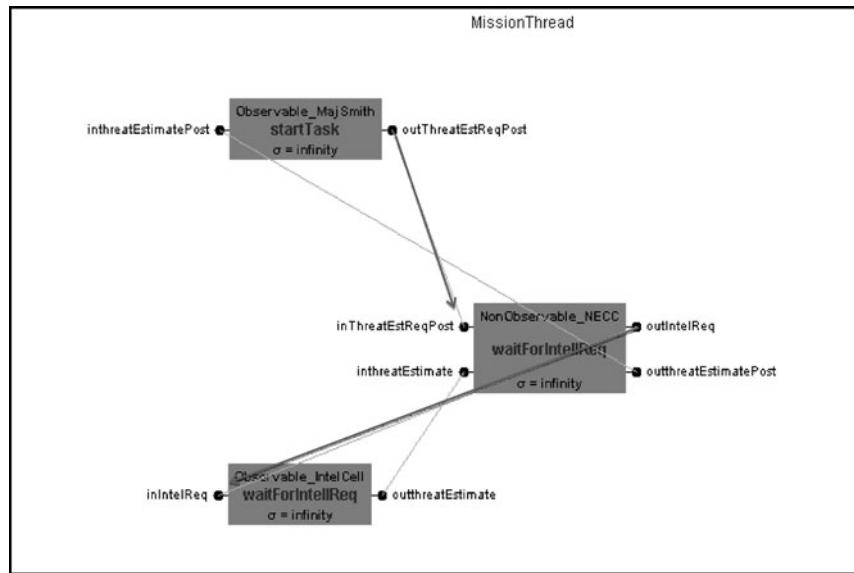


Fig. 13.21 Example mission thread.

a request has been posted. This in turn will enable IntellCell's agent to correlate any observed request for details that it might see with the prior posting. Recall that interaction among participants in a mission thread is taking place in the context of ongoing use of the web services so that information exchanges have to be properly identified to provide useful information.

13.8.3

DEVS-Agent Endomorphic Models

The process is illustrated in Figure 13.22. First we prune the mission thread SES to select the observable actors, as in Figure 13.20 (shown in black words, e.g., Observable is selected from MajSmithObservabilitySpec. After extracting the observable actors, SES fragments can be generated that set up the assignment of agents to observable actors, both at the level of agentPair and each of its components:

- An agentPair can be MajSmith or IntelCell in observerType
- An observee can be MajSmith or IntelCell in observability
- An observer can be MajSmith or IntelCell in observability

These SES fragments, shown boxed in Figure 13.20 for the example, are automatically merged into the agentPairs SES of Figure 13.22. Then in the pruning process, an agentPair is selected for each of the observable actors, and the agentPair's observee and observer are pruned to the same actor. For example, we obtain MajSmith AgentPair which includes MajSmith observee, and MajSmith observer.

When the pruned entity structure, shown in Figure 13.23a, is transformed into a DEVS model, the result is the DEVS coupled model as shown in Figure 13.23b. This model contains agentPairs each of which consist of an observer and observee.



Fig. 13.22 Merger of SES segments into the SES for AgentPairs.

For example, the MissionThread of Figure 13.21 is embedded into the model shown in Figure 13.23 that contains agentPairs for each of the two observable components. Each observee is an instance of a class that inherits from the associated observable class of the MissionThread. For example, MajSmith observee inherits its DEVS functionality from the MajSmith class whose instance is in the example MissionThread. The coupling specification for the agentPairs' observee components reflects the coupling between the corresponding observable components in the MissionThread. In addition there is coupling between an observee and observer within the same agentPair, and between observers in different agentPairs. The latter represents the coupling that enables agent observers to coordinate and exchange information about the actors they are observing.

These types of couplings are illustrated in Figure 13.23. For example, the black arrow sequence from MajSmith Observee to IntelCell Observee is transferred from the corresponding black arrow sequence in Figure 13.21. Note that the ThreatEstimatePost output port of MajSmith Observee is coupled to the setOutput input port of MajSmith Observer representing its ability to intercept service requests in the actual test environment. The dark-grey arrow sequence from MajSmith Observer to Intel Observer represents the sequence of couplings that enable the former (af-

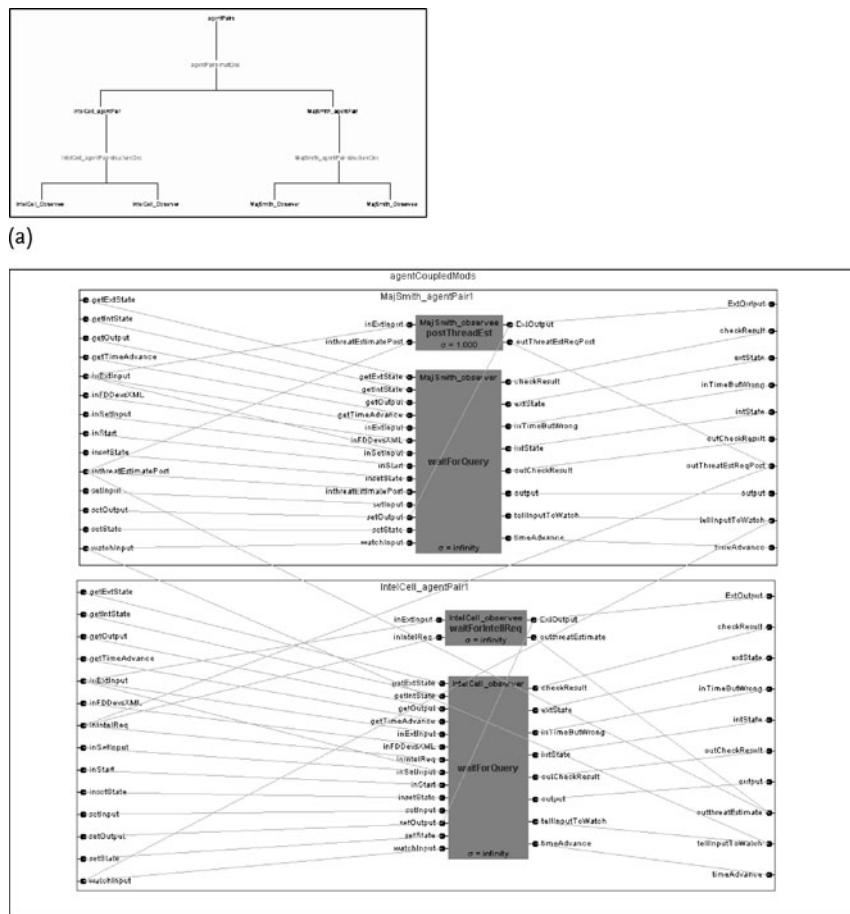


Fig. 13.23 The agent pairs model (b) generated by the transformation of the pruned entity structure of (a).

ter observing MajSmith's intelligence request) to inform the latter of the request to watch for in the near future.

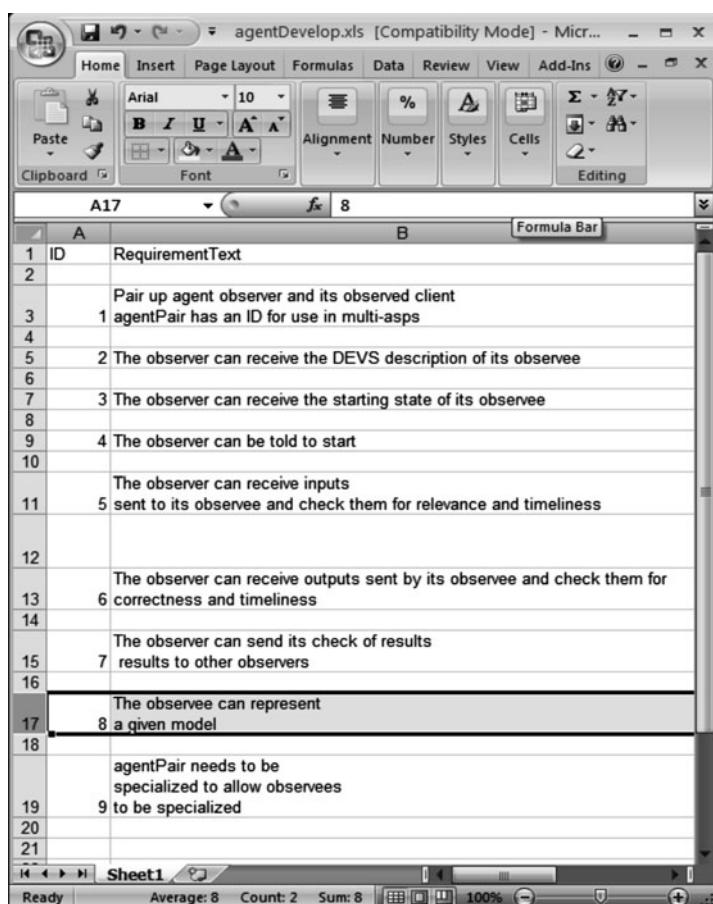
13.9

Summary and Conclusions

We have shown how the DEVS M&S framework provides an integrated development and testing methodology for net-centric systems, particularly those based on service-oriented architecture. The bifurcated system development methodology organizes the transition from user-stated requirements to both implementation and testing of implementations, in an integrated manner. The system development

includes the definition of requirements, capture of specifications to map formalized DEVS model components and create a reference master model. The test suite development includes execution of test models to run against the system under test. In particular for net-centric systems, the test models constitute experimental frames that can be distributed to observe the behaviors of, and information exchanges among, the nodes of the system. The process is iterative, allowing return to modify the reference master DEVS model and the requirements specifications. We elaborated on the role of requirements in the bifurcated methodology and the progression to development of test models for net-centric system testing. We showed how the test models are implementable using a concept of agent technology, where agents are DEVS models that can be distributed over an infrastructure that supports the DEVS simulation protocol.

The test methodology is currently being implemented to support testing of the Department of Defense's rapid transition to net-centric operation of all its agen-



A screenshot of Microsoft Excel showing a table titled "agentDevelop.xls [Compatibility Mode]". The table is on "Sheet1" and contains requirements for an "agentPair". The columns are labeled "A" and "B". The rows are numbered 1 through 21. Row 1 is a header with "ID" in A1 and "RequirementText" in B1. Rows 2 through 16 list numbered requirements. Row 17 is highlighted in gray and lists two requirements. Row 18 is partially visible. The formula bar shows "A17" and the value "8". The status bar at the bottom shows "Ready", "Average: 8", "Count: 2", "Sum: 8", "100%", and a zoom icon.

1	ID
2	RequirementText
3	Pair up agent observer and its observed client
4	1 agentPair has an ID for use in multi-asps
5	2 The observer can receive the DEVS description of its observee
6	
7	3 The observer can receive the starting state of its observee
8	
9	4 The observer can be told to start
10	
11	5 The observer can receive inputs sent to its observee and check them for relevance and timeliness
12	
13	6 The observer can receive outputs sent by its observee and check them for correctness and timeliness
14	
15	7 The observer can send its check of results to other observers
16	
17	8 The observee can represent a given model
18	
19	9 agentPair needs to be specialized to allow observees to be specialized
20	
21	

Fig. 13.24 Agent development example: define requirements.

cies, contractors, and personnel over its global internet, the Global Information Grid/Service Oriented Architecture (GIG/SOA). The Joint Interoperability Test Command (JITC) has the responsibility to test for GIG/SOA compliance for such projects as Net-Centric Enterprise Services (NCES) and Net-Enabled Command Capability (NECC). NCES is a major acquisition program of the Department of Defense (DoD) to deploy core services that enable information sharing by connecting people and systems that have information (data and services) with those who need information. These services are vital to Net-Centric Operations and Warfare. DoD's Net-Centric Environment (NCE) is radically different from commercial environments, particularly in the security requirements. Commercial tools have not been developed with DoD's stringent and demanding security and war-fighting environment in mind.

In its fully operational state, agent-implemented test methodology will provide testers the ability to deploy software agents to monitor, observe, and measure how

	A	B	C
1	ID	RequirementText	SESMicroRepresentation
2		Pair up agent observer and its observed client agentPair has an ID for use in 1 multi-asps	From a message perspective, the agentPair is made of observer and observee ! agentPair has an ID! The range of agentPair's ID is string !
3		The observer can receive the DEVS description of its 2 observee	From a message perspective, the agentPair sends FDDevsXML to the observer !
4		The observer can receive the 3 starting state of its observee	From a message perspective, the agentPair sends setState to the observer !
5		The observer can be told to 4 start	From a message perspective, the agentPair sends Start to the observer !
6		The observer can receive inputs sent to its observee and check them for relevance and 5 timeliness	From a message perspective, the agentPair sends inExtInput to the observer as inSetInput ! From a message perspective, the agentPair sends ExtInput to the observee !

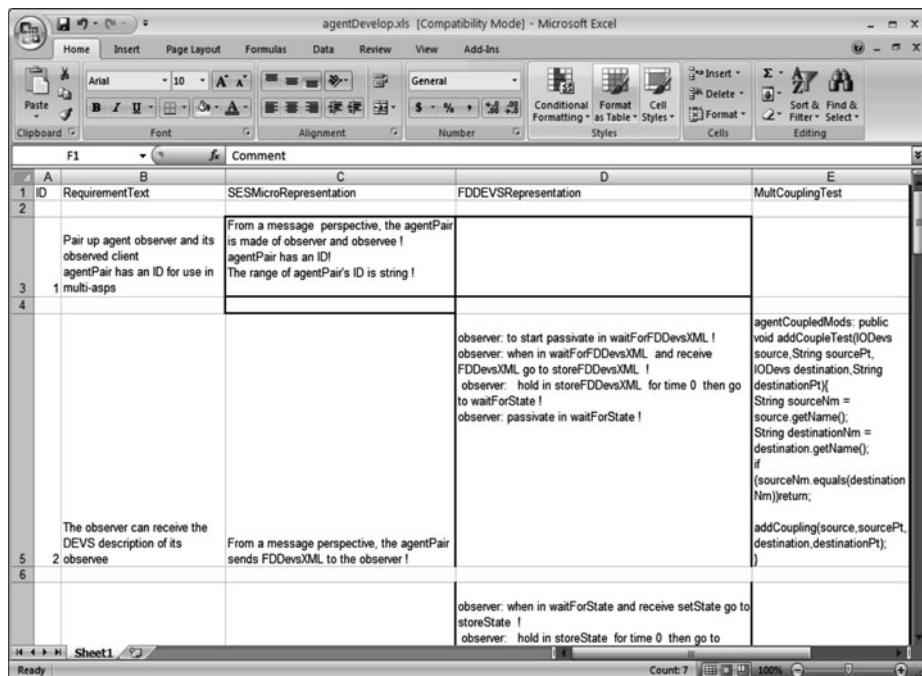
Fig. 13.25 Agent development example: define structural aspects.

well the NCES core services support mission threads developed by testers to emulate actual operational activities in critical DoD applications. DoD's project implementing this methodology will evaluate the effectiveness of the tools and underlying methodology in meeting the test and evaluation requirements before fielding the system in actual operational testing. This evaluation will reveal the extent to which the agent-implemented M&S concepts and bifurcated test and development methodology succeed in supported the demanding application to net-centric test and evaluation.

13.A

cAutoDEVS – A Tool for the Bifurcated Methodology

AutoDEVS is a software tool that supports the integrated bifurcated methodology for DEVS-based test agent development (Salas, 2008). This methodology is based on the DEVS modeling and simulation framework. It supports several stages of development that can be traversed initially in waterfall manner and subsequently in arbitrary steps while iterating until satisfactory results are obtained. Next, is a description of the different stages that AutoDEVS supports. The first stage of the AutoDEVS methodology is defining user requirements for the system. As seen in Figure 13.24, the requirements for the new system are collected and organized in



The screenshot shows a Microsoft Excel spreadsheet titled "agentDevelop.xls [Compatibility Mode] - Microsoft Excel". The table has five columns: A (ID), B (RequirementText), C (SESMicroRepresentation), D (FDDEVSRepresentation), and E (MultCouplingTest). Row 1 contains column headers: ID, RequirementText, SESMicroRepresentation, FDDEVSRepresentation, and MultCouplingTest. Row 2 contains: 1, "Pair up agent observer and its observed client. agentPair has an ID for use in 1 multi-asps", "From a message perspective, the agentPair is made of observer and observee ! agentPair has an ID! The range of agentPair's ID is string !", "", ". . .". Row 3 contains: 2, "The observer can receive the DEVS description of its 2 observee", "From a message perspective, the agentPair sends FDDEvsXML to the observer !", "observer: when in waitForFDDEvsXML and receive FDDEvsXML go to storeFDDEvsXML ! observer: hold in storeFDDEvsXML for time 0 then go to waitForState ! observer: passivate in waitForState !", "agentCoupledMods: public void addCoupleTest(IODEvs source, String sourcePt, IODEvs destination, String destinationPt){ String sourceNm = source.getName(); String destinationNm = destination.getName(); if (sourceNm.equals(destinationNm))return; addCoupling(source,sourcePt,destination,destinationPt); }". Row 4 is empty. Row 5 contains: 3, "", "From a message perspective, the agentPair sends FDDEvsXML to the observer !", "observer: when in waitForState and receive setState go to storeState ! observer: hold in storeState for time 0 then go to", "Count: 7". Row 6 is empty. The bottom of the screen shows the Excel ribbon and status bar.

A	B	C	D	E
ID	RequirementText	SESMicroRepresentation	FDDEVSRepresentation	MultCouplingTest
1	Pair up agent observer and its observed client. agentPair has an ID for use in 1 multi-asps	From a message perspective, the agentPair is made of observer and observee ! agentPair has an ID! The range of agentPair's ID is string !		
2			observer: start passivate in waitForFDDEvsXML ! observer: when in waitForFDDEvsXML and receive FDDEvsXML go to storeFDDEvsXML ! observer: hold in storeFDDEvsXML for time 0 then go to waitForState ! observer: passivate in waitForState !	
3	The observer can receive the DEVS description of its 2 observee	From a message perspective, the agentPair sends FDDEvsXML to the observer !		agentCoupledMods: public void addCoupleTest(IODEvs source, String sourcePt, IODEvs destination, String destinationPt){ String sourceNm = source.getName(); String destinationNm = destination.getName(); if (sourceNm.equals(destinationNm))return; addCoupling(source,sourcePt,destination,destinationPt); }
4				
5			observer: when in waitForState and receive setState go to storeState ! observer: hold in storeState for time 0 then go to	
6				

Fig. 13.26 Agent development example: define behavioral aspects.

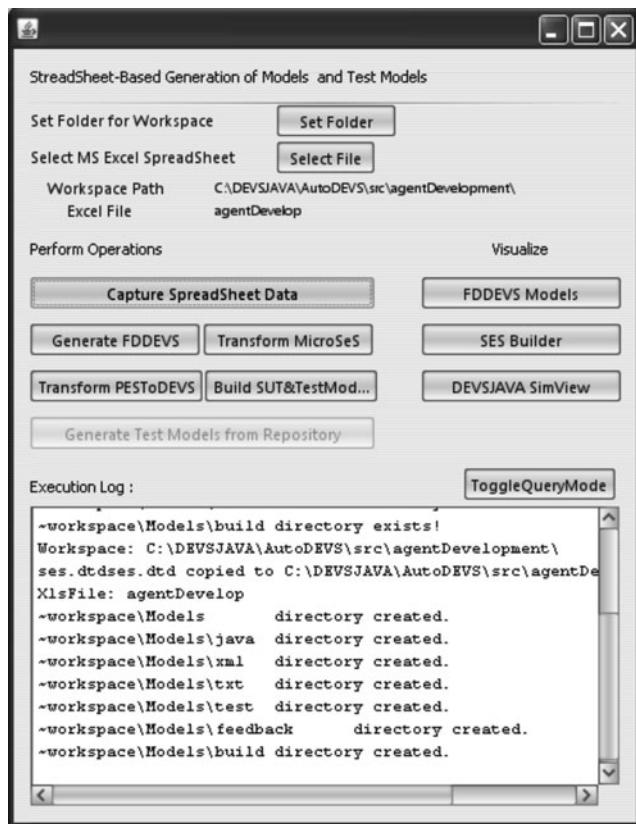


Fig. 13.27 Agent development example: capture spreadsheet data.

spreadsheet table, “RequirementText” column. The second stage is describing the structural aspects for each of the requirements, that is, fill the “SESMicroRepresentation” column, refer to Figure 13.25. The third stage is describing the behavioral aspects for each of the requirements, that is, fill the “FDDEVSRepresentation” column, refer to Figure 13.25. Note that the second and third stages can be intermixed as the design emerges iteratively. The fourth stage is defining the multiaspect coupling for the coupled models, see Figure 13.26. This is defined in the “MultiCouplingTest” column and automates the coupling generation for multiaspect models. The fifth stage is running the AutoDEVS tool to capture the spread sheet data, that is, agentDevelop.xls, see Figure 13.27.

Notice that the user needs to set the folder and spreadsheet file to be captured in the AutoDEVS tool, that is, “Set Folder” and “Select File” buttons. In addition, notice that subsequent to capturing the data from the spreadsheet, AutoDEVS encodes the captured data into an XML schema/document type definition (XSD or DTD), that is, Sheet1agentDevelopRowsSchema.xsd, ses.dtd. The sixth stage is to generate FDDEVS models based on the schema type definition and the captured

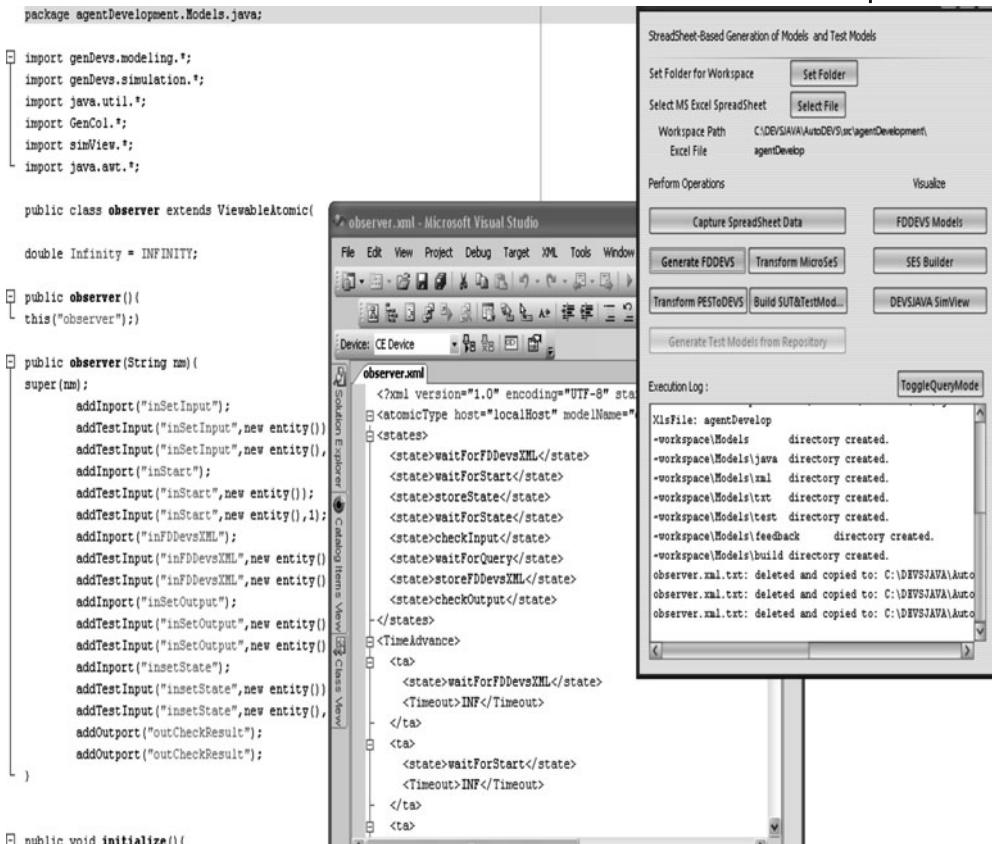


Fig. 13.28 Agent development example: generate FDDEVS models.

XML data from previous stage, that is, behavioral aspects of the system (FDDEVSRepresentation column). As seen in Figures 13.27, 13.28, DEVS java models are automatically generated, including an XML representation of those models, that is, observer.java, observer.xml. By clicking on the FDDEVS Models button, these representations can be viewed and inspected, 13.29. Based on the MicroSES-Representation column defined in the spreadsheet, the seventh stage is to add the structural aspects on the DEVS models created in the previous stage. During this stage implemented by the Transform MicroSES button, SES representations of the models are created and parsed into an XML file, that is, agentDevelopagentCoupledModsSeS.xml. Notice that this file could serve to see the SES representation as a tree view, see Figure 13.30. In addition, an automatic PES that represents the logically possible set state descriptions consistent with the SES is created, that is, agenDevCoupModInst.xml.

The eighth stage is to run the PES that was automatically created in the previous stage, that is, Transform PEStoDEVS. During this stage the specialized models are

```

public void delint() {
    if (phaseIs("checkOutput")) {
        passivateIn("waitForQuery");
    } else if (phaseIs("storeFDDevsXML")) {
        passivateIn("waitForState");
    } else if (phaseIs("checkInput")) {
        passivateIn("waitForQuery");
    } else if (phaseIs("storeState")) {
        passivateIn("waitForStart");
    } else if (phaseIs("waitForFDDevsXML")) {
        passivateIn("waitForFDDevsXML");
    } else {
        passivate();
    }
}

public void deltext(double e, message x) {
    Continue(e);
    for (int i = 0; i < x.getLength(); i++) {
        if (this.messageOnPort(x, "inSetOutput", i)) {
            if (phaseIs("waitForQuery")) {
                processcheckOutput();
                holdIn("checkOutput", 0.0);
            }
        }
        if (this.messageOnPort(x, "inSetInput", i)) {
            if (phaseIs("waitForQuery")) {
                processcheckInput();
                holdIn("checkInput", 0.0);
            }
        }
        if (this.messageOnPort(x, "inStart", i)) {
            if (phaseIs("waitForStart")) {
                processwaitForQuery();
            }
        }
    }
}

```

Fig. 13.29 Agent development example: generate FDDEVS.

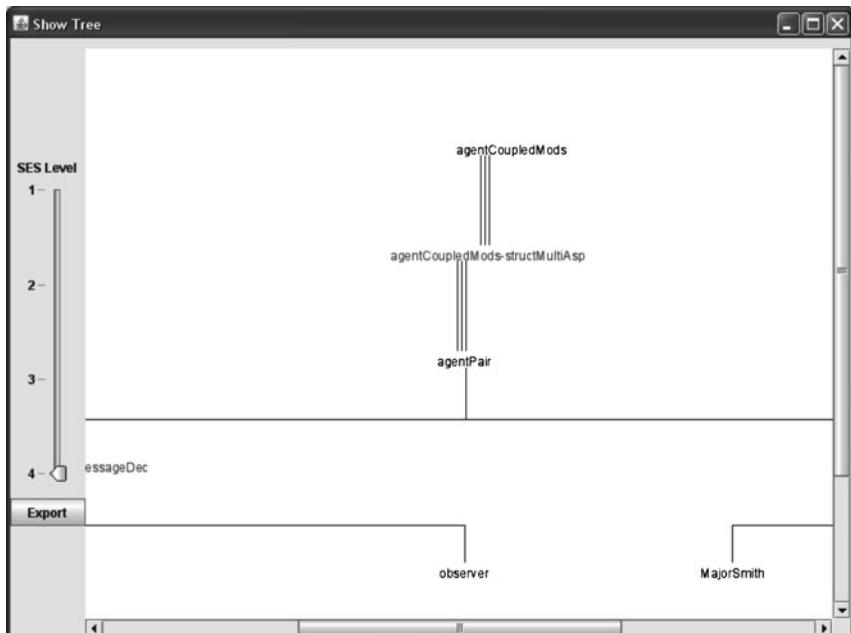


Fig. 13.30 Agent development example: SES tree view.

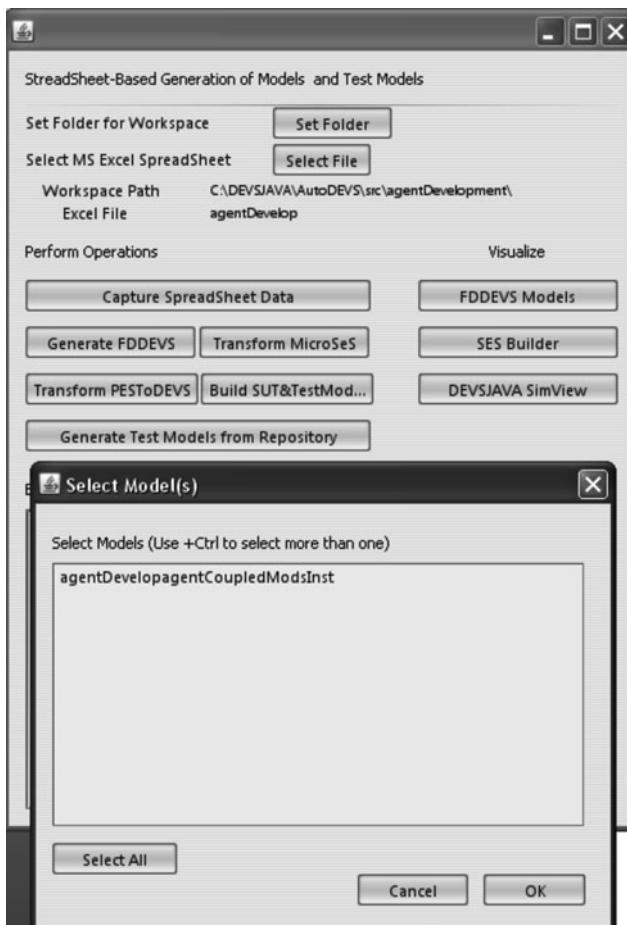


Fig. 13.31 Agent development example: choosing a PES.

created and the DEVS models are updated with the corresponding structural aspects, that is, PES transformed into DEVSJAVA models. This PES can also be modified by the developer to create his own pruning and analyze the models of interest. The AutoDEVS tool allows choosing the PES desired and then run it in the system, as shown in Figure 13.31. The ninth stage is to create a set of test models to validate the system under development, see Figure 13.30. As seen in Figure 13.33, AutoDEVS lets the developer choose from a list the test models to create. As described previously, these test models are based on minimal testable I/O pairs restricted to messages and are created with the objective to verify the correctness of the DEVS models. The tenth stage is to verify the models created in the FDDEVS Models and SES Builder DEVSJAVA SimView applications, that is, Figure 13.34, and then to modify the models according to the desired needs and start the simulation.

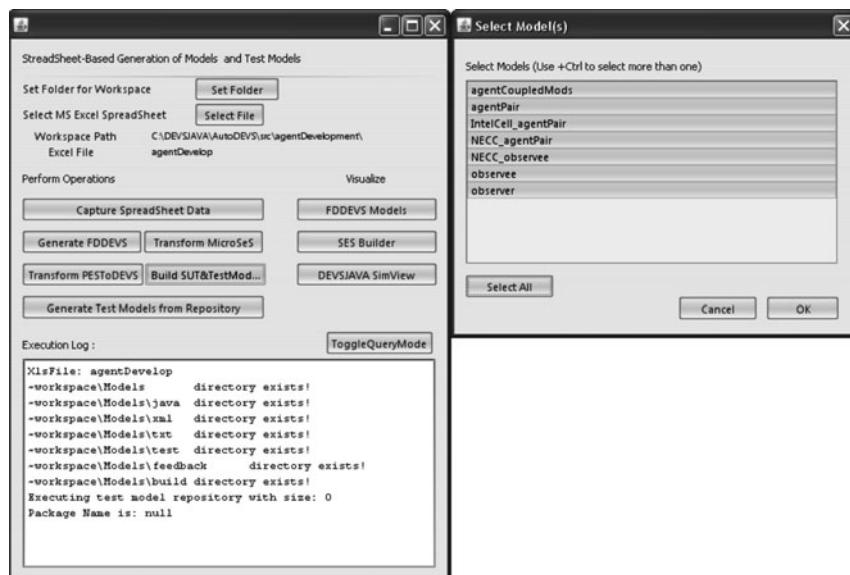


Fig. 13.32 Agent development example: generate test models.

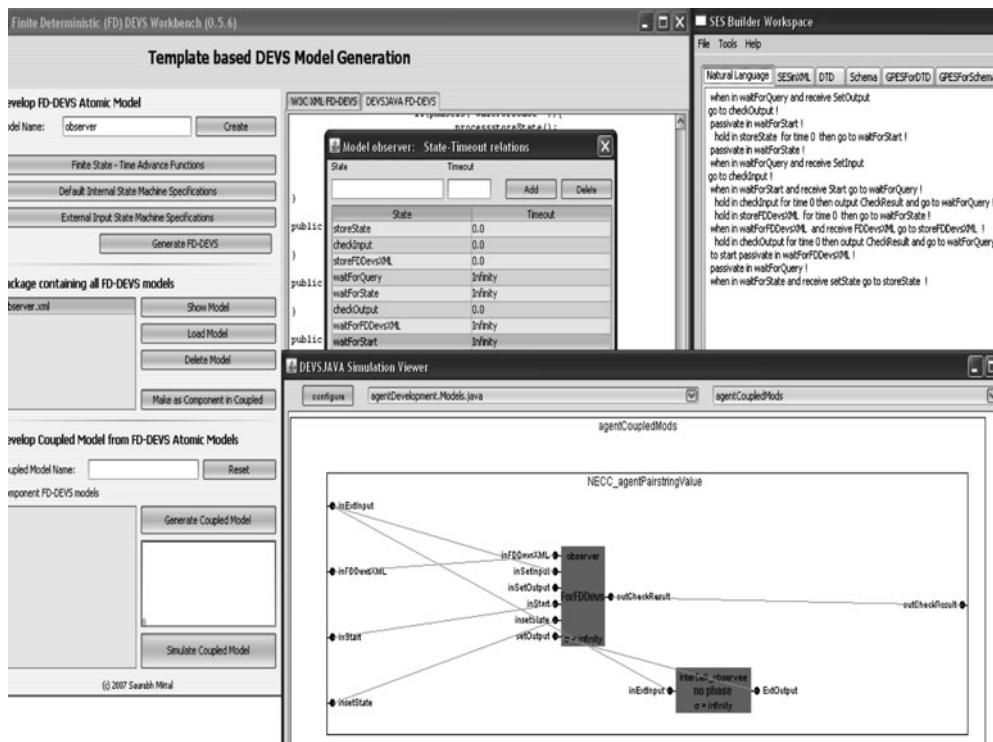


Fig. 13.33 Agent development example: generate test models.

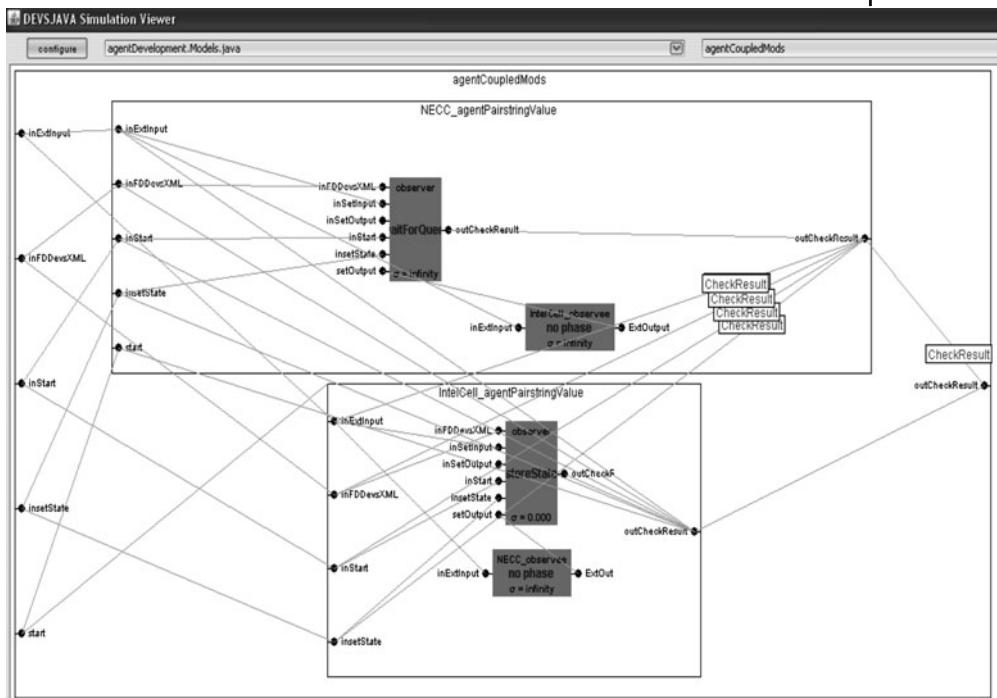


Fig. 13.34 Agent development example: running models in SimView.

References

- DOD 5000.59 (1994) *Modeling and Simulation Management*, Department of Defense. 1985
- Agha, G. and Hewitt, C. (1985) *Concurrent programming using actors: Exploiting large-scale parallelism*. Proceedings of the Foundations of Software Technology and Theoretical Computer Science, Fifth Conference, pp. 19–41.
- Firby, R.J. (1992) *Building symbolic primitives with continuous control routines*. Proceedings of the First Int. Conf. on AI Planning Systems, pp. 62–69.
- Forsberg, Mooz, and Cotterman (2005) *Visualizing Project Management: Models and Frameworks for Mastering Complex Systems*.
- Gasser, L., Braganza, C., and Herman, N. (1987) Mace: An extensible testbed for distributed AI approach. *Distributed Artificial Intelligence – Research Notes in Artificial Intelligence*, pp. 119–142.
- Grady, J.O. (2006) *System Requirements Analysis*, Academic Press.
- Hooks, F.I. (1994) *Guide for Managing and Writing Requirements*, Compliance Automation, Inc.
- Hu, X. (2004) *A Simulation-based software development methodology for distributed real-time systems*, University of Arizona.
- Hu, X. and Zeigler, B.P. (2005) Model continuity in the design of dynamic distributed real-time systems. *Doctoral Dissertation – Electrical and Computer Engineering Dept.*, University of Arizona.
- Mittal, S. (2007) *Devs unified process for integrated development and testing of service and testing fo service oriented architectures*, University of Arizona.
- Mittal, S., Zeigler, B., Martín, J., Sahin, F., and Jamshidi, M. (2009) Modeling and simulation for systems engineering.

- Systems of Systems Engineering*, page to appear.
- Ören, T. (2005) *Maturing phase of the modeling and simulation discipline*. Proceedings of: ASC – Asian Simulation Conference 2005, Beijing, P.R. China, International Academic Publishers – World Publishing Corporation, Beijing, P.R. China, pp. 72–85. <http://www.site.uottawa.ca/~oren/pubs-pres/2005/pub-0512-maturing.pdf>.
- Ören, T. and Zeigler, B.P. (1979) Concepts for advanced simulation methodologies. *Simulation*, 32(3), 69–82.
- Rao, A.S. and George, M.P. (1987) *Bdi-agents: from theory to practice*. Proceedings of the First International Conference on Multiagent Systems.
- Salas, M.C. (2008) Autodevs: A methodology for automating systems development. *Masters Thesis – Electrical and Computer Engineering Dept., University of Arizona*.
- Wymore, W.A. (1993) *Model-based Systems Engineering: An Introduction to the Mathematical Theory of Discrete Systems and to the Tricotyledon Theory of System Design*, CRC, Boca Raton, FL.
- Zeigler, B.P. (2008) *Toward a Next Generation Standard for Modeling and Simulation Interoperability*, Military Modeling and Simulation, Singapore.
- Zeigler, B.P. (1976) *Theory of Modelling and Simulation*, John Wiley & Sons New York, N.Y.
- Zeigler, B.P. (1990) *Object Oriented Simulation with Hierarchical, Modular Models: Intelligent Agents and Endomorphic Systems*, Academic Press, Orlando, FL.
- Zeigler, B.P. and Hammonds, P.E. (2007) *Modeling & Simulation-Based Data Engineering: Introducing Pragmatics into Ontologies for Net-Centric Information Exchange*, Academic Press.
- Zeigler, B.P., Kim, T.G., and Praehofer, H. (2000) *Theory of Modelling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems*, Academic Press, Boston.

14

Agents and Decision Support Systems

Andreas Tolk, Poornima Madhavan, Jeffrey W. Tweedale, and Lakhmi C. Jain

14.1

Introduction

In this chapter, the focus lies on agents in the context of systems engineering for decision support systems. The chapter starts with the historical context resulting in the motivation for agent-directed decision support simulation systems and some working definitions. Next, the cognitive and technical foundations for decision support systems will be dealt with before some examples are given. Many related topics are still under current evaluation, so that this chapter is often based on current and ongoing research and new results can be expected. Nonetheless, each system engineer being tasked with the application or development of an agent-directed decision support system needs to know the foundations on which is built the core of this contribution.

14.1.1

History

Early information systems represented traditional filing systems, such as a filing cabinet or Rollerdex. As computers became established in business, this definition evolved to include data mining, expert systems, and, more recently, decision support systems. These applications originally ran in batch mode, like mainframe payroll and accounting applications. With the introduction of applications like spreadsheets, information management systems (IMS) began to incorporate analytical feedback. Data mining and data warehousing also facilitated knowledge-based systems (KBSs) that utilized the different artifacts for knowledge representation (Sowa, 2000) to capture information and knowledge about an application in machine-supported ways. As the *Information Age* matures (Jones, 1990), the variety of intelligent decision support systems (IDSSs) generated the tools and access to knowledge. These developments increased management's confidence in reasoning and strategic decision support tools. As the Internet matured and became a reliable backbone for worldwide distributed web-based applications, the next step in the

evolution of decision support was initiated, introducing the new challenge to describe the functionality of services in such a way that potential users could identify solutions based on these descriptions. Agents play an increasing role in supporting all these tasks.

According to Taylor (1911), science enables collaboration between workers and managers. He perfected the concept of “task decomposition” to optimize the efficiency of work practices establishing task and interface descriptions. Edersheim (2006) pioneered modern management practices and created decentralized organizations while promoting trust/respect for worker and repeat customers to optimize profit. In both views, machines were evolved to replace humans. Although such machines did displace many, today factories full of machines – operated by humans – have mechanized a significant number of production processes. Decision support systems have to be evaluated – and designed – in this context as well. They are intelligent machines, but they are still machines. Their main task is to support the decision process of those human beings in charge of such complex operations, and take over in particular the routine work that does not require complex human interaction. Agents are needed in this environment to observe the system and recognize routine tasks to be executed as well as circumstances that require the human decision maker to become active. In addition, agents have to mediate information of the operational environment into the data and knowledge bases used by the supporting machines.

As these environments were constantly changing and new requirements are produced, the decision support systems had to become flexible and adaptive. In the beginning, many early decision support systems were modified using proprietary applications that were not well aligned with business processes. Learning was gleaned mostly after analyzing success and failure (mistakes). As this was not satisfactory for the support of ongoing business, it was soon recognized that sound reasoning processes in rapidly changing environments demanded rigorous evaluation processes during the process of support. Such a set of reasoning models for the knowledge base to undergo rapid change needs validation prior to being incorporated into the body of knowledge. Thus, new systems need to be able to interpret, discriminate, and classify knowledge in real time. Overall, this presents context and currency issues for decision support in an information-rich environment. These tasks show the close proximity to the learning ideas applicable to intelligent agents as defined in earlier chapters of this book.

All learning methods require to measure improvements of behavior based on new knowledge. In order to support this, measures of merit are needed. A series of such measures were created to report on the performance of decision support systems, using factors such as: accuracy, response time and explainability. Related issues were raised as constraints to be considered before specifying *courses of action* (Dhar and Stein, 1997). Other heuristic measures could be used to measure the quality of utility within a decision support system and provide feedback to balance efficiency within specified solutions. Today, decision support systems are accepted and applied in many domains. The interested reader is referred to an overview of intelligent software systems in the historical context given by Pohl (2005).

The current research focus of this chapter lies in intelligent decision support systems. The way to introduce intelligence to decision support systems in the scope of this book – besides applying heuristics within the system – is to use agent technology. Before dealing with the necessary foundations, a short motivation is given.

14.1.2

Motivating Agent-Directed Decision Support Simulation Systems

The simple motivation is that agent-directed decision support simulation systems are applied based on the assumption that they will help make better decisions. The underlying assumption is that the decision maker to be supported will follow a rational decision process, which means that he/she has a set of defined goals that he wants to accomplish using his/her resources optimally. Therefore, the decisions to be supported must be *normative* or *prescriptive*. The underlying decision theories are documented in many textbooks, such as that by Bell et al. (1988).

In the optimal case following this paradigm, the decision maker is fully informed, computes with perfect accuracy, and is fully rational in his/her decisions. In reality, this is very unlikely. Decision support systems shall support these rational processes by approximating the real-world conditions to the optimal case. Consequently, we assume the following hierarchy of the quality of decisions:

- Decisions can be made without any support, just based on observation and perception, and may comprise communications with others.
- Using a decision support system helps to build a “common picture” of the current situation. This will help to establish a context improving the decision process. Instead of data points, data in context – also known as information – is used.
- While a common picture is good, to understand the current situation in the temporal-spatial context of a “common executable model” is better. A simulation can provide the technical means to display the behavior of a system over time, including following observed trends to predict future possible states. The decision can be based on (procedural) knowledge.
- No matter how good a simulation system is, it can never be complete and certain, as the underlying model itself is a purposeful abstraction of reality. In addition, the data that are used are not perfect either: it is more than likely that they are incomplete, uncertain, vague, and contradictory. Perpetual improvements of the simulation system by adapting to the environment and learning from bad and good decisions are necessary. Agents can help with these tasks, as the required characteristics are well known in agenthood as defined earlier in the book.
- Agents can support the selection and composition of applicable systems using their *social abilities* as ambassadors for the potential system. Representing the abilities and constraints of the decision support system, they can negotiate with other system ambassadors on how best to support the overall decision process.

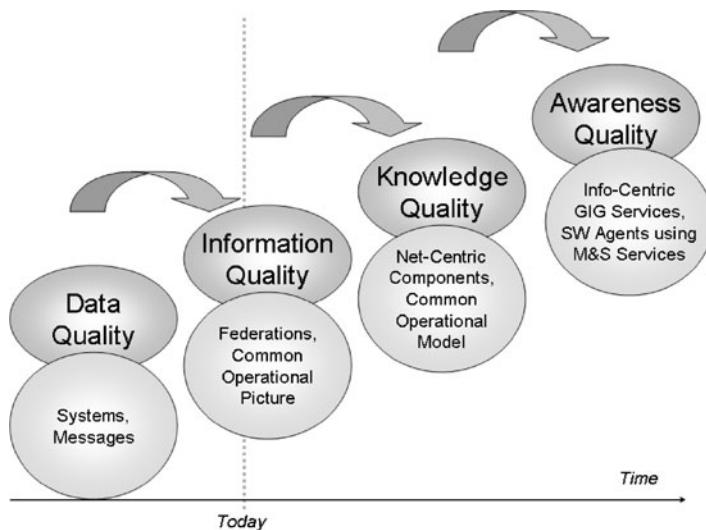


Fig. 14.1 Quality improvements in command and control.

In summary, we need decision support systems to embed data into information. We need simulation systems to apply the information dynamically, which results in knowledge, and we need intelligent agents to support application of the knowledge, resulting in awareness. The following figure was developed to show the benefits of related techniques in support of military command and control. Experiments conducted by the armed forces suggest that the command-and-control quality is improved by an order of magnitude when a new level of quality is reached in the value chain of data – information – knowledge – awareness. It was introduced by Tolk (2005) and discussed during several decision support and command-and-control symposia. The value chain for command and control is explained in more detail by Alberts and Hayes (2003).

In the first stage, decisions are based exclusively on messages and similar data exchange means. Introducing the common picture helps the distributed decision makers to base decisions on a common context. Introducing simulations that can be executed via the underlying network as net-centric components results in common knowledge including spatial and temporal dynamics, a common executable model. The use of intelligent agents to select and compose as well as to learn and adapt ultimately results in an awareness of possible developments.

Decisions are normally not made under the “clinical conditions” of a laboratory. The real-world environment for decision support systems in general and agent-directed decision support simulation systems in particular is the most challenging configuration for system engineers:

- The environment is normally nonaccessible or partly accessible. Only parts of the data needed for a fully informed decision can be obtained.

- The sensors used or data sources identified will rarely support perfect accuracy. As the environment is continuous and not discrete, that adds additional numeric errors and uncertainties.
- The environment is likely to be highly stochastic.
- The environment is sequential, that is, former actions can still have effects (or can result in effects of effects) in the current observation episode.
- The environment changes dynamically and is filled for other players competing for the same resources, having opposing interests and similar characteristics.

Consequently, decisions have to be made based on imperfect data under uncertainty and under the pressure of competition for a limited set of resources.

In summary, the way from where we are to the objective of agent-directed decision simulation support systems is challenging, as many requirements are not fulfilled, and some of these requirements are not even fully understood by the community. In the following sections, the state of the art will be summarized and topics of research will be identified. To be able to do so, we need to introduce some working definitions.

14.1.3

Working Definitions

In this chapter, we use the working definition for intelligent agents derived earlier in this book: The agent is situated, it *perceives* its environment, and it *acts* in its environment. The *communication* with other agents is of particular interest in systems comprising multiple agents, as agents can collaborate and compete for tasks. This latter characteristic has also been referred to as *social ability*. The agent should be *autonomous*, in the sense that it can operate without the direct intervention of humans or others and autonomy requires control about its own state and behavior. They must be guided by some kind of goal and value system. To be *flexible* for an agent means to mediate between reactive behavior, being able to react to changes in its environment, and having deliberativeness to pursue its goals. A suitable mediation is one of the critical aspects for an agent to achieve its tasks in a dynamic environment. An agent can act upon its knowledge, its rules, beliefs, operators, goals, and experiences. It can *adapt* to new constraints and requirements – or even new environments – as required. For example, new situations might ask for new goals, and new experiences might lead to new behavior rules. Also, being *mobile* adds to the flexibility of an agent.

For the working definitions used in this chapter, we are following the ideas of agent-directed simulation (ADS) as discussed, among others, in Yilmaz and Ören (2007). Agents as described above can be used to significantly improve simulations as well as the applications that utilize them. As a first step, we generalize the terms introduced for ADS for agent-directed systems. Comparable to the ideas captured in the ADS metaphor, we distinguish between two categories:

- *Systems for agents* are systems implementing agents for use in engineering, human and social dynamics, military applications, and others.
- *Agents for systems* are divided into two subcategories. *Agent-supported systems* deal with the use of agents as a support facility to enable computer assistance in problem solving or enhancing cognitive capabilities. *Agent-based systems* focus on the use of agents for the generation of model behavior in the system (such as is needed in support of system studies and analyses).

Furthermore, we distinguish between *data-driven environments*, in which the focus lies on finding and representing data and information, and *process-driven environments*, in which the focus lies on presenting the development of data over time. In addition, we distinguish between traditional systems and agent-directed systems. With these definitions in mind, the following simple taxonomy for decision support systems as depicted in Figure 14.2 is defined as follows:

Decision Support Systems: The generally accepted definition is that decision support systems are information systems supporting operational (business and organizational) decision-making activities of a human decision maker. The decision support systems shall help decision makers to compile useful information from raw data and documents that are distributed in a potentially heterogeneous IT infrastructure, personal or educational knowledge that can be static or procedural, and business models and strategies to identify and solve problems and make decisions.

Agent-directed Decision Support Systems: As long as the system can be designed in advance and the focus lies on data, in practice the application of decision support systems is often sufficient. However, when the environment is chang-

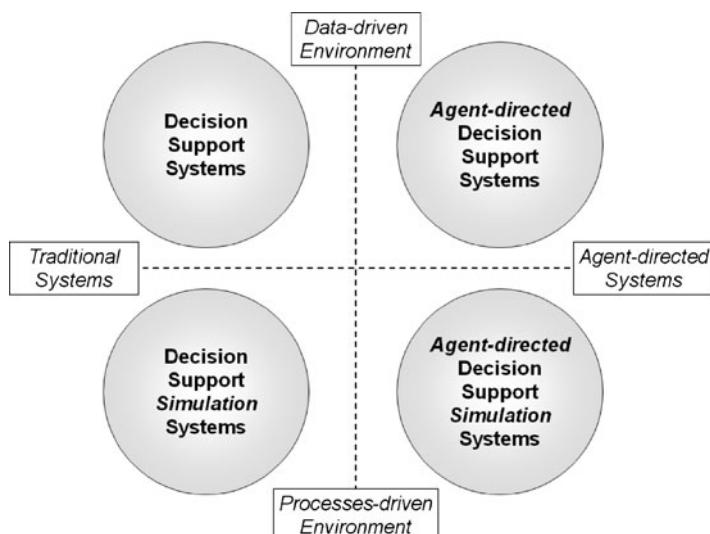


Fig. 14.2 Taxonomy of decision support systems.

ing rapidly and the system needs to be able to adapt to this new environment, technologies as introduced for the agent metaphor are needed. The application of agent-directed methods to improve decision support system results in *agent-directed decision simulation systems*. It is also possible that agents are used to support the selection and composition of decision support systems.

Decision Support Simulation Systems: The traditional view of decision support systems is still dominated by the collection and presentation of data. Simulation added a new feature by adding the model-based development of these data over time, focusing on the processes. We therefore define *decision support simulation systems* as simulation systems supporting operational (business and organizational) decision-making activities of a human decision maker by means of modeling and simulation. They use decision support system means to obtain, display, and evaluate operationally relevant data in agile contexts by executing models using operational data exploiting the full potential of modeling and simulation and producing numerical insight into the behavior of complex systems.

Agent-directed Decision Support Simulation Systems: An agent-directed decision support simulation system is an ADS that is applied as a decision support system. The focus lies on processes, and the system can adapt to new requirements and constraints in the environment.

It should be pointed out that the focus of this chapter lies in the *agent for systems* component: selecting and composing systems for decision support is a typical example of agent-supported applications; adapting to a new environment is a good example of agent-based applications. However, the use of decision support systems and decision support simulation systems as a *system for agents* is interesting as well; it just could not be covered in the same detail within this chapter. Nonetheless, the term *agent-directed decision support simulation systems* was chosen on purpose to make the reader aware of all aspects that need to be captured.

Following the arguments in the motivation, the ultimate goal must therefore be to establish *agent-directed decision support simulation systems*. In practical applications, however, the desired objective may already be achievable by other forms of decision support systems. As always, rigorous requirement and context analyses for the sponsor's problem must be the basis for the appropriate system engineering.

14.2

Cognitive Foundations for Decision Support

In this chapter, the authors will contribute to the discussion about necessary tools and techniques. However, before the focus is directed at the functional aspects of decision support systems, another viewpoint shall be introduced first that is often neglected in the traditional engineering domain, namely, cognitive foundations.

Traditional system engineering focuses on aspects like functional, physical, and operational architectures of systems. The social components and human and organizational factors are not always taken as much into account as needed, which often resulted in surprises regarding the acceptance of new systems – or the lack thereof – when the system is introduced to the market. System of systems engineering as defined by Keating et al. (2003) stresses the necessity to evaluate a system in all its contexts, including the social component.

As mentioned earlier, machines were evolved to replace humans. Although the role of decision support systems is not to replace but to support the decision maker, decision support systems replaced humans that used to compile useful information from raw data and documents from distributed sources. As such, trust and credibility are important for this new “partner decision support system” within the operation. This section will deal with some selected topics belonging to the cognitive foundations for decision support.

14.2.1

Decision Support Systems as Social Actors

The introduction of automation into complex systems such as aircraft cockpits, nuclear power plants, and air traffic control rooms and other domains as referenced above has led to a redistribution of operational responsibility between human operators and computerized decision support system. Although decision support systems should support and not replace decision makers, in many domains the role of the human operator de facto has metamorphosed from that of a primary controller to an active teammate sharing control with automation. Specifically, automated decision support systems are increasingly being perceived as “partners” rather than as tools (Klein et al., 2004). Therefore, human-system integrated cognition is a topic of significant interest to researchers in the present day.

Several modern decision support systems are designed to interact or behave in a manner similar to a human, imitating human language structures where applicable and often possessing unique knowledge and functional algorithms that may be inaccessible to human teammates. Indeed, some researchers have argued that such human-automation teams function similarly to human-human teams (Bowers et al., 1996, 1998), and scientific evidence suggests that people do enter into “relationships” with computers, robots, and interactive decision support systems in a manner similar to their relationships with other humans (Nass et al., 1996; Reeves and Nass, 1996). This is supported by the “computers are social actors” or CASA studies (Nass and Moon, 2000; Nass et al., 1995, 1993, 1994) that have demonstrated that social rules guiding human-human interaction may apply equally to human-system interaction; people reportedly apply politeness norms and gender stereotypes to computers and get “attracted” to systems with “personalities” that match their own (Nass and Lee, 2001).

In keeping with the CASA studies are suggestions that human-machine joint cognition is influenced strongly by human trust in the decision support systems (e.g., Dijkstra (1999)). The concept of human trust in systems has been the focus

of a vast body of research over the last decade (e.g., Blomqvist (1997); Lee and Moray (1992, 1994); Lee and See (2004); Lerch et al. (1997); Muir (1987, 1994)). In this section, we discuss the key cognitive factors that influence human trust in decision support systems and the implications derived from this for presenting the system to the user to ensure appropriate system utilization.

14.2.2

How to Present the System to the User and Improve Trust

Trust refers to the expectation of another, or confidence in another, and is based on the probability that one party assigns to cooperative or favorable behavior by other parties (Barber, 1983; Hwang and Buergers, 1997; Muir, 1987). In the context of human interaction with decision support systems, *system trust* refers specifically to an operator's belief that a system is operating in a predictable manner and in keeping with expectations (Lewis and Weigert, 1985; Luhmann, 1979, 1988). There are multiple system characteristics that could potentially influence human trust and human-decision support system integrated cognition. The most critical of these factors and possible solutions to effectively present the system to the user are discussed below.

- *Perceptual system features*: At the lowest level perceptual features of systems play an important role in human-decision support system integrated performance. The most important among them are as follows.
 - *Redundancy* – Critical system information must be presented via both visual and auditory channels. This principle is elaborated in the multiple resource theory (Wickens, 2002). According to this theory, performing multiple tasks in different modalities (visual vs. auditory – scanning a display and listening to auditory instructions) causes less cognitive interference and is easier than performing multiple tasks in the same modality (e.g., two visual tasks – scanning a display and reading instructions).
 - *Proximity compatibility principle* – Information that must be mentally integrated (e.g., switches, dials, readouts) must be placed together on a spatial display, whereas information that must be processed separately must be spatially separated. For example, controls must be arranged in the same configuration as the dials they control. If the dial is placed on the right of a display, then the corresponding control must also be placed on the right side of the display.
- *System reliability*: Logically, users should depend on a decision support system more when it is more reliable rather than when it is less reliable. This is based on the assumption that operators are accurate judges of system reliability. However, results of studies that have systematically varied decision support system reliability levels are mixed, with some suggesting that individual operators are sensitive to different levels of reliability (e.g., Parasuraman et al. (1993); Wiegmann et al. (2001)), while others suggest that operators are insensitive to reliability differences (e.g., Dzindolet et al. (2001)). These findings hold true largely for established tried-and-true systems.

Madhavan and Phillips (2008) examined whether “ignorance is bliss” when presenting a new decision support system to users. In other words, is it really necessary to inform users of the “new and untested” status of a novice decision support system, or is it more advisable to not provide such information, thereby allowing users to develop their own assessments of system reliability? The results revealed that ignorance is not bliss when presenting a user with a decision support system; even in systems of low or unproven reliability it is vital for users to be informed as precisely as possible of what the system is capable or incapable of accomplishing. However, providing too much information can have counterproductive effects; it can prematurely bias users’ perceived “credibility” of the system.

- *System pedigree:* The pedigree is mainly defined by its *credibility*, defined as the extent to which an entity can be relied on to do what it says it will do (Herbig and Milewicz, 1993). Research on human interaction with decision support systems has revealed that decisions to trust in and rely on a decision support system are largely a function of the perceived credibility of these systems. Human operators tend to be influenced more by what they are told about a system rather than the actual diagnostic value (or reliability) of the system in itself. This eventually leads operators to interact with the aid based on its presumed rather than actual expertise.

When presenting a new decision support system to users, one school of thought is that there is a tendency for operators to be somewhat distrustful of the reliability of new decision support system until such systems have proven themselves (Parasuraman and Riley, 1997; Sheridan, 2002). This was refuted in a recent study, which revealed that operators believe a decision support system to be reliable until it proves otherwise (Madhavan and Wiegmann, 2007). This is particularly true when users interact with new decision support systems; users have more realistic expectations of a decision support system when they are aware that the system is a novice. In such situations, users are more capable of making objective assessments of system reliability and are more “forgiving” of system errors. Conversely, when the same decision support system is portrayed as an *expert*, there is a more rapid breakdown in trust in the decision support system when it generates an error, suggesting that high initial expectations of expertise accompanied by errors can lead to very damaging effects on decision support system trust and utilization.

- *System errors:* For the psychologist, it is not only of interest what errors occur, but also how they are perceived. Underlying studies have shown that two factors are very important: if the system gives a false alarm vs. missing an observation where an alarm was required, and what level of complexity the decision is embedded in.
 - *False alarms vs. misses* – Decision support systems that generate a large number of false alarms (i. e., “cry wolf”) (Breznitz, 1983) create under-trust in automation (Parasuraman and Riley, 1997; Gupta et al., 2001) and consequently affect user compliance with decision support systems. Re-

search suggests that false alarms may indeed be more degrading of trust than misses (failing to detect some phenomenon); see Cotte et al. (2001), Gupta et al. (2001), and Maltz and Shinar (2003). This is because a false alarm directs the operator's attention from other important tasks, ultimately resulting in the operator wasting time and effort in dealing with an alarm of no diagnostic value. Users are likely to lose trust in such a system that demands this extra effort (Thomas et al., 2003).

A miss, on the other hand, does not require extra attention to be focused on a false event, thereby maintaining the initial level of trust in the system. In several cases, a miss might go undetected for a long time and cannot be traced to a particular cause, thereby having a less degrading effect on user trust.

Contradicting evidence, however, suggests that under some circumstances misses are potentially more damaging and degrading of trust than false alarms (Masalonis and Parasuraman, 2003). An example of such a situation is the catastrophic midair collision of two aircraft as a consequence of cockpit decision support systems failing to detect and signal the presence of conflicting flight paths. The misses vs. false alarms tradeoff and its effects on trust calibration is likely mediated by the immediacy, criticality, and complexity of errors.

- *Simplicity versus complexity* – Decision support systems may be trusted differently depending upon the apparent easiness or simplicity of system errors. A recently conducted study (Madhavan et al., 2003) used the following setting: participants performed a target detection task wherein some trials were easy (for the unaided human) and others were difficult (for the unaided human). They did this with the help of a 70% reliable decision support system. Although overall system reliability was equal for all participants, decision support system errors were distributed such that the system exclusively generated either “easy” or “difficult” errors for different groups of participants.

Results revealed that participants using decision support systems that missed targets on “easy” trials mistrusted the aid, misperceived its reliability, and disagreed with the aid more frequently than did participants using a decision support system that generated only difficult errors. A followup study (Madhavan et al., 2004) revealed that the effect of “easy” decision support system errors on trust and dependence was robust even when the errors were only occasional and comprised of false alarms. The results of these studies support the “easy-errors hypothesis” that decision support system errors on tasks easily performed by users undermine user trust in and reliance on decision support systems, even if the system is, on average, statistically more accurate than the average user (Madhavan and Wiegmann, 2007).

- *User ability and system communication*: Trust and decision support system utilization are significantly influenced by the individual user’s ability to perform the task without diagnostic assistance. For example, when users trust

a decision support system that is more reliable than manual (i.e., unaided) performance, they are more likely to rely on the decision support systems than on their own diagnoses. Similarly, when operators distrust a decision support system that is less reliable than manual performance, they are more likely to ignore the system and rely on themselves to diagnose a situation (i.e., self-reliance). In both cases, appropriate reliance occurs (Dzindolet et al., 2003). However, overreliance or misuse can occur when a user overtrusts a decision support system that is less reliable than unaided performance. Conversely, when users undertrust a decision support system that is more reliable than manual performance, underreliance or disuse of the decision support system can occur (Parasuraman and Riley, 1997).

Assessments of manual ability are frequently affected by participants' self-confidence in their own abilities to perform the various tasks without diagnostic assistance. In addition to self-confidence, computer self-efficacy (the perception of one's own ability to use computerized systems) has been recently identified as a stable construct that influences users' willingness to trust and depend on a decision support system (Madhavan and Phillips, 2008).

Self-confidence, in turn, is reportedly influenced by the type of communication from a decision support system. Clear and concise feedback should inform the user what to do in case of a problem (e.g., "System overheated, turn main power switch off!") rather than just highlight the fact that a problem exists (e.g., "System overheated!"). Furthermore, directions should use affirmative terminology (e.g., "Leave the lights on!") and avoid the use of negative terminology (e.g., "Do NOT turn the lights off!") since humans are cognitively less capable of processing negative information than affirmative information.

14.2.3

Relevance for the Engineer

As mentioned before, engineers tend to focus on the functional aspects of a system. The "soft factors" of human sciences are recognized to be important, but often still not fully understood. However, the use of *agent-directed decision support simulation systems* is one of the most challenging applications for ADS. In no other context is the famous insight "models are to think with" or the famous saying "All models are wrong, but some are useful" (credited to George Box) as important as in decision support. No simulation system is a "magic mirror" providing insight into the future. Simulation systems are analytical tools based on the purposeful abstraction of reality with the objective of supporting evaluation of certain sponsor problems.

The cognitive foundations show the need to focus on questions like "How does one present the decision support system to the user?" as well as "How does one present the results of the decision support system to the user?" Both challenges go far beyond the standard solutions for interface design and human-machine in-

terfaces. While such techniques are necessary, they are not sufficient for the successful design or application of decision support system technology. The idea of understanding the *social role* of decision support systems as well as the technical challenges is new for engineers, but crucial for successful decision support system research. If these cognitive foundations are ignored, it is likely that the typical errors of model application for decision support will occur, namely:

- Correct recommendations of the system are ignored as the trust in the solution is too low based on the circumstances under which the solution was presented (like too many false alarms);
- Incorrect recommendations of the systems are applied as the user did not question their validity sufficiently.

14.3

Technical Foundations for Decision Support

After the cognitive foundations have been discussed, this section will focus on the technical foundations for agent-directed decision support and decision support simulation systems. We will focus on two main application cases.

First, agents can be used to select and compose decision support and decision support simulation systems. In this application category, the systems themselves are not changed. They are just well defined regarding their interfaces and execution behavior so that agents can use this information to search for the best tool to solve a given problem. In order to be able to do so, the agent needs to understand the problem, the current environment, and the applicability of available tools.

Second, agents are integrated into the system to improve the realistic behavior of the system. This also allows the system to adapt to new requirements or changing constraints in the environment, as the functionality of the system is now provided by agents that can adapt. However, the use of systems for decision support of real-world operations is the most challenging application, as wrong decisions are often connected with significant harm or loss. Some of the typical characteristics of agenthood that are welcomed in many application domains can stand in the way of successful decision support.

Agent-directed systems will be applied in highly challenging environments, as they shall help to solve complicated real-world problems. Most of these real-world decision-making phenomena cannot be dealt with using a single system, in particular when existing systems have to be reused. In order to support a decision, a set of complementary decision-making models representing multiple perspectives must be composed and orchestrated to describe the whole process. These systems are likely to be characterized by different resolutions and applicability in different phases. Multimodels and multisimulation are topics of current research to deal with this challenge (Yilmaz and Tolk, 2008).

14.3.1

Machine-Based Understanding for Decision Support

In particular, in distributed decision environments, one of the main tasks is to detect and identify applicable tools that can help to support the decision process. This task becomes even more important in net-centric environments, such as enterprises using service-oriented architectures allowing one to recompose preexisting services to deliver required functionality “on the fly” for the user. Intelligent agents are often recommended to support this process. In order to be able to do so, the agent must understand the problem, the environment, and the provided functionality of the tools.

Zeigler (1986) was among the first to publish a model explaining what is necessary for machine-based understanding. Following the lead of Ören and Zeigler, we will use in this chapter the elements shown in Figure 14.3 to identify three requirements:

Perception The first requirement is that the observing system must have a *perception* of the system to be understood. This means that the data and processes characterizing the observed system must be observable and perceivable by the observing system. The properties used for the perception should not significantly differ in scope and resolution from the properties exposed by the system under observation.

Metamodels The second requirement is that the observing system needs to have an appropriate *metamodel* of the observed system. The metamodel is a description of data, processes, and constraints explaining the expected behavior of the observed system. Without such a model of the system, understanding is not possible. Machine-based understanding necessitates identifying and applying the correct metamodel.

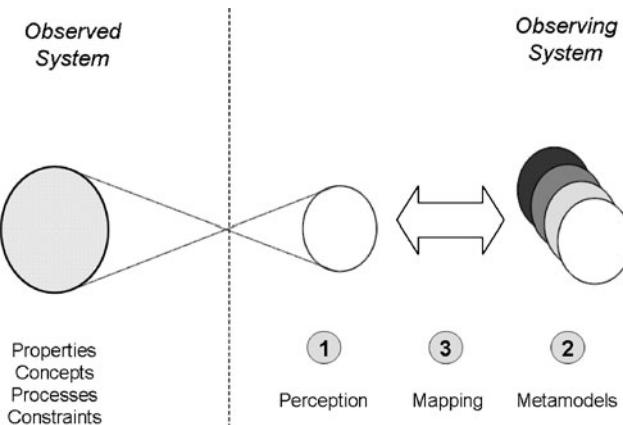


Fig. 14.3 Principles of machine-based understanding.

Mapping The third requirement is that a mapping between observations resulting in the perception and metamodels explaining the observed data, processes, and constraints must exist and be identified.

These requirements are directly applicable to the agent metaphor and how agents gain understanding. Learning – in this context – is the refinement or modification of existing metamodels or the creation of new metamodels. Part of this process must be the mapping of observable properties to these new metamodels. In other words: “What the agent doesn’t know, he cannot understand!” However, it is possible that the agent recognizes a similar metamodel. How to select the best fit and what to do if no metamodel is identified has to be defined by the system engineer.

It should be pointed out, as explained in more detail by Tolk et al. (2008, in print), that the description of data, processes, and constraints can differ in scope, resolution, and structure. *Scope* is the range of propertied concepts that is covered. If one system includes more – or other – concepts than the second, they differ in scope. *Resolution* defines the granularity of the characteristic properties. If two systems differ in resolution, aggregation and disaggregation need to be applied to map their properties to each other, which is normally connected with loss of information. *Structure* is defined by the concepts that are characterized by the properties. It is possible that two systems share the same concepts, but they group them into different structures to describe different concepts. All three challenges can be merged in real-world problems, so that differences in scope, resolution, and structure can occur at the same time, often making it nearly impossible to identify the similarity between two systems.

Nonetheless, in order to enable agents to support decision support in real-world contexts, these problems must be dealt with. This is a topic of current research, although in particular the simulation community identified partial solutions for some challenges.

14.3.2

Requirements for Systems When Being Used for Decision Support

In order for services to be eligible as decision support systems, several requirements need to be fulfilled. This is of particular importance in net-centric environments where users are not necessarily system engineers who are able to evaluate the applicability of a certain tool based on well-informed judgments, but where users need to be able to “trust” the system to be a reliable adviser. Recent reports written by U.S. National Research Councils deal with the particular importance of simulation systems in this regard as well (Council, 2002; N.R.Council, 2006). These reports extend ideas published in Tolk (1999b), which builds the foundation for this subsection as well.

This chapter introduces the use of agent-directed technology for decision support. Of particular interest are *agents for systems* and their two subcategories – *agent-supported systems* and *agent-based systems*, as defined previously.

The following four requirements were identified as belonging to the most important ones, no matter which application domain needed to be supported by the agents. Similar observations were made for military applications, financial and business applications, and others. These four general requirements, which need to be completed by domain-specific requirements and potentially also made more concrete for a domain, are as follows:

Modeling of relevant system characteristics All categories of decision support systems assist the decision-making process based on a model that by itself is a purposeful abstraction of reality. The purpose of the model was to help a scientist or engineer to deal with a sponsor's problem. If this model is applied to support a decision maker, it is necessary that all characteristics of the system that are important to the decision maker be modeled in the system as well. These characteristics comprise data, processes, and constraints.

Validation and verification These are processes to determine the hsimulation credibility. *Validation* is the process of determining the degree to which a model or simulation is an accurate representation of the real world from the perspective of the intended uses. *Validation* determines the behavioral and representational accuracy. Verification is the process of determining that a model or simulation implementation accurately represents the developer's conceptual description and specifications. Verification determines the accuracy of transformation processes. The decision support system as well as the data used should be validated and verified.

Ability to obtain all relevant data Model results are influenced by two factors: the model used and the input data. The quality of inputxanjkood practice to use a common data infrastructure for the operational system as well as the decision support systems to avoid data transformation, in particular when information loss may be the result of the transformation.

Creating situation adequate behavior Agents are often used to integrate better and more realistic behavior to existing systems. It is of crucial importance that the behavior is situation adequate, which means that the metrics used for the internal decision logic and the metrics used for the external evaluation logic are aligned with each other as well as with the supporting model. If the model does not provide the data to support the metrics or if the internal decision logic and the external evaluation logic are not aligned, the results are structural variances that can make the result unusable for decision support, see Tolk et al. (2008).

All four requirements will be evaluated in more detail in general, and regarding agent-directed decision support applications in particular.

- All decision support systems are based on a model of reality. When building the decision support system, it is good practice in systems engineering to define the underlying *conceptual model* and provide documentation thereof. This conceptual model, if the artifacts used to document the model are machine readable and define the system in sufficient detail (including data, processes, and constraints defined by scope, resolution, and structure of the respective concepts), is the basis for evaluating if the first requirement is

fulfilled. Robinson only recently compiled underlying work on conceptual models for simulation systems (Robinson, 2008). His definition is:

The conceptual model is a non-software specific description of the computer simulation model (that will be, is or has been developed), describing the objectives, inputs, outputs, content, assumptions and simplifications of the model.

Robinson argues that the conceptual model establishes a common viewpoint that is essential to developing the overall model. Agents must be able to read and understand the conceptual model. If they do, they can support the selection and composition of tools to ensure that all relevant system characteristics are modeled within the composition. This may require one to cope with challenges of multimodels and multisimulations, which will be dealt with in the next subsection.

- Validation and verification have been recognized to be important. It is now good practice to include them explicitly in project plans and management documents. However, when intelligent agents are used, additional challenges to those being traditionally being recognized emerge literally (Zeigler and Sarjoughian, 2002). *Emergence* has been recognized as an outstanding characteristic of nonlinear dynamic systems in general, and of agent-based systems in particular. Emergence is the unexpected system behavior that cannot be easily explained by the behavior of its components. The very nature of this working definition contradicts verification and validation: if the emergent behavior is unexpected, it can hardly be verified and validated, as it would be no longer unexpected if we could predict it. The validation and verification of underlying components itself, however, is not enough, as the emergence of the system can hardly be explained by the behavior of the components. Verification and validation for agent-directed systems in general – and agent systems in particular – are still the subject of current research. An overview of the current state of the art regarding validation and verification for agent-directed applications is summarized in Yilmaz (2006).
- Decision support systems are used in many application domains. Most of these domains use operational systems to store the relevant data describing operations, resources, project plans, and others. Unfortunately, decision support systems do not always use the same operational data as the hosting or providing systems. They require special formats, aggregates and functional transformation, or other modifications. Before the data can be used, they need to be transformed, and – once a recommendation has been derived – the output data must be transformed back as well.

Even more serious is the fact that operational data are often incomplete, vague, uncertain, and contradictory. Furthermore, they may not have the required accuracy, or the data source itself may not be trustworthy or reliable. If data need to be obtained via the Internet or similar infrastructures, latencies or technical difficulties may be in the way of getting the data in time for the support.

These technical and organizational difficulties need to be taken into account and managed. The NATO Code of Best Practice (NATO, 2002) recommends a Data Engineering Plan that helps to manage activities that are related to these challenges. However, even when a common “language” in the form of a common data model with communication protocols – or other equivalent constructive methods – is used, it is necessary to distinguish between three fundamentally different concepts: *information exchange requirements* is the information that needs to be exchanged from the operational viewpoint; *information exchange capability* are the information that can be produced by the data provider; and *information exchange need* is the information that can be utilized by the data consumer. Theoretically, all three concepts should map to each other. In practice, the differences in scope, resolution, and structure are often significant.

- It is today accepted in the simulation community that intelligent agents can be used to model human behavior better than is the case with other methods. In particular, multiagent simulation is becoming an often applied tool supporting exploring and experimenting with human behavior dynamics. How to incorporate decision science into intelligent agents by viewing decision making as a cognitive reasoning process is documented in Yilmaz and Tolk (2008). What is often neglected is the alignment of model-supported concepts, internal decision logic, and external evaluation logic.

In order to apply logic, the model-supported concepts must be able to support the metrics being used. This is not a trivial task, in particular when existing systems have to be reused that may not have been designed for this purpose. In this case the system engineer must ensure that the necessary measures of efficiency and measures of performance that are operationally required are supported by the system.

While the necessity to support the metrics for internal decision logic and external evaluation logic is well understood, the alignment of the two logics is still not common practice. However, if the internal decision logic pursues different objectives than those that are evaluated by the external logic, *structural variances* may occur (Hawkins, 1984). This leads to nonmonotonocities and counterintuitive results, as often the availability of more resources leads to worse results, which sometimes even demonstrate chaoslike behavior (Tolk, 1999a).

It is therefore good practice to ensure that the behavior generated by the intelligent agents is based on the same evaluation ideas that are used later to find the best solutions. However, as Davis (1992) pointed out, nonmonotonicity is also part of the real world. If real decision makers had perfect knowledge, then they could have better decision algorithms, and if the right metrics were used and the decisions were more nearly continuous, nonmonotonocities would decrease or go away. But in the real world, heuristic decision rules are the best the decision maker often can hope for. They do not have perfect information, much less perfect predictive models. There-

fore, if nonmonotonicities occur, it must be evaluated whether this is due to structural variances or is part of the real problem.

While the use of agent-directed methods can help to generate emergent behavior and more realistic system models, it also introduces new challenges for validation and verification. Also, the introduction of automated or semiautomated behavior without ensuring that this is situation adequate may result in new problems. Finally, data engineering principles need to ensure that data mediation can be applied to obtain the necessary data as well as the required metrics. A holistic overarching systems engineering approach is needed for successful agent-directed applications for decision support.

14.3.3

Agent-Directed Multimodel and Multisimulation Support

This section summarizes the main findings originally published by Yilmaz and Tolk (2008), which have been slightly modified to highlight the applicability for agent-directed applications for decision support.

Models are purposeful abstractions of reality. Complex challenges require the use of several different views or abstractions to cover the full spectrum. This motivates the use of *multimodels*. A multimodel is a modular model that subsumes multiple submodels that together represent the behavior of a complex multiphased decision-making process. A multimodel encapsulates several aspects of reality. As a situation unfolds, the parameters of the decision and payoff matrices, the state space of the problem, the attitudes, and preferences may change. Therefore, the time path of a decision-making process should map onto a time path of decision-making styles embedded within the models. This path becomes the “blueprint” for the multimodel and indicates when to apply which submodel representing the appropriate decision-making process.

Multisimulation is a simulation of several aspects of reality in a study or project. It can include simulation with multimodels, but may also implement variations, such as simulation with multispect models and simulation with multistage models. Such simulations with multimodels allow computational experimentation with several aspects of reality and therefore support better decisions. Multisimulation can be used to branch out multiple simulations, where each simulation uses a specific component configured with an exclusively selected strategy component. Similarly, multiple distinct stages of the problem can be qualified at a given point in time during the simulation by virtue of the evaluation of an updating constraint. In such a case multisimulation enables branching multiple distinct simulations each one of which generates the behavior of a distinct plausible stage within the problem domain.

Intelligent agents can support the process of selecting and composing models and simulations to multimodels and multisimulations. The general architecture for a multisimulation engine as depicted in Figure 14.4 (see details in Chapter 4):

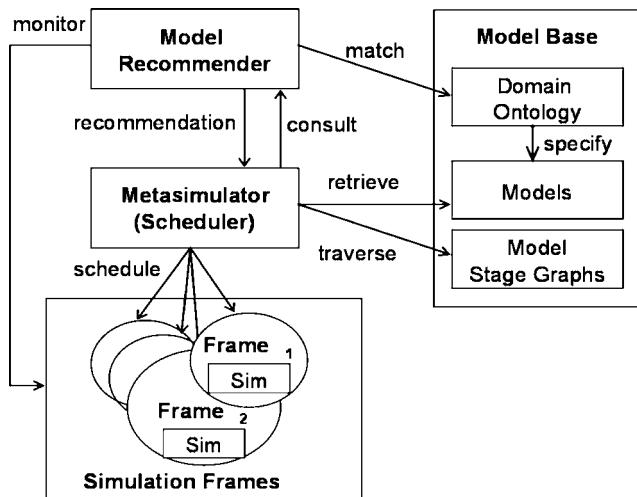


Fig. 14.4 Architecture for a multisimulation engine.

A metasimulator is a scheduler that generates staged compositions of models by traversing the model stage graph and coordinates their simulation and staging within distinct simulation frames. Each frame simulates a distinct subset of models derived from the model stage graph. A recommended model matches trigger and transition conditions to preconditions of applicable models. Given a collection of models, a stage graph can be generated for a decision to be supported. The edges in a model stage graph denote plausible transitions between models as the problem shifts from one stage to another. Once the graph is established, the selected models can be composed.

Each component in this architecture can be represented by an intelligent agent. These agents – who act on behalf of the component they represent – can interact with each other and communicate preconditions, triggers, and other constraints for the model composition. This is still subject to current research, but initial experiments as presented by Yilmaz and Tolk (2008) are promising.

14.3.4

Methods Applicable to Support Agent-Directed Decision Support Simulation Systems

As stated in earlier sections, decision support systems have matured over a number of changing viewpoints on data. The resulting categorization is shown in Table 14.1. Decision support systems were initially generated by computer programmers who attempted to capture the knowledge of subject matter experts in an information management system that could ideally be used to assist management in making decisions without the need of consultation or detailed analysis. The building blocks of such systems were *production rules* in the form of logical expressions, which consolidated the *knowledge* entered into the repository as a set of

Table 14.1 Decision support system domains categorized.

Category	Domain	Example
Data Mining	Data driven	Lists, databases, and DMS
Constraints	Rule-based systems	Expert- and knowledge-based systems
Temporal	Case-based systems	Reasoning- and analogy-based systems
Thought	Neural networks	Learning and pattern matching
Symbolic	Fuzzy logic	Transform ambiguity (crisp to fuzzy)
Evolutionary	Genetic algorithms	Optimization and search spaces
Inductive	Machine learning	Iterative creation of dynamic rule sets

inferences. For example, Mycin was an early commercial version of a decision support system using an expert system as its source of knowledge/inference (Jensen, 1997). In general, the knowledge bases consist of expertise related to a particular domain. Humans are not always precise in their behavior and thus fuzzy logic can be incorporated for mimicking the aspects of human cognition. In some situations, the knowledge is incomplete and may not even exist. In these situations, neural networks can help by analyzing large quantities of data to establish patterns and characteristics related to a given scenario.

To summarize these ideas, the table shows selected methods applicable to support agent-directed decision support simulation systems and can be used in support of all components, in particular agents and simulations. The methods are listed against each category as interpreted by the authors, as they obviously overlap to a certain degree, and alternative categories can be applied.

The methods captured in Table 14.1 can be described as follows:

- *Data-driven methodologies* are based on data mining techniques. They evolved with lists, databases, and data warehousing, which developed into a domain of research that is still called data mining. The inputs are still based on a list of related facts (even when held in databases) that need to be associated and queried prior to being interpreted.
- *Rule-based systems* use a combination of knowledge-based systems, working memory, and rule interpreters to discriminate between a collection of facts represented as evidence in knowledge bases and queries input by operators. Forward chaining (branching down the tree) and backward chaining (evidence matching up through working memory) algorithms are applied.
- *Case-based reasoning* systems take advantage of previous attempts to solve a problem or improve the accuracy of the system. Training is gained using a predefined data set and matured using new patterns as time passes. Reasoning is generated using a series of measures that can be used to describe the situation within a scenario. Cases are determined using a pattern matching approach of analysis.

- *Neural networks* simulate cognitive approaches to learning using trial and error. The method involves the creator maintaining a prediction loop from which comparisons and an adjustment process can be used to reconcile transfer functions, internal weightings, and swagging functions. Forward and backward chaining neural networks have been created to minimize overall network errors. This is another approach for heuristic optimization.
- *Fuzzy logic* has been introduced to deal with vague data. The problem with many rule-based systems is that they don't approximate vague data like humans. When baking, for instance, a recipe may call for 1/2 cup of milk. Does that mean exactly 125 milliliters, 100 milliliters, or 150 milliliters? Most of the time humans measure against a range and minor errors are acceptable, which makes commands like "take a sharp left turn" actionable without the need for accurate values for humans, but hard to follow for robots. Fuzzy logic transforms the membership of data from *crisp sets* to *fuzzy sets*, making data themselves fuzzy.
- *Genetic algorithms* are a style of data processing that emerged as a result of an excessive number of parameters. The exponential growth of combinatorial statistics called for a new approach to deriving answers. The use of Darwinian theory relating to "survival of the fittest" concept was proven to efficiently find the optimal or best solution of a given probable subset and has been adopted to solve many search space problems. This approach is often combined with another method for heuristic optimization, namely, *simulated annealing*.
- *Machine learning* refers to frequently applied operations research and heuristics (but is not limited to them). Rules are recursively created and connected using trees to represent possible outcomes, based on statistical patterns and algorithms. Tic-tac-toe and chess use good examples of branching and pruning to derive a number of possible results (end-means). However, other means of artificial intelligence used to support inductive and deductive learning algorithms are important as well.

These methods can and have been applied alone or combined with each other. It is, for example, good practice to "fuzzify" rules by making the input and output sets fuzzy sets to broaden their applicability, see among others Driankov et al. (1993). Another example for good practice is the application of machine learning and heuristic optimization to improve rule-based approaches.

Due to the constraints of this chapter, it is not possible to include all relevant methods and related relevant research. The selection of the authors does not necessarily reflect the general importance of the research.

14.4

Examples for Intelligent and Agent-Directed Decision Support Simulation Systems

The objective of this final section of the chapter is to give a short and selected overview of the currently applied state of the art in application and research. Industry and other users have begun collecting data since the introduction of computers. Archives have grown exponentially requiring new technologies to find relevant data and present them in understandable form. In addition, collaboration between distributed partners required alignment and orchestration of decisions of overarching importance. In the forward to an overview book on state-of-the-art applications and research, the following evolution of systems is enumerated (Gupta, 2005): data-oriented systems, management expert systems, multidimensional data analysis, query and reporting tools, online analytical processing (OLAP) tools, group decision support systems, conferencing/groupware, and distributed document management systems. All these systems contribute to making better decisions in distributed environments. As industry continuously strives to maintain the competitive edge, many information management systems have been developed to transform raw facts and figures into information from which knowledge can be derived that becomes the basis for awareness. Researchers have attempted all manner of collection and assessment methodologies. Agent-directed systems contribute to this research (Tweedale et al., 2007).

14.4.1

Supporting Command and Control

In addition to this commercial applications, the military is active in this domain as well. The advantages of the military domain is that the decision-making process has been formalized and is well known to all military decision makers, as it is part of the education on all levels of command. The military decision process is divided into four phases: *observe*, *orient*, *decide*, and *act*. As the act phase is followed by observing – initializing the next set of decisions – this process is normally referred to as the OODA loop (Boyd, 1987).

- In the observe phase, the decision maker acquires information from the environment, usually through sensors or their stimuli.
- In the orientation phase, the decision maker builds his perception and understanding of the current situation. This includes the degree to which the former decisions were accomplished, if there are gaps between plan and real situation, and other operationally relevant questions. This normally leads to a number of alternative courses of action.
- These alternatives are then evaluated. The most promising alternative is selected. This is the decision phase.
- The decision leads to a plan and resulting actions. These can be actions on the battlefield (down to base options, such as shoot, look, move, and communicate) or the generation of orders for subordinated units.

This well-structured decision-making process sets a good framework for decision support. A general decision support architecture for military applications is described in Tolk (2005).

Traditionally, OODA was not designed to structure and describe the military decision process and did not focus on coordination and cooperation explicitly (Grant, 2005). However, models of OODA have been extended to integrate the two abilities. One research direction involved integrating the decision processes within teams. Within the concept of shared situational awareness, team members share knowledge where decisions can be made (Keus, 2002). To this end, the stimulate–orient–decide–act (SODA) loop was introduced to show how cooperation can actually be implemented. In this approach, cooperation is a result of sharing the stimulation and orientation phases, while sharing the deciding and acting phases represents coordination. The implementing agents follow the beliefs–desire–intentions paradigm explained earlier in this book (Rao and Georgeff, 1991). The mental models used by these agents result from common beliefs about the environment based on the shared observation of the environment represented by shared stimulation. The desires enable the orientation by creating a common perception of the environment. Figure 14.5 depicts these ideas for decision support in the command and control environment.

In summary, decision support systems have a relatively long history in the military domain. More examples are given by Pohl and colleagues presenting the result of a project sponsored by the Defense Advance Project Research Agency (DARPA). The Integrated Marine Multi-Agent Command and Control System (IMMACCS) is a multiagent, distributed system (Pohl et al., 1999). IMMACCS is designed to provide a common tactical picture and was one of the first military adapters of the agent-directed paradigm for decision support. Between 1999 and 2004, the US Of-

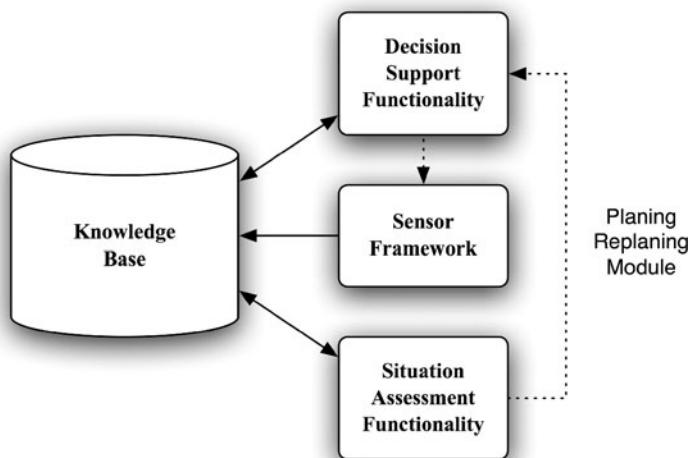


Fig. 14.5 SODA/C4ISR model.

fice of Naval Research (ONR) sponsored a series of workshops on decision support systems. Today, many applications are documented in workshops on ADS as well as new technologies for command and control support and training and education.

14.4.2

Supporting Inventory Control and Integrated Logistics

Inventory control is a common problem for companies, large or small. Issues such as quality control, supplier performance, and economic risk all contribute to image and profit. The big difference is that individual companies place priorities according to rules that govern its operation. These rules need to be embedded as operational constraints within any decision support system. It is possible to furthermore incorporate minimum and maximum attributes to monitor resupply or patronage habits via countertransactions. The reports already are used by management, shareholders, and other stakeholders for their decision. Such reports can summarize sales, supplier deliveries, customer trends, and suggested changes to menus, operating hours, and other factors affecting the business model.

Concrete examples for decision support use in this environment are *integrated logistics systems* (ILSs) as defined for the military domain in DoD (2001). In this document, ILS is defined within the various stages and phases of a product design development, very similar to the constraints generally defined for multimodel and multisimulation support described earlier. As such, it also incorporates the role of ILS to fit the bigger picture of systems engineering where the considerations for future support and maintainability of the system is embedded within the system design development. The ILS forms an integral part of system readiness, maintainability and its field supportability. In general, ILSs have been defined as a unified and structured approach to effectively and efficiently plan and manage the resources required to maintain operational readiness (Eisner, 2008). Parts of the ILS paradigm are the key concepts of maintenance planning, supply support, technical data, design interfaces, database management systems, and support equipment.

In today's ILS environment, companies utilize their proprietary or utility-specific integrated tools to automate the links between logistic management systems, maintenance management systems, and engineering management systems. These specific suits are also known as maintenance management information systems. One such system that is used by leading defense companies is the *Enhanced Automated Graphical Logistics Environment* (EAGLE) (Raytheon, 2007). These automation packages are capable of providing a single platform to meet all the requirements of maintainers, suppliers, and engineers. These tools provide a comprehensive basis for configuration management on a single operating platform. In order to support the overall process, various types of maintenance are performed on repairable items in terms of preventive or corrective maintenance (Langford, 2007). This maintenance is either performed through manual testing techniques or through automated test equipment, which is already an established maintenance practice around the world. Regardless of the maintenance technique being used,

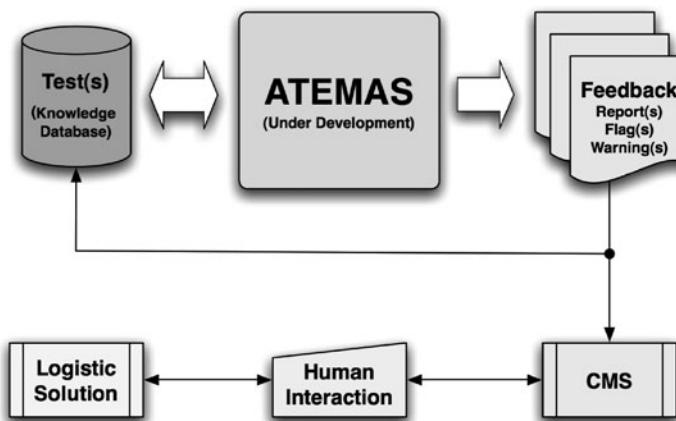


Fig. 14.6 Automated test equipment multi-agent system (ATEMAS).

the maintenance data are fed into one of the management information systems being used.

Whereas existing tools like EAGLE and other similar management information system softwares have a huge potential to serve as very effective knowledge bases due to their inherent powerful and comprehensive data logging capabilities, these tools are not intelligent and therefore are not capable of predicting component reliability and, in turn, their future likely obsolescence. This gap can be closed using intelligent agents that fulfill these tasks currently mainly supported by human experts. The resulting initial concept demonstration for an *automated test equipment multi-agent system* (ATEMAS) is depicted in Figure 14.6.

The architecture defined for ATEMAS can be generalized into an agent-directed decision support system that monitors the environment, logging data, and generating inferences based on the analysis of data stored in its repository. The system is event-driven and operates using a feedback mechanism. The different activities this model performs include:

- Acquiring different inputs from various systems and subsystems;
- Carrying out computations on the available information to construct a meaningful picture to trigger some decisions;
- Generating output information in terms of certain actions based on the formulations conducted within the system. These outputs can be warnings, indications, flags, and various other relevant actions;
- Generating regimes and feedback mechanisms to improve data that can then be used to make informed decisions.

A *collection* agent is teamed with a *refinement* agent in an agent-directed architecture to achieve functionality. These agents interact with the environment in an independent way while interacting with each other to collect data and convert reliable information into knowledge.

Collection agent This agent is the primary tool for collecting all relevant data from the environment. Some of the sources required to obtain pertinent maintenance test data from the ATE include previous trends for usage, failure, and defects on a particular component. It also captures the data pertaining to reliability, and OEM specifications in terms of mean time between failures (MTBF) as well as statistical history. It should be noted that the data captured by the collection agent (CA) at this stage is only currently in blackboard form.

Refinement agent The conversion of data into meaningful information is the job of the second agent, the refinement agent (RA) in the ATEMAS IDSS model. The purpose of the refinement agent is to create useful information which will be used to make a database also known as a KB.

Feedback agent The third and most critical agent in the system operates as a central point for information processing and transforming into references that can be used to assist decision making. The job of this agent is to process the information to provide feedback in terms of messages, warnings, flags, and modification or configuration change requests.

This methodology can improve the overall maintainability and reliability predictions by statistical analysis of all available information and prevailing operational circumstances. It is proposed that the recommended central feedback agent will interact with the system at different stages and levels including, information repository, comparison matrices, failure and servicing records database, and more. Based on the information, this agent will be capable of redefining the parameters and specification for carrying out the test more efficiently. Supporting methods are Bayesian, fuzzy logic, or neural networks (Jensen, 1997). The block diagram of the resulting architecture for an *intelligent decision support system* (IDSS) – a multiagent system – is depicted in Figure 14.7.

Within the developed IDSS improving the ATEMAS ideas, four agents were used, namely, MMS, LMS, EMS, and inference agent:

- Three agents act as information collection agents that continually observe the environment for a potential change and report any such change back to the inference agent. These collection agents also process information and convert it into knowledge (as a set of beliefs about their perceived environment). They access maintenance management systems (MMSs), logistic management systems (LMS), and engineering management systems (EMS).
- The inference agent accesses the expert knowledge in terms of production rules and continually interprets the knowledge created and updated by the collection agents.

In the developed IDSS, each agent is developed to continually monitor the assigned environment. Each agent consists of one or more plans to execute certain logical operation based on particular events or based on its own beliefs. The three collection agents interact with their environment which consists of information stored on various diverse independent information storage platforms and information

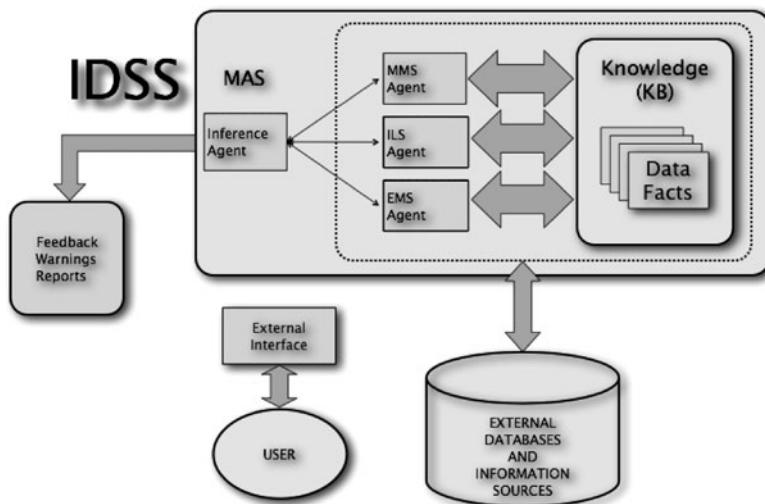


Fig. 14.7 Architecture for an intelligent decision support system.

servers. Furthermore, the IDSS contained a user interface to enable user-requested analysis. The outputs of the IDSS are the generated reports, warnings, and other relevant feedback generated about the current state of the system.

While developing the IDSS, platform independence and reduced development time were achieved using commercial off-the-shelf (COTS) software. Two software packages were selected to implement the IDSS:

- Java (Gosling and McGilton, 1995) was chosen as a development language because it is portable, platform independent, and has extensive development support.
- Due to the lack of immediate Java-related classes, the JACK framework was used to implement the software agents (A.O.S., 2006).

14.5

Conclusion

Efficient and consistent decision making in today's complex environment is not an easy task without decision support tools. Tools such as the IDSS have supported military decision making in a complex environment characterized by high uncertainty and rapidly changing conditions. The architectures have been developed for various military applications to demonstrate possible advances of such systems. A number of these applications include:

- Command and Control (C^2) (Tolk, 2005) enhancements to assist military commander in planning time-critical tasks.
- Improving situational awareness (Eyekon) (Hicks et al., 2002) using a soldier's wearable computer and thus reducing the cognitive overload for the

soldier. This IDSS needs to operate in real time and provide support to the individual soldier and to the team.

- MokS AF (Lenox et al., 2000), a software system that supports mission-critical team decision making and provides a virtual environment for route planning and team coordination allowing commanders to design new scenarios, plan individual routes to a common rendezvous point, communicate synchronously across great distances, negotiate the selection of platoon units, and plan joint missions via a shared virtual environment.
- Various other applications are being developed and pursued by the Defense Advanced Research Projects Agency (DARPA), which comes under the US Department of Defense (DoD). The Information Processing Technology Office (IPTO) is presently developing the Advanced Soldier Sensor Information System and Technology (ASSIST) based on its *bootstrapped learning*, which is stimulated by the Coordination Decision Support Assistants (COORDINATORS) (Wagner, 2007).
- Impact Technologies LLC USA (Barlas et al., 2005) has presented a self-evolution process in a knowledge base to handle data organization to improve maintenance information and to remove incorrect or inefficient information retrieval and to handle removal of unnecessary data with the vision for an asset-readiness decision-making system.
- Research into development of an IDSS in defense maintenance methodologies using a multiagent system is being conducted and an introductory paper describing the blackboard conceptual model has already been published in proceedings of the IEEE (Haider et al., 2006).
- Daimler Chrysler AG, Germany (Junghanns and Tatar, 2006), has a study focused on replacing its manual wiring process with an automated tool to increase productivity on its Airbus final assembly line.

The use of agents for decision support belongs to the most challenging application domains for agents and as such requires rigorous application of systems engineering methods. This chapter summarized the need to know the cognitive as well as the technical foundations. Many topics are still the subject of current research and more breakthrough papers and applications can be expected.

References

- D.S. Alberts and R.E. Hayes. *Power to the Edge: Command and Control in the Information Age*. Command and Control Research Program Press, Washington, DC, 2003.
- A.O.S. JACK intelligent agents teams manual 4.1. [Online, accessed 2nd March] <http://www.agent-software.com.au/shared/resources/index.html>, March 2006.
- B. Barber. *The Logic and Limits of Trust*. Rutgers University Press, New Brunswick, NJ, 1983.
- I. Barlas, A. Ginart, and J.L. Dorrity. Self-evolution in knowledge bases. In *Autotest-con*, pp. 325–331. IEEE, Sept 26–29 2005.
- D.E. Bell, H. Raiffa and A. Tversky. *Decision Making: Descriptive, Normative, and Pre-*

- scriptive Interactions*. Cambridge University Press, Cambridge, CA, 1988.
- K. Blomqvist. The many faces of trust. *Scandinavian Journal of Management*, 13(3), 85–97, 1997.
- C.A. Bowers, R.A. Oser, E. Salas and J.A. Cannon-Bowers. Team performance in automated systems. In M. Mouloua R. Parasuraman, (ed.), *Automation and Human performance: Theory and Application*, pp. 243–263, Mahwah, NJ, 1996. Lawrence Erlbaum Associates.
- C.A. Bowers, F. Jentsch, E. Salas and C.C. Braun. Analyzing communication sequences for team training needs assessment. *Human Factors*, 40(4), 672–680, 1998.
- R.J. Boyd. Slide series: A discourse on winning and losing. Technical Report M-U 43947, Maxwell Airbase, August 1987, 1987.
- S. Breznitz. *Cry Wolf: The Psychology of False Alarms*. Lawrence Erlbaum, Hillsdale, NJ, 1983.
- N. Cotte, J. Meyer and J.F. Coughlin. Older and younger drivers' reliance on collision warning systems. In *Human Factors and Ergonomics Society Annual Meeting Proceedings*, volume 45, pp. 277–280, 2001.
- National Research Council. *Modeling and Simulation in Manufacturing and Defense Acquisition: Pathways to Success. Committee on Modeling and Simulation Enhancements for 21st Century Manufacturing and Defense Acquisition*. National Academies Press, Washington, DC, 2002.
- P.K. Davis. Dynamic instability. *PHALANX – Military Operations Research Society*, (12/92), 1992.
- V. Dhar and R. Stein. *Intelligent decision support methods: the science of knowledge work*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1997.
- J.J. Dijkstra. User agreement with incorrect expert system advice. *Behaviour and Information Technology*, 18(6), 399–411, 1999.
- DoD. Mil-std-61: Configuration management. Military standards handbook, US Department of Defence, HQ AFMC//ENS, Ohio, USA, 2001. DoD. Mil-std-61: Configuration management. Military standards handbook, US Department of Defence, HQ AFMC//ENS, Ohio, USA, 2001.
- D. Driankov, H. Hellendorf and M. Reinfrank. *An Introduction to Fuzzy Control*. Springer-Verlag, Berlin Heidelberg, 1993.
- M.T. Dzindolet, L.G. Pierce, H.P. Beck, L.A. Dawe and B.W. Anderson. Predicting misuse and disuse of combat identification systems. *Military Psychology*, 13(3), 147–164, 2001.
- M.T. Dzindolet, S.A. Peterson, R.A. Pomranky, L.G. Pierce and H.P. Beck. The role of trust in automation reliance. *International Journal of Human-Computer Studies*, 58(6), 697–718, 2003.
- E.H. Edersheim. *The Definitive Drucker: The Final Word from the Father of Modern Management*. McGraw-Hill, New York, NY, 2006.
- H. Eisner. *Essentials of Project and Systems Engineering Management*. Series in Engineering and Technology Management. John Wiley & Sons, 3rd. edition, 2008.
- J. Gosling and H. McGilton. The java language environment: A white paper. *Sun Microsystems*, Mountain View, CA, 1995.
- T. Grant. Unifying planning and control using an OODA-based architecture. In *SAICSIT 2005 – Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists*, White River, South Africa, 2005. ACM.
- D.N.J. Gupta. Forward to the book. In G.E. Phillips-Wren and L.C. Jain, (eds), *Intelligent decision support systems in agent-mediated environments*, volume 115 of *Frontiers in Artificial Intelligence and Applications*, pages v – vi. IOS Press, Amsterdam, 2005.
- N. Gupta, M.A. Bisantz and T. Singh. Investigation of factors affecting driver performance using adverse condition warning systems. In *Human Factors and Ergonomics Society Annual Meeting Proceedings*, volume 45, pp. 1699–1703, 2001.
- K. Haider, J. Tweedale, P. Urlings and L. Jain. Intelligent decision support system in defense maintenance methodologies. In *International Conference of Emerging Technologies (ICET '06)*, Peshawar, pp. 560–567. IEEE Press, New York, 13–14 Nov 2006.
- G. Hawkins. Structural variance and other related topics in the shape armour/anti-armour study. In R.K. Huber, (ed.), *Systems Analysis and Modeling in Defense – Development, Trends, and Issues*, New York, NY, 1984. Plenum Press.

- P. Herbig and J. Milewicz. The relationship of reputation and credibility to brand success. *Journal of Consumer Marketing*, 10, 18–24, 1993.
- J. Hicks, R. Flanagan, P. Petrov and A. Stoyen. Taking agents to the battlefield. In *second international workshop on formal approaches to agent-based systems, Greenbelt, MD, USA*, volume 2699 of *Lecture Notes in Computer Science*, pp. 220–232, Germany, October 29–31 2002. Springer.
- P. Hwang and W.P. Buergers. Properties of trust: An analytical view. *Organizational Behavior and Human Decision Processes*, 69(1), 67–73, 1997.
- F.V. Jensen. *An Introduction to Bayesian Networks*. Springer Berlin, Heidelberg, London, UK, 1997.
- B. Jones. *Sleepers, Wake!: Technology and the Future of Work*. Oxford University Press, Inc., New York, USA, 2nd. edition, 1990.
- A. Junghanns and M. Tatar. Diagnosing highly configurable products: Troubleshooting support for airbus final assembly line. In G. Brewka, S. Coradeschi, A. Perini and P. Traverso, (eds), *ECAI 2006 – 17th European Conference on Artificial Intelligence, 2006, Riva del Garda, Italy*, vol. 141 of *Frontiers in Artificial Intelligence and Applications*, pp. 637–641, Amsterdam, The Netherlands, August 29–September 1 2006. IOS Press.
- C. Keating, R. Rogers, R. Unal, D. Dryer, A. Sousa-Poza, R. Safford, W. Peterson and G. Rabadi. System of systems engineering. *Engineering Management Journal*, 15(3), 2003.
- E.H. Keus. A framework for analysis of decision processes in teams. In *CCRP – 7th International Command and Control Research Technology Symposium*, Monterey, CA, 2002.
- G. Klein, D.D. Woods, J.M. Bradshaw, R.R. Hoffman and P.J. Feltovich. Ten challenges for making automation a team player in joint human-agent activity. *IEEE Intelligent Systems*, 4, 1541–1672, 2004.
- J.W. Langford. *Logistics: Principles and Applications*. Logistics Series. McGraw-Hill, New York, N.Y, 2nd. edition, 2007.
- J.D. Lee and N. Moray. Trust, control strategies and allocation of function in human machine systems. *Ergonomics*, 22(6), 671–691, 1992.
- J.D. Lee and N. Moray. Trust, self-confidence, and operators' adaptation to automation. *International Journal of Human-Computer Studies*, 40, 153–184, 1994.
- J.D. Lee and K.A. See. Trust in automation: Designing for appropriate reliance. *Human Factors*, 46(1), 50–80, 2004.
- T.L. Lenox, M. Lewis, S.K. Hahn, T. Payne and K. Sycara. Task characteristics and intelligent aiding [route-planning tasks]. In *IEEE International Conference on Systems, Man, and Cybernetics*, vol. 2, pp. 1123–1127. IEEE Press, New York, October 2000.
- F. Javier Lerch, M.J. Prietula and C.T. Kulik. The turing effect: the nature of trust in expert systems advice. *Expertise in context: human and machine*, pp. 417–448, 1997.
- D.J. Lewis and A. Weigert. Trust as a social reality. *Social Forces*, 63, 967–985, 1985.
- N. Luhmann. *Trust and Powers*. John Wiley & Sons Inc, 1979.
- N. Luhmann. Familiarity, Confidence, Trust: Problems and Alternatives. *Trust: Making and Breaking Cooperative Relations*, pp. 94–107, 1988.
- P. Madhavan and R.R. Phillips. Is ignorance bliss? In *Proceedings of the 8th Annual Symposium on Human Interaction with Complex Systems*, Norfolk, VA, 2008.
- P. Madhavan and D.A. Wiegmann. Effects of information source, pedigree, and reliability on operator interaction with decision support systems. *Human Factors*, 49(5), 773–785, 2007.
- P. Madhavan, D.A. Wiegmann and F.C. Lackson. Automation failures on tasks easily performed by operators undermines trust in automated aids. In *Proceedings of the 47th Annual Meeting of the Human Factors and Ergonomics Society*, Santa Monica, CA, 2003. Human Factors & Ergonomics Society.
- P. Madhavan, D.A. Wiegmann and F.C. Lackson. Occasional automation failures on easy tasks undermines trust in automation. In *Proceedings of the 112th Annual Meeting of the American Psychological Association*, Honolulu, HI, 2004.
- M. Maltz and D. Shinar. New alternative methods of analyzing human behavior in cued target acquisition. *Human Factors*, 45(2), 281–295, 2003.
- A. J. Masalonis and R. Parasuraman. Effects of situation-specific reliability on trust and

- usage of automated air traffic control decision aids. In *Proceedings of the Human Factors and Ergonomics Society 47th Annual Meeting*, pp. 533–537, 2003.
- B.M. Muir. Trust between humans and machines, and the design of decision aids. *International Journal of Man-Machine Studies*, 27, 527–539, 1987.
- B.M. Muir. Trust in automation: Part i. theoretical issues in the study of trust and human intervention in automated systems. *Ergonomics*, 37(11), 1905–1922, 1994.
- C. Nass and K.M. Lee. Does computer-synthesized speech manifest personality? experimental tests of recognition, similarity-attraction, and consistency-attraction. *Journal of Experimental Psychology: Applied*, 7(3), 171–181, 2001.
- C. Nass and Y. Moon. Machines and mindlessness: Social responses to computers. *Journal of Social Issues*, 56(1), 81–103, 2000.
- C. Nass, J. Steuer, E. Tauber and H. Reed-er. Anthropomorphism, agency, and ethopoeia: computers as social actors. In *CHI '93: INTERACT '93 and CHI '93 conference companion on Human factors in computing systems*, pp. 111–112, New York, NY, USA, 1993. ACM.
- C. Nass, J. Steuer and E. R. Tauber. Computers are social actors. In *CHI '94: Proceedings of the SIGCHI conference on Human factors in computing systems*, pp. 72–78, New York, NY, USA, 1994. ACM.
- C. Nass, Y. Moon, B.J. Fogg, B. Reeves and D.C. Dryer. Can computer personalities be human personalities? *International Journal of Human-Computer Studies*, 43, 223–239, 1995.
- C. Nass, B.J. Fogg and Y. Moon. Can computers be teammates? *International Journal of Human-Computer Studies*, 45, 669–678, 1996.
- NATO. *NATO Code of Best Practice for Command and Control Assessment*. Command and Control Research Program, Washington, DC, 2002.
- N.R. Council. *Defense Modeling, Simulation, and Analysis: Meeting the Challenge*. National Academies Press, Washington, DC, 2006.
- R. Parasuraman and V. Riley. Humans and automation: Use, misuse, disuse, abuse. *Human Factors*, 39(2), 230–253, 1997.
- R. Parasuraman, R. Molloy and L.L. Singh. Performance consequences of automation-induced “complacency”. *The International Journal of Aviation Psychology*, 3, 1–23, 1993.
- J. Pohl. Intelligent software systems in historical context. In G.E. Phillips-Wren and L.C. Jain, (eds), *Intelligent decision support systems in agent-mediated environments*, volume 115 of *Frontiers in Artificial Intelligence and Applications*, pp. 3–34. IOS Press, Amsterdam, 2005.
- J.G. Pohl, A.A. Wood, J.K. Pohl and J.A. Chapman. Immaccs: A military decision-support system. In *Proceedings of the DARPA-JFACC Symposium on Advances in Enterprise Control*. Defense Advanced Research Project Agency, 1999.
- A.S. Rao and M.P. Georgeff. Modeling rational agents within a BDI-architecture. In James Allen, Richard Fikes, and Erik Sandewall, (eds), *Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning (KR'91)*, pp. 473–484. Morgan Kaufmann, 1991.
- Raytheon. Enhanced automated graphical logistics environment. *Raytheon EA-GLE, Tucson, Arizona, USA*, 2007. URL <http://raytheonagle.com>.
- B. Reeves and C. Nass. *The media equation: how people treat computers, television, and the new media like real people and places*. Stanford University Press, Center for the Study of Language and Information, Stanford, CA, 1996.
- S. Robinson. Conceptual modelling for simulation part i: definition and requirements. *Journal of the Operational Research Society*, 59, 278–290, 2008.
- T.B. Sheridan. *Humans and Automation: System Design and Research Issues*. John Wiley & Sons, Santa Monica, CA, 2002.
- J.F. Sowa. *Knowledge representation: logical, philosophical, and conceptual foundations*. Brooks/Cole, Pacific Grove, CA, 2000.
- F.W. Taylor. *The Principles of Scientific Management*. Harper & Row, New York, N.Y., 1911.
- L.C. Thomas, C.D. Wickens and E.M. Rantanen. Imperfect automation in aviation traffic alerts: A review of conflict detection algorithms and their implications for human factors research. In *Proceedings of the 47th Annual Meeting of the Human Factors*

- and Ergonomics Society, Santa Monica, CA, 2003.
- A. Tolk. An agent-based decision support system architecture for the military domain. In G. Phillips-Wren and L.C. Jain, (eds), *Intelligent decision support systems in agent-mediated environments*. IOS Press, Netherlands, pp. 187–205, 2005.
 - A. Tolk. Non-monotonicities in HLA federations. In *Proceedings of the Spring Simulation Interoperability Workshop*, volume I, pages 1–7, Orlando, FL, 1999a. IEEE CS Press.
 - A. Tolk. Requirements for simulation systems when being used as decision support systems. In *Proceedings Fall Simulation Interoperability Workshop*, pp. 29–35. IEEE CS Press, 1999.
 - A. Tolk, S.Y. Diallo and C.D. Turnitsa. Implied ontological representation within the levels of conceptual interoperability model. *International Journal of Intelligent Decision Technologies*, 2(1), 3–19, 2008.
 - A. Tolk, S.Y. Diallo, R.D. King and C.D. Turnitsa. A layered approach to composition and interoperation in complex systems. In A. Tolk and L.C. Jain, (eds), *Complex Systems in Knowledge-based Environments*, Springer-Verlag Berlin Heidelberg, 2008, in print.
 - J. Tweedale, N. Ichalkaranje, C. Sioutis, P. Urlings, G. Phillips-Wren and L.C. Jain. Building a decision making framework using agent teams. In *Intelligent Decision Technologies*, volume 1(4). IOS Press, Amsterdam, The Netherlands, 2007.
 - T. Wagner. Coordination decision support assistants. Solicitation Report BAA-04-029, DARPA, Arlington, VA, 2007. T. Wagner. Coordination decision support assistants. Solicitation Report BAA-04-029, DARPA, Arlington, VA, 2007.
 - C.D. Wickens. Multiple resources and performance prediction. *Theoretical Issues in Ergonomics Science*, 3, 159–177, 2002.
 - D.A. Wiegmann, A. Rich and H. Zhang. Automated diagnostic aids: The effects of aid reliability on users' trust and reliance. *Theoretical Issues in Ergonomics Science*, 2(4), 352–367, 2001.
 - L. Yilmaz. Validation and verification of social processes within agent-based computational organization models. *Computational & Mathematical Organization Theory*, 12(4), 283–312, 2006.
 - L. Yilmaz and T.I. Ören. Agent-directed simulation systems engineering. In *SCSC: Proceedings of the 2007 summer computer simulation conference*, pp. 897–904, San Diego, CA, USA, 2007. Society for Computer Simulation International.
 - L. Yilmaz and A. Tolk. A unifying multimodel taxonomy and agent-supported multisimulation strategy for decision-support. In G. Phillips-Wren, N. Ichalkaranje and L. Jain, (eds), *Intelligent Decision Making: An AI-Based Approach*, volume 97 of *Studies in Computational Intelligence*, pp. 193–226, Springer-Verlag Berlin Heidelberg, 2008.
 - B.P. Zeigler. Toward a simulation methodology for variable structure modeling. In M.S. Elzas, T.I. Ören, and B.P. Zeigler, (eds), *Modeling and Simulation Methodology in the Artificial Intelligence Era*, Amsterdam, 1986. North Holland.
 - B.P. Zeigler and H.S. Sarjoughian. Implications of M&S foundations for the V&V of large scale complex simulation models. In *Proceedings of the Foundations for V&V in the 21st Century Workshop*. Johns Hopkins University Applied Physics Laboratory, 2002.

15

Agent Simulation for Software Process Performance Analysis

Levent Yilmaz and Jared Phillips

15.1

Introduction

Software systems engineering, like systems engineering, involves both a technical and management process. While software engineering focuses on the analysis, design, coding, testing, and integration of software, software systems engineering involves problem definition, solution analysis, process planning, process control, and product evaluation. Hence, software systems engineering is not a task with a specific job description, but rather a process. The technical process component focuses on tasks and analytical effort needed to transform an operational need into various artifacts such as

- A formal or semiformal description of the software system,
- A software analysis and design consistent with the requisite configuration, size, and quality,
- Documentation requirements and design specifications,
- The guidelines and procedures to instill confidence in the operation of the software system via proper verification and test methods, as well as criteria to accept the completed product,
- The documentation to help users operate and maintain the system.

Software processes entail a coherent set of policies, procedures, and technologies that are used within an organizational structure to produce and maintain software products (Yilmaz and Phillips, 2006). The process involves knowledge acquisition activity, during which teams of engineers collaborate and coordinate within the constraints imposed by the management, as well as organizational norms, technology, culture, and policies. The human activity is at the core of the software development practice, as decisions and control actions are not taken by organizations or systems, but rather by a number of decision makers carrying out their activities at various levels and locations within the work system (Constantine, 1993) that involves people engaging in activities over time.

Existing simulation-based exploration of dynamics of software processes often involve focusing on flows of products and data through the process using discrete-event and continuous models (Abdel-Hamid and Madnick, 1991). However, the lack of conceptualization and incorporation of strategic change, adaptive human, team, organizational, and cultural factors into models of software processes pose special problems.

- *Software processes are goal-directed and adaptive:* In adaptive software development organizations goals and constraints are often implicit and embedded in the work practice and norms associated with the organization. Goals vary, requirements change, and employee turnover is common. Hence, in a dynamically changing environment, an effective organizational work depends on self-organizing and adaptive mechanisms that are in place to change properties of the process to meet the current needs.
- *Processes improve over time:* In software development, management frequently modifies the structure and mechanisms of the process to keep some measure (e.g., defect density) related to the relevant performance objective (e.g., reliability) near an optimum. Control of adaptation, however, is distributed across all teams, engineers, and organizational subsystems. A useful and credible model for the analysis of the process and prediction of responses to changes in the circumstances must reflect the mechanisms underlying the evolution of work practice.
- *Software processes are human-centered work practices:* Human actors that manage and coordinate software processes are adaptive and goal-directed agents. What people actually do, how they communicate and collaborate, how they solve problems, resolve conflicts, and learn behavior matters in the outcome of a project. Therefore, simulating human activities requires modeling communication, collaboration, team work, conflict resolution, and tool and technology usage. Decision making strategies and mental models of humans, as well as various forms of team archetypes (Yilmaz and Phillips, 2006), influence the performance and effectiveness of software processes.

Given the above observations, the position advocated in this article is that there is a need for a software process simulation framework that represents not only technical activities, policies, and procedures, but also the resources, preferences, and cognition of staff members, together with functional and social organization and strategic management, all in unified and coherent terms. A cross-disciplinary framework should support coordinating findings and models from several fields. Human organizations are the subject of study of organization theory, and the presented framework in this article is firmly grounded in the science of organizations (Galbraith, 1977), in particular, the contingency theory (Scott, 1992). The organization-theoretic perspective to simulation modeling of software processes entails characterizing the components of organizational design, as well as the types of team cooperation mechanisms that are associated with selected team archetypes. In this study, we use an agent-oriented model called Team-RUP to examine the following:

- which team archetypes and cooperation mechanisms are effective with incremental and iterative software development strategies such as the RUP and
- the extent of the impact of turbulence (i.e., requirements change and employee turnover) on the effectiveness of software development under various team archetypes in small organizations.

Various researchers have developed alternative work system modeling and design approaches. Business or enterprise process modeling is an active area of research that uses formal specifications of business process to facilitate business process reengineering. There exist cognitive modeling frameworks that are based on unified theories of cognition to explore mental processes (Newell, 1990). The field of distributed artificial intelligence has provided contributions in modeling collaboration of teams of people in complex uncertain environments. Advances in computational organization theory (Agha and Hewitt, 1999) enable modeling organizational structure and dynamics in terms of intelligent agent organizations. Such models enable exploring the impact of human and social dynamics on the effectiveness and efficiency of organizations. Human-centered work practice modeling is also advocated to improve fidelity of simulations using activity theory. This paper builds on the observations that depict software development as an adaptive human-centered work system to develop a framework that integrates operational (human, social, and organizational dynamics) and strategic levels.

15.2 Related Work

Software production methods are enacted via interactions of software teams that cooperate to build software (Agha and Hewitt, 2004); therefore, organizational structure and dynamics can have significant effects on project coordination. Hence, it is critical to better understand complex interrelations between technical activities and organizational factors that affect the performance of software processes (Little, 2005).

15.2.1 Organization-Theoretic Perspective for Simulation-Based Analysis of Software Processes

Acuna and Juristo (2005) argue that the lack of conceptualization and inclusion of organizational dynamics in software process models is a critical obstacle in exploring the sociotechnical aspects of software processes. Hofstede (1998) and others (Hayashi, 1988; Cabrera et al., 2001, 2002; Chevrier, 2003) point out that organizational culture can have significant effects on project coordination, and yet, this is not reflected in current project management paradigms applicable to software development (Cusumano and Selby, 1997). This article presents a strategy

for developing software process simulation models from an organization-theoretic perspective (Herbert, 1976; Ilgen and Hulin, 2000), and it illustrates the utility of this perspective using an agent-based simulation framework, which is called Team-Rational Unified Process (Team-RUP) (Yilmaz and Phillips, 2006). Team-RUP is a framework for agent-based simulation modeling of cooperative team behavior in software development organizations that use the RUP. The premise of the presented strategy is based on the following observations: Human organizations, including software development organizations, (a) continually acquire, manipulate, and produce artifacts through joint cooperative activities and (b) are comprised of multiple distributed agents (i. e., software engineers) that exhibit collective properties via communication, collaboration, and coordination.

15.2.2

Simulation Methods for Software Process Performance Analysis

Simulation is an indispensable tool to explore the dynamics of software processes. Various methods have been used for software process modeling.

- *Analytic Models:* The idea of creating abstract models to better understand software processes is by no means new. Analytic models (Albrecht, 1979; Boehm, 1981) that find representation to this day trace their histories back to the late 1970s and early 1980s. These models consist of mathematical equations that represent relationships between various process entities. Although useful in certain contexts, analytic models divorce process attributes from their effects and fail to consider the dynamic interaction of process factors (Martin and Raffo, 2000; Donzelli and Iazeolla, 2001).
- *System Dynamics:* System dynamics arose from Jay Forrester's research (Forrester, 1961) concerning the dynamics of business and social systems. It involves the modeling of systems in terms of feedback loops and ordinary differential equations. In the field of software processes, a seminal work aimed at understanding human resource management, software production, controlling, and planning was developed by Abdel-Hamid and Madnick (1991).
- *Discrete-Event Models:* Discrete models have contributed greatly to the understanding of software processes. Discrete-event models, which utilize event scheduling, process interaction, or three-phase simulation techniques, allow queues to be easily represented and can postpone processing until the required resources become available (Martin and Raffo, 2000). For example, Padberg (2002) has used discrete-event modeling to study software project scheduling policies. Another benefit of this technique is that model entities can be described in detail via assigned attributes. State-based modeling is another discrete approach that has proven useful. Concrete examples of state-based models include Petri-net and cellular automata models.
- *Agent-oriented Models:* Agent simulation presents a wealth of possibilities to the software process modeling community. Unlike other simulation techniques, agent-based simulation allows societies to emerge within the technical process structure. According to Wickenberg and Davidsson (2003),

agent-based simulation of processes provides “a natural way to describe both communication between individuals, individual characteristics, and the discrete decisions made during negotiations.” Strangely enough, research in this field is limited and not new. For example, Mi and Scacchi (1990) used agents to simulate processes more than a decade ago in their work on developing the Articulator environment.

15.3

Team-RUP: A Framework for Agent Simulation of Software Development Organizations

Team-RUP enables representing a specific form of structure, task model, reward (e.g., hiring/firing processes), and cooperation models relevant to RUP.

15.3.1

Organization Structure

As is most common among businesses, the Team-RUP organization is structured as a hierarchy of agents. It consists of a project manager, a design manager, and teams of engineers. The project manager performs high-level coordination tasks, whereas the design manager provides oversight to the design process and decomposes tasks of large granularity. The teams of engineers create the actual artifacts of the software development process. The remainder of the software development organization, including an independent testing group, is implemented using standard object orientation. The structure of an organization modeled in the Team-RUP framework is shown in Figure 15.1.

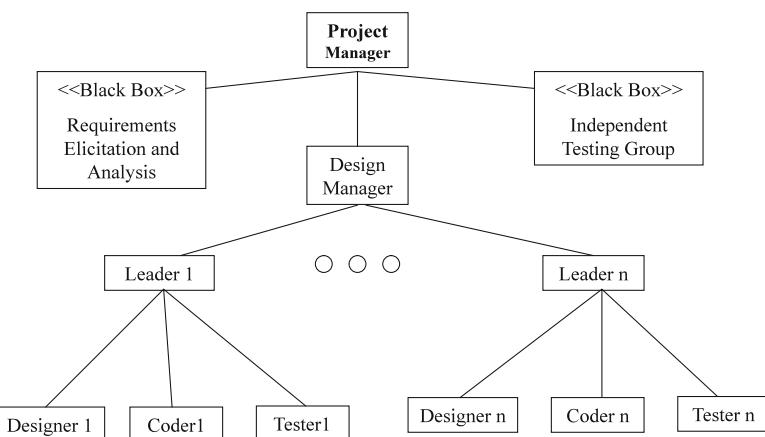


Fig. 15.1 Team-RUP organization structure.

A team is a distinguishable set of two or more individuals who interact interdependently and adaptively to achieve specified, shared, and valued objectives (Salas and Fiore, 2005). To represent cooperation at the team level, our model considers four group archetypes based on characteristics resulting from collaboration and coordination techniques. Ferber (1999) defines a collaboration technique as a task allocation mechanism that enables agents to distribute tasks, information, and resources in the advancement of a common labor. In Team-RUP, one can conceive different teams at different levels of resolution. The leader, designer, coder, and tester may constitute a single design team coordinated by the team leader. Yet, at a lower level of resolution, the group of team leaders constitutes a team that is coordinated by a design manager. Our simulation study focuses on the latter type of team formed under the governance of the design manager.

15.3.2

Team-RUP Task Model

In accordance with RUP, the development of software in Team-RUP is viewed as a multistage transformation. Representatives of the development organization elicit requirements from the customers and users. These requirements are validated, refined, and translated into an analysis model. Team-RUP models perform these tasks procedurally. The agent organization becomes involved when software construction takes place. That is, currently, Team-RUP models and simulates the design and development phase of the process. Construction begins with the analysis model being passed to the project manager, who forwards it to the design manager. During a series of time-boxed iterations, the design manager and subordinates map the analysis model into a design model and implementation. For simplification purposes, the latter two artifacts are not syntactically distinct entities in the model. As will be evident later, both are sublists of the same (semi-) sorted list. To reflect RUP's incremental nature, testing occurs during each iteration. We recall that, in the RUP, software projects are completed through a sequence of time-boxed iterations, each of which may comprise design, implementation, and/or testing. Development organizations using RUP should complete some subsets of the final system at the end of each iteration. Because of its iterative nature, however, RUP allows an organization to cope with the requirements changes. RUP also stipulates that those parts of the project involving greater risks should be addressed in the early iterations so that the overall risk to the project can be mitigated. To model this incremental and iterative process, the organization sorts integer arrays using the variation of a well-known iterative sorting algorithm, namely, Shell sort. In the Team-RUP framework, an array of integers represents a project configuration, and problem facets are modeled as ordered pairs of elements. A problem facet (x, y) becomes a task that needs to be performed if $x > y$. The reversal of such inversions constitute the tasks performed by agents in the model. The set of all possible facets pertaining to a configuration C (and hence a project) is as follows:

$$(x_i, y_i) \in C \times C , \quad i < j . \quad (15.1)$$

Suppose (x_i, y_j) is a problem facet corresponding to a configuration C , and let z_k be an element of C . If $k \geq j$, comparing x_i and z_k translates into performing a task associated with (x_i, y_j) . Similarly, if $k \leq i$, comparing z_k and y_j is also analogous to performing a task associated with (x_i, y_j) . Clearly, many tasks can be accomplished via a single comparison. As in the real world, not all facets of a problem need to be addressed to accomplish a given project; certain tasks can remain undone. Team-RUP classifies tasks according to two categories. A supporting task does not reverse the order of a problem facet pair. Principal tasks are the second form of task, and they address a special form of problem facet. Team-RUP represents the requirements in terms of inversions, that is, pairs of array elements that are out of order. A single inversion is interpreted as a principal, atomic task yet to be performed. Thus, a set of inversions is an incomplete requirement fulfillment (i.e., principal task), and removing a set of inversions corresponds to fulfilling a requirement. Sets of inversions can be decomposed into subsets, just as a task can be decomposed into subtasks. With this mapping of tasks to inversions, Shell sort is a metaphor for RUP for several reasons. The algorithmic approach of removing inversions in phases coincides with the RUP iterations. Of greater significance is the fact that a Shell sort phase does not undo the work performed in earlier phases: 2-sorting a 5-sorted list generates a 5-sorted (as well as a 2-sorted) list. This reflects the fact that each RUP iteration produces a part of the final system rather than throwaway work products of draft quality. Recall that Shell sort initially swaps unordered elements that are far apart and decreases, with each phase, the distance between the elements it compares. Because it eventually sorts the set of adjacent elements, the algorithm is guaranteed to sort the entire list. Since inversions inserted by an outside entity during the execution of Shell sort will ultimately be removed by later phases, the algorithm reflects RUP's ability to cope with changing customer requirements.

15.3.3

Team-RUP Team Archetypes and Cooperation Mechanisms

In our research, the technical aspects of the team operation (i.e., task work) in conjunction with all interactive behavior among the members are regarded as teamwork. Cooperation is the general form of interaction that is common in software development organizations. The problem of cooperation reduces to who does what, by what means, in what way, when, and with whom. Therefore, task allocation, coordination of actions, and resolution of conflicts constitute the fundamental components of cooperation. Hence, we classify teams in terms of the degree of autonomy afforded by cooperation strategies. In particular, team collaboration strategies are classified as top-down or bottom-up. As the former entails stepwise refinement, a large degree of oversight is required, which diminishes autonomy. The latter, however, provides more flexibility since the structure of the final integrated product is not entirely preconceived. These categories are further subdivided in terms of coordination. According to Ferber (1999), coordination of actions means “the articulation of the individual actions accomplished by each of the agents in such

Degree of autonomy in collaboration	Low (Top-down task allocation & problem solving)	High (Bottom-up task allocation & problem solving)
Degree of concurrency in coordination		
Low (Linear)	Synchronized	Agile
High (Concurrent)	Asynchronous	Autonomous

Fig. 15.2 Team-RUP team archetypes.

a way that the whole ends up being a coherent and high-performance operation.” Figure 15.2 presents the types of archetypes explored in Team-RUP.

15.3.4

Reward Mechanism in Team-RUP

Within a company, internal turbulence results from employee turnover. Employees can be fired or they can quit their jobs. Internal turbulence significantly affects both the efficiency and effectiveness of an organization. Therefore, these two characteristics are of principal interest in applications of the framework; Team-RUP explicitly addresses internal turbulence. Three variables, each of which takes on a value between 0 and 1, influence the level of internal turbulence.

- The *base rate* parameter B represents the degree to which a company is downsizing. Its value has the greatest impact among these variables in determining the security and desirability of any given employee’s job. With a high base rate, an employee is more likely to be fired or quit due to pressure or an ample severance package. Notice that Team-RUP treats firing and quitting as dependent events. While an individual’s decision to quit his job may not be related to a projected payroll size, the rate at which such decisions are made for the company as a whole is closely linked in the general case.
- The *sensitivity to performance* parameter S determines the degree to which an employee’s performance affects his chance of separating from the company. This number, of course, has little meaning without the performance variable P . Performance is measured at the team level. Initially, the performance for each team is 0.5. It varies with respect to the team’s current and overall historical productivity. Suppose the requirements array is to be n -sorted by k teams during iteration i , and m inversions should be removed by this operation. The expected number β_{ti} of removed inversions for team t is defined as follows:

$$\beta_{ti} = \frac{m}{k} . \quad (15.2)$$

Assuming team t actually removed ati inversions during iteration i , the milestone performance δ_{ti} of t during i is defined by

$$\delta_{ti} = \frac{ati}{\beta_{ti}} . \quad (15.3)$$

The separation of an employee from the company in the Team-RUP framework is a probabilistic event. The probability F_{ti} that some member of team t will separate from the company at the end of iteration i is given by the following equation:

$$F_{ti} = B + S(1 - \delta_{ti}) . \quad (15.4)$$

The internal turbulence can be influenced by the model user by adjusting the *base rate* and *sensitivity to performance* parameters. A strict interpretation requires these two parameters to sum to 1 in order to satisfy the definition of a probability. Internal turbulence T can then be quantified as any linear combination of the base rate and sensitivity to performance:

$$T = \alpha B + \beta S . \quad (15.5)$$

In the current implementation, α and β are set to 1 and $B = S$.

15.4

Design and Implementation of Team-RUP

A software project is not the simple execution of a predefined sequence of atomic steps. Rather, it is a progression through a series of interdependent states governed by a complex system of time-dependent conditions. Figure 15.3 reveals the workflow of autonomous teams during a single iteration. Although it focuses on a single team type, it illustrates high-level coordination activities common to all behaviors. The project manager is responsible for starting and stopping each iteration.

At the start of an iteration, the project manager tells the design manager to implement a set of components. The design manager chooses one of the components and creates a work breakdown structure. Each leaf of this tree represents a subcomponent to be implemented by a single team. These subcomponents are assigned to teams until either the leaves or the available teams are exhausted. As teams finish the subcomponents, the assignment process continues. Once some of the leaves have been addressed, the design manager may assign interior nodes. Interior nodes can be assigned to teams once all their descendants have been completed. Interior nodes represent integration tasks and are assigned to two teams. Each team should have previously been assigned as a child of this node. They collaborate to integrate the subcomponents. Once all the nodes of a work breakdown structure are addressed, the completed component is sent to the testing division. If the testing division does not find it acceptable, the component is returned to the design managers queue of components to be implemented during the current iteration. The

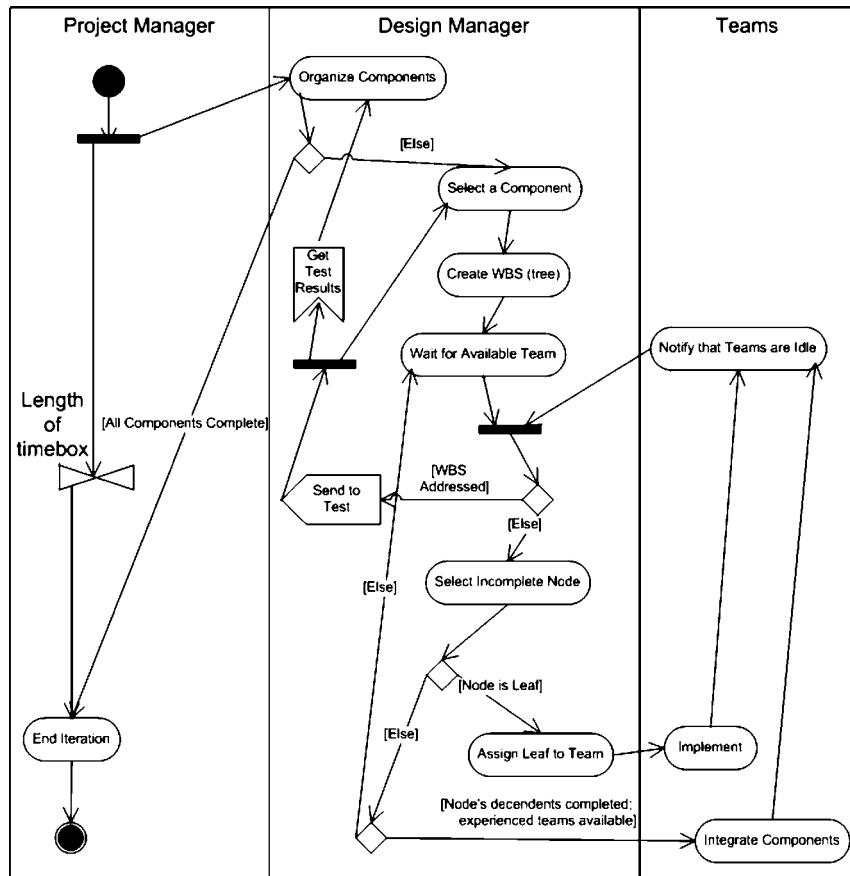


Fig. 15.3 Workflow for autonomous teams.

construction division continues the process of designing and implementing components until the component set is exhausted or the time-box expires. In the Team-RUP model, the nature of collaboration varies with the selection of team behavior. While it is always centralized due to organizational constraints, task allocation possesses characteristics from both the imposed and brokered paradigms (Ferber, 1999). Task assignments involving the project manager are imposed. In contrast, the design manager can exhibit trader qualities. For example, synchronized teams query the design manager to ascertain the team whose skill set matches a service needed by the inquiring team. Likewise, agile teams consult the design manager to find out which team manages a particular requirement. In contrast, imposed allocation characterizes the autonomous team behaviors.

15.4.1

Performance Metrics

To maximize the applicability of the results across the vast diversity of software development objectives and constraints, efficiency and effectiveness have been chosen as the principal indicators of utility in this study. We measure efficiency in terms of productivity and staff utilization. Effectiveness is viewed as a combination of timeliness and software quality. In accordance with Practical Software Measurement (Office of the Under Secretary of Defense for Acquisition and Technology) (OUSDAT, 1998), productivity P is defined as the ratio of the product size to the effort, where effort is typically quantified in terms of labor hours. Taking a functional view of product size, we define this quantity as the total number I of inversions in the original requirements list. Effort E is the product of the number of agents A and the number of time ticks T in the simulation. That is,

$$P = \frac{I}{TA} . \quad (15.6)$$

The formula for staff utilization Su also contains effort in its denominator. The numerator, however, equals the total number Pt of “productive ticks” taken across all agents. A time tick is considered productive for a particular agent provided that agent performs some work-related activity during that time tick. For example, consider a three-tick experiment with two agents. If one agent works during every tick and the other agent only works during the second tick, we have $Pt = 4$. In summary,

$$P = \frac{Pt}{TA} . \quad (15.7)$$

Timeliness is measured as a function of build content. The build content measure is the number of components Cc completed during an iteration divided by the number of components Ca allotted to the iteration at its start (OUSDAT, 1998). In this study, the granularity of a measurable component is the same as that of a testable component: a single cell of an n -cell partition of indices created during an attempt to n -sort the values corresponding to those indices. Timeliness (Tm) is the mean value of the build content measures taken over all iterations. In an experiment with n iterations, we have

$$P = \frac{1}{n} \sum_{i=1}^n \frac{Cc_i}{Ca_i} . \quad (15.8)$$

Quality Q is measured in terms of defect density. Practical Software and System Measurement (PSSM, 2006) defines defect density as the number of defects in a component divided by the size of that component. Translating this definition into the language of the Team-RUP model, quality equals the number of remaining inversions, R_i , divided by the total number of inversions (I) in the original requirements list:

$$Q = \frac{R_i}{I} . \quad (15.9)$$

15.4.2

Validation of the Model

To validate the model, we compared its behavior to the established facts observed in the empirical studies. The first test focuses on the degree of concurrency in coordination. Whenever multiple tasks and multiple labor sources coexist, the issue of synchrony arises. Typically, asynchronous systems are implemented with the hope of improving performance. Depending on the task at hand, however, this strategy may or may not be effective. Performance, therefore, cannot be used to distinguish between linear and concurrent systems. Stability, however, can be used as a delineating factor.

Proposition 1: Asynchronous systems are less predictable than synchronous systems

(Shamsi et al., 2005): To validate the model in terms of coordination, we designed a test quantifying predictability in terms of standard deviation. With each behavior type, 50 replications were run for each factor combination (workload and number of teams). Productivity, staff utilization, timeliness, and quality measures were collected. For each of these metrics, we calculated the standard deviations across each set of replications. This calculation resulted in 25 data points per metric per team behavior type. The average of each set of 25 data points was calculated and histograms were generated. In each of the graphs in Figure 15.4, autonomous and concurrent behaviors exhibit greater variability than agile and synchronous teams. As expected, teams that coordinate their work efforts concurrently are less predictable than those that synchronize their activities.

Proposition 2: A bottom-up strategy is more adept at responding to change : Our second test focuses on the degree of autonomy in collaboration. That is, it focuses on the delineating characteristics of teams that use top-down versus bottom-up collaboration strategies. Top-down strategies offer a tighter fit between the set of requirements and the implemented product. They are more intuitive and match well with strategies employed by other engineering disciplines. Unfortunately, they respond poorly to requirements changes. A new development plan must be created or the original plan must be adapted. Because implemented components are delayed until the end of the project, strict deadlines may result in unfulfilled requirements. Bottom-up strategies reverse this trend. This is shown in Figure 15.5.

Hypothesis 3: Parent organizations absorb the inefficiencies of random (agile) teams and shield them from fluctuating market trends; therefore, efficiency is not an inherent trait of random (agile) teams (Constantine, 1993): An efficient team accomplishes more work in less time with fewer people. Therefore, efficiency can be quantified as the ratio of productivity to staff utilization. Experiments with Team-RUP reveal that agile teams are particularly efficient in comparison to other team behaviors as the size of the organization and workload of individual teams increase (Yilmaz and Phillips, 2006). However, agile teams are not inherently efficient with respect to synchronized teams, on average.

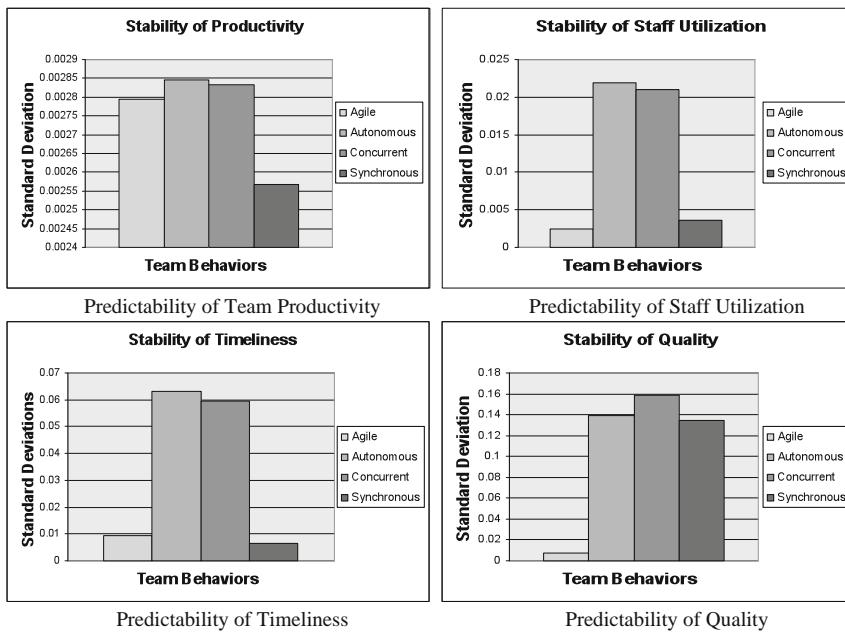


Fig. 15.4 Predictability across team archetypes.

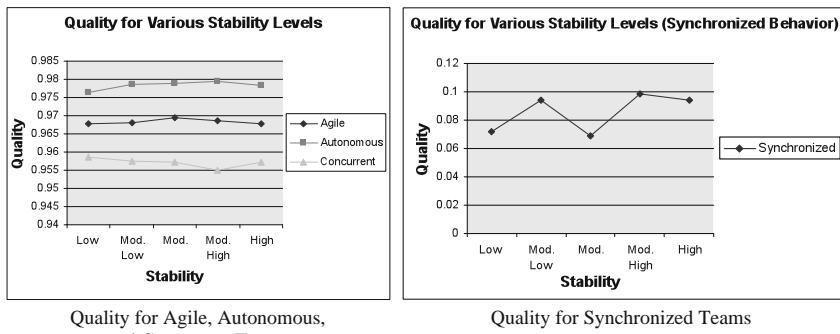


Fig. 15.5 Adeptness to change.

15.5

Results and Discussion

The results of this experiment reveal several interesting trends. In terms of productivity (Figure 15.6), all teams perform similarly. As would be expected, increasing the number of teams leads to a decrease in productivity. More resources (teams) are expended on the same number of requirements. It is interesting to note that a point of inflection exists for each team and each turbulence level at some point between three and five teams.

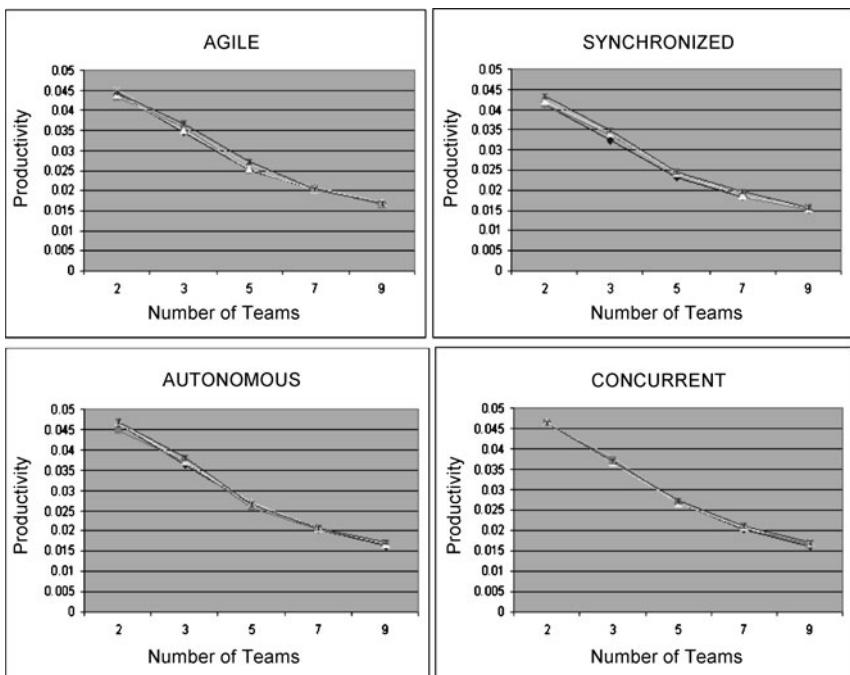


Fig. 15.6 Productivity of teams under turbulence.

As would be expected, we found out that increasing the number of teams leads to a decrease in productivity. More resources (teams) are expended on the same number of requirements. It is interesting to note that a point of inflection exists for each team and each turbulence level at some point between three and five teams. A similar trend exists for the staff utilization graph. Except for an initial increase for synchronized teams, staff utilization also decreases as the number of teams increases. The initial increase in utilization for synchronized teams can be explained in terms of specialization. As the number of teams increases, the more specialized the teams become. This increases the likelihood that a team will encounter a task for which it requires another teams specialized skills. Since more teams must be involved, staff utilization increases.

Of greater interest to this study, however, is the fact that agile and synchronized teams are timelier when fewer teams are present. The synchronization tasks are less costly when fewer people are involved. In contrast, autonomous and concurrent teams are timelier when more teams are present. Because of the greater number of teams, more components can be distributed at any given time for concurrent implementation. The payoff associated with this staff increase is significantly reduced as turbulence increases. A final point of interest centers on the agile team behavior. The agile team behavior surpasses all other behaviors for every factor combination. At its worst setting of 12 teams, the agile team is timelier than autonomous and concurrent teams. Agile teams face a tradeoff between timeliness

and quality. Smaller organizations are more deeply affected by change regardless of their process strategy. On the other hand, the difference between the best and worst quality resulting from agile teams is less than 2 percentage points. Furthermore, this worst quality rating surpasses the best generated by any other team. This fact reveals the superiority of the agile behavior with small organizations. Another interesting tradeoff is found with synchronized teams. Experiments with Team-RUP indicate that the synchronized strategy, which is the most rigid strategy, produces low quality software for each setting. It performs its worst for teams of size 3 and 5. Specialization also explains the low-quality responses at these levels. When many specialists are involved on a project, quality increases because worker skill sets are more finely honed. This observation is a simple restatement of Adam Smith's division-of-labor principle. For a small number of teams, however, the specialization areas are too large for skill improvement to lead to a performance increase. However, the labor divisions prevent teams from aiding one another, which creates blockages in the workflow.

15.6 Conclusions

Team-RUP is an agent-based simulation framework that supports the mutual study of software development, team dynamics, processes, organizational paradigms, and cultures. Used to create exploratory models, the framework is designed to discover general results applicable on an industrywide basis. In order for these results to be useful in a real-world context, however, Team-RUP is firmly grounded in one of the most widely used process frameworks: the RUP. Team-RUP uses a distinctive layered approach allowing features to be modeled at the individual, team, and enterprise levels. Crosscutting features like internal turbulence can also be captured. Also, it uses a unique modeling approach based on sorting. Finally, interagent communication serves as a central modeling component. In the current Team-RUP implementation, agency is represented on the team level. That is, the simulated development of software takes place at the level of interacting teams. Extension of the work to collaboration and cooperation among individual software agents is under investigation. A central feature of the Team-RUP framework is the classification of team behaviors according to the degree of autonomy in collaboration and the degree of concurrency in coordination. This classification yields four archetypical team behaviors: agile, autonomous, concurrent, and synchronous. In the current implementation, the efficiency and effectiveness of teams of these types under turbulent conditions were of primary interest.

With the current implementation of Team-RUP, we have focused on two experiments. The first experiment centered on how teams with various team behaviors responded to different levels of internal and external turbulence. The second considered teams of various sizes in order to study the impact of turbulence on small development organizations. Each provided interesting insights. First, we observed that the RUP provides a layer of insulation against the negative effects of employee

turnover and requirements changes regardless of the team behavior type. However, concurrent teams are most affected by turbulence. As turbulence increases, timeliness and quality decrease, despite an increase in productivity. Although turbulence has the greatest impact on concurrent teams, this behavior does not yield the lowest performance among the team types.

With its rigid structure, the synchronized behavior is the least suited for adaptation to changing requirements and immovable deadlines. Division of labor among specialized teams can be either a help or a hindrance to synchronized teams. When a large number of teams are present, the skill improvement due to specialization leads to increases in timeliness and quality. For a small number of teams, specialization can lead to blockages in the workflow, which ultimately reduces timeliness and quality. In direct contrast to the synchronized behavior is the agile behavior.

This study provides quantitative evidence that agility is a valid and useful counterbalance to the inevitable changes involved in most real-world software projects. If quality and timeliness are the primary objectives of a development effort, small organizations should strongly consider adopting an agile process to promote agile team behavior. If, however, an autonomous or a concurrent approach is desirable for some project-specific purpose, it will be most applicable in larger organizations.

References

- Abdel-Hamid, T. and Madnick, S. (1991) *Software Project Dynamics: An Integrated Approach*, Prentice Hall, Upper Saddle River, NJ.
- Acunam, T.S. and Juristo, N. (2005) *Software Process Modeling*, International Series in Software Engineering, vol. 10, Springer.
- Agha, G. and Hewitt, C. (1999) Computational organization theory. *Multi-Agent Systems: A Modern Approach to Distributed Artificial Intelligence*, pp. 19–41.
- Agha, G. and Hewitt, C. (2004) Software development teams. *Communications of the ACM*, **47** (2), 95–99.
- Albrecht, A. (1979) Measuring application development productivity. *Proceedings of IBM Application Development Joint SHARE/GUIDE Symposium*, pp. 33–39.
- Boehm, B. (1981) *Software Engineering Economics*, Prentice-Hall, Englewood Cliffs, NJ.
- Cabrera, A., Cabrera, F.E. and Barajas, S. (2001) The key role of organizational culture in a multi-system view of technology-driven change. *International Journal of Information Management*, **21**, 245–261.
- Cabrera, A., Cabrera, F.E. and Barajas, S. (2002) The maturation of offshore sourcing of information technology work. *MIS Quarterly Executive*, **1**, 65–77.
- Chevrier, S. (2003) Cross-cultural management in multinational project groups. *Journal of World Business*, **38**, 141–149.
- Constantine, L. (1993) Work organization: Paradigms for project management and organization. *Communications of the ACM*, **36** (10), 35–43.
- Cusumano, A.M. and Selby, W.R. (1997) How microsoft builds software. *Communications of the ACM*, **40**, 53–61.
- Donzelli, P. and Iazeolla, G. (2001) A hybrid software process simulation model. *Software Process Improvement and Practice*, **6**, 97–109.
- Ferber, J. (1999) *Multi-agent Systems: An Introduction to Distributed Artificial Intelligence*, Addison Wesley Longman Inc, New York, NY.
- Forrester, J. (1961) *Industrial Dynamics*, Pegasus Communications, Waltham, MA.
- Galbraith, J.R. (1977) *Organization Design*, Addison-Wesley, Reading, MA.

- Hayashi, S. (1988) *Culture and Management in Japan*, University of Tokyo Press, Tokyo.
- Herbert, T.T. (1976) *Dimensions of Organizational Behavior*, Macmillan Publishing Company, New York, NY.
- Hofstede, G. (1998) Identifying organizational subcultures: an empirical approach. *Journal of Management Studies*, 35, 32–49.
- Ilgen, D.R. and Hulin, C.L. (2000) *Computational Modeling of Behavior in Organizations*, American Psychology Association, Washington, DC.
- Little, R.G. (2005) Organizational culture and the performance of the critical infrastructure: modeling and simulation in socio-technological systems. *Proceedings of the 38th Hawaii International Conference on System Sciences*, pp. 92–101.
- Martin, R. and Raffo, D. (2000) A model of the software development process using both continuous and discrete models. *Software Process Improvement and Practice*, 5, 147–157.
- Mi, P. and Scacchi, W. (1990) A knowledge-based environment for modeling and simulating software engineering processes. *IEEE Transactions on Knowledge and Data Engineering*, 2 (3), 283–294.
- Newell, A. (1990) *Unified Theories of Cognition*, Harvard University Press, Cambridge, MA.
- OUSDAT (1998) *Practical Software Measurement: A Foundation for Objective Software Measurement, Version 3.1a*. Office of the Under Secretary of Defense for Acquisition and Technology, Joint Logistics Commanders, Joint Group on Systems Engineering.
- PSSM (2006) *Practical Software System Measurement*, <http://www.psmsc.com/>. Last accessed November 2006.
- Padberg, F. (2002) A discrete simulation model for assessing software project scheduling policies. *Software Process Improvement and Practice*, 9, 127–139.
- Salas, E. and Fiore, S.M. (2005) *Team Cognition: Understanding the Factors that Drive Process and Performance*, American Psychology Association, Washington, DC.
- Scott, R.W. (1992) *Organizations: Rational, Natural, and Open Systems*, Prentice-Hall: Englewood Cliffs, NJ.
- Shamsi, J., Chu, C. and Brockmeyer, M. (2005) Towards partially synchronous overlays: issues and challenges. *International Workshop on Advanced Architectures and Algorithms for Internet Delivery and Applications*, pp. 19–25.
- Wickenberg, T. and Davidsson, P. (2003) *On multi-agent based simulation of software development processes*, <http://www.ide.bth.se/ipdv/Papers/MABS2002.pdf>. (accessed 1 May 2008).
- Yilmaz, L. and Phillips, J. (2006) Organization-theoretic perspective for simulation modeling of agile software processes. *Software Process Change – LNCS*, 3966, 234–241.

16

Agent-Directed Simulation for Manufacturing System Engineering

Jeffrey S. Smith, Erdal Sahin, and Levent Yilmaz

16.1

Introduction

Manufacturing accounts for a significant portion of the world's economic output. For instance, according to the Bureau of Economic Analysis (2008), 11.7% of gross domestic product in the United States falls into the manufacturing sector. Recently, globalization has significantly affected the manufacturing sectors of many of the world's economies. Recent advances in transportation and communication technologies have significantly reduced the competitive advantage of manufacturing close to the point of use and products can be manufactured virtually anywhere and transported to the point of use. As a result, companies are being forced to optimize all aspects of their manufacturing operations in order to remain competitive in this global environment. These issues are particularly important for small to mid-size manufacturing companies.

One of the main reasons for this is related to the demand side. The ability of competitors to produce the same types of products is much higher now and customers are more willing to switch between producers based on cost and delivery options offered by different manufacturers (Buckley, 2008). As a result, companies face increased demand volatility, requiring more flexibility in terms of product output. Another important effect of globalization of manufacturing derives from the fact that globalization has created the need for more innovative products with shorter product life cycles. This new idea requires businesses to have flexible manufacturing systems (FMSs) where they can switch producing new products depending on the market to stay competitive.

With increased competition, manufacturing systems analysis becomes critical in order for companies to survive. Manufacturing systems analysis provides the foundation for system design and system operation. Effective analysis allows the decision makers to make effective strategic, tactical, and operational-level decisions with regard to issues like site location, system configuration, staffing, production planning and scheduling, and shop floor control. Due to the complexity of the sys-

tems being studied, simulation has long been a widely used analysis tool in this domain.

This chapter describes the application of simulation and agent-based modeling in the manufacturing system domain. This chapter is organized as follows. Section 16.1.1 describes what is meant by manufacturing system for this chapter. Section 16.3 describes the general concepts of agent-based modeling. Section 16.2 describes the application of simulation to manufacturing systems design (Section 16.2.1) problems and operation (Section 16.2.2) problems. Section 16.3 describes agent-directed simulation and some applications of this technology to manufacturing system control and agent-based manufacturing. The holonic approach for addressing the limitations of centralized control is also discussed in this section. Finally, Section 16.4 presents the summary and conclusions for the chapter.

16.1.1

Manufacturing Systems

Before discussing how simulation can be and historically has been applied to manufacturing systems engineering, we first define what we mean by a *manufacturing system* and introduce the concepts of agent-based modeling. For our purposes, a manufacturing system is a set of interacting entities and resources that create physical *things*. We specify *physical* things to distinguish manufacturing systems from systems that generate knowledge, services, or other nonphysical things. Some examples of manufacturing systems include common systems such as automobile assembly lines, semiconductor fabrication facilities, soda bottling facilities, machine shops, textile plants, and so on. In addition to these OEM-type facilities, there are many more component manufacturers and suppliers that supply raw materials, components, and subassemblies to OEM manufacturers. The combination of the OEM and the individual suppliers is commonly referred to as the *manufacturing supply chain*.

Within the manufacturing system, *entities* are things that move around within the system and either are “processed” or contribute to the processing of other entities. In the manufacturing system domain, entities can include things such as parts, components, finished products, people, raw material, waste/scrap, and so on. *Resources* are things that have limited capacity for which entities compete. Similarly, resources can include things such as machines, physical space, people, transporters, and so on. Allocation of limited resource capacity to process entities and delivery of these entities to market in a cost-effective manner is the crux of the manufacturing operations problem.

As an example, consider a small assembly and packing system that takes 10 components, assembles them into a finished product, and packages the product for distribution. In this system, the 10 components are the entities. If the components are assembled into subassemblies prior to the final assembly, these subassemblies are also entities. The packaging raw materials are also entities. The operators that perform the assembly operations along with any assembly and/or packaging machines are the system resources. The raw material, finished goods, and assembly

line space is also a resource. Analysis is required to support the system design (partitioning the available space into storage and assembly space and configuring the storage space and the assembly line, for example), system planning (determining the overall staffing levels and long-term system throughput, for example), and system operation (assembly line balancing and staff/task allocation, specifying required daily throughput, and production scheduling, for example).

16.1.2

Agent-Based Modeling

Agent-based modeling (ABM) is a very successful conceptualization paradigm that easily allows the modeling of highly complex physical systems composed of autonomous and interacting entities (Gianni, 2008). However, a very important question arises here: Should every company adopt agent-based simulation for every application? The answer to this question is not necessarily. If the application on hand consists of a complex structure and changes in the future, the agents will be the right tool to use (Odell, 2007). To address the same question above, Lockemann and Nimis (2004) also state that agent technology is superior than other approaches where the problems are nondeterministic.

Although different sources have different definitions for describing an agent, there are some key characteristics mentioned commonly by different sources that an agent should have. First, an agent is an autonomous, social, and adaptive entity. An agent is autonomous because it can decide on its own action since it has some level of control over its own actions (Odell, 2007). In other words, there is no need to tell an agent what to do such as with a human or any other external object.

Second, an agent is social or interactive, which is why an agent is able to communicate with the environment it belongs to, such as through other agents and humans (Bellifemine et al., 2007). Communication here means that agents asynchronously send and receive messages from other agents. Finally, agents are adaptive. Being adaptive means that an agent should be able to respond to the other agents and changes in the environment in a “correct” way (Odell, 2007). This also means that agents should improve their level of knowledge over time while making decisions in the future.

ABM includes the identification of the objects in a system. After object identification, the agent modeler has to decide which objects are going to be assigned as agents, in other words, which objects will be able have agent characteristics as explained above. According to Cantamessa (1997), finding the right balance in deciding what objects, or entities, should be modeled as agents is very important in order to have a robust model.

From the agent application point of view, different authors have used different ways of deciding whether an object should be modeled as an agent or not. However, Cantamessa (1997) states that intelligence and autonomy may be assigned to all objects in a system and that this will increase the flexibility of the system. On the other hand, assigning all the objects as agents in the system will cause very expensive implementations because of hardware and software allocation for each

agent. Moreover, this may also make the understanding of the system behavior very difficult.

ABM not only deals with agent assignments but also includes the general modeling of a system. ABM decides on the system architecture, agents, agent types and roles, interactions between agents, and how those interactions will be controlled.

16.2

Simulation Modeling and Analysis for Manufacturing Systems

Simulation is a widely used tool for analyzing manufacturing systems. Discrete-event simulation (DES) in particular is ideally suited for modeling manufacturing systems. While continuous simulation is a popular methodology for modeling and analysis of continuous manufacturing systems (e.g., petroleum refineries, chemical processing, food and beverage processing, etc.) and some large-scale manufacturing models, this chapter focuses exclusively on discrete manufacturing systems.

ABM is another simulation paradigm that has recently seen significant advances and increase in use. This modeling approach is newer than the previously mentioned ones and gives better results and more detailed information about a system than the previously mentioned traditional methods under some circumstances. Dubiel and Tsimhoni (2005) state the path of entity movements in DES has to be predefined so that the next movement of an entity will be dictated accordingly. However, humanlike travel is not very easy to model with these constraints because how people move cannot be predefined as in DES. For situations like these, ABM can provide greater flexibility to simulate the real-time interaction between entities of the system.

In addition, Dubiel and Tsimhoni (2005) state that autonomous movements of individual entities are very difficult with DES, and so modeling real-time decisions of individual agents is very difficult with DES. However, ABM uses agents and their interactions between them in order to reach a certain goal in the system. Cantamessa (1997) discusses the fact that objects in ABM in a manufacturing environment are assigned agents and so they can communicate among each other to manage the entire system. The interaction between agents are mostly done by using the negotiation and auction mechanisms. Clearly, this leads to flexibility and decentralized control in manufacturing systems, which is different from traditional centralized control.

In terms of use of ABM from the system architecture point of view, Cantamessa (1997) indicates that there are two main structures, heterarchical and hierarchical and indicates that most agent-based models developed are associated with heterarchical structures. A heterarchical, or decentralized, structure is one that does not include a predefined organizational hierarchy and master-slave relationships. As explained in agent characteristics above, each object in a system has autonomy and can directly interact with the other agents in the system. Each object has enough information to achieve its own goal. As is clear, this heterarchical structure is very well suited for ABM.

The second and traditional structure is the hierarchical structure. Hierarchical structure implies a main control unit that collects and processes all the data and manages the entities in a manufacturing system. In other words, entities of the manufacturing system receive the message coming from the centralized control unit and perform the task accordingly. Although creating this structure is easy, it has some disadvantages. First, the responsiveness of the system decreases as the system increases in size; this sets a limitation on the flexibility of such systems. More importantly, as the system gets bigger, its complexity grows and decision making at the higher levels in the hierarchy becomes quite difficult.

In a parallel view, Shen et al. (2006) state that heterarchical structure can respond to changes and disturbances faster via local decision making enabled by autonomy. This approach also provides much better fault tolerance than traditional approaches because of the autonomy of individual agents. This is because in traditional approaches, the whole success of the system greatly relies on the central control unit.

Throughout this chapter, we use the manufacturing system classification described by Smith (2003) to describe how simulation has been applied to the design and analysis of manufacturing systems. The original purpose of the classification was to organize and survey the published literature describing the use of simulation for manufacturing systems design and operation. For this chapter, we have a related, but different, focus. Our focus is on describing how simulation can be and has been used to study, design, and analyze manufacturing systems. However, the classification provides a convenient framework with which to discuss these uses. Note that we also discuss some more recent research that has appeared since the publication of the original literature review.

16.2.1

Manufacturing System Design

Design decisions are, by their nature, long-term decisions. That is, the time frame over which manufacturing system design questions are asked is long relative to the time frame for operational decisions. Given this magnitude of the decisions, the simulation analysis generally involves considering many different alternatives and is fundamentally not time-constrained (Smith, 2003).

DES first gained popularity as an analysis tool for designing manufacturing systems. Our classification includes the traditional facility layout and material handling design problems. These are well-known problems and have long histories of research and application. In addition, the classification also includes cellular manufacturing system and FMS design problems. These are relatively newer design problems that have arisen with the need for increased flexibility and reduced production volumes in the underlying manufacturing systems.

16.2.1.1 General Design and Facility Layout

The terms “plant design” and “plant layout” in fact imply different meanings. Plant layout refers to design and installation of worker systems, materials, and tools. However, plant design is broader by definition since it also includes other decisions such as location of the plant, planning of the finances, and so on (Moore, 1962). From this point of view, facility layout is regarded as a subset of facility design and basically deals exclusively with the physical requirements in a facility.

Smith (2003) surveys several papers that describe the use of simulation for general design and layout problems. In particular, Savsar (1991) describes an application of simulation that evaluates layout alternatives. Another application done by Pritsker (1982) shows how simulation is used for a pharmaceutical manufacturing design problem. Williams and Celik (1998) implement simulation for the evaluation of an automobile assembly line. The research includes the analysis of material handling and conveyors in the manufacturing process and comparison of different possible layouts. Williams and Gevaert (1997) use simulation in the analysis of an automobile supply company to assess the performances of different designs.

More recently, Noguera and Watson (2004) show how DES is used to evaluate different plant layout designs for a chemical process facility and how the outcomes are combined with statistical tools to have deeper insight by conducting sensitivity analysis. Gien and Jacqmart (2005) describe the use of DES and a fuzzy approach in designing manufacturing systems under uncertain, inaccurate design data. Greasley (2008) implements DES to design the facility layout for a textile plant under two different scenarios of the bottleneck constraint by calculating the WIP since larger WIP numbers would require more facility space. Cochran and Kaylani (2006) examine combining push and pull systems for a multiproduct serial line. Their emphasis is mostly on junction points, amount of safety stock for the push part, and number of Kanbans for the pull part to achieve the best system performance. In addition to the applications above, simulation is also being extensively used in semiconductor fabrication design. For instance, Kuroda and Tomita (2005) use simulation to analyze semiconductor wafer fabrication processes and how to build a robust production system, and Chen et al. (1997) use simulation in a three-phase method to obtain detailed design for wafer fabrication.

16.2.1.2 Material Handling System Design

Material handling is an area where simulation can be beneficial at the design stage. It is often the case that not enough attention is paid to material handling requirements at the plant design stage. As such, the material handling system design is often done after the initial facility design, leading in many cases to suboptimal designs. Plant floor material handling is a weak link of most manufacturing systems and simulation promises a fast and adjustable way of modeling material handling (Hao and Shen, 2008). In other words, the better the material handling system is, the better the system performance is.

Smith (2003) surveys many papers that show the usage of simulation in material handling. Specifically, Chang et al. (1986) explain the use of simulation for an FMS. Hutchinson (1983) describes how simulation is used for the design and analysis of the automated material handling system for a sheet metal manufacturing plant. Ozden (1988) discusses how different design factors such as queue capacities, number of automated guided vehicles (AGVs) for an AGV system will affect throughput and utilizations for an FMS. There have been studies of simulation in material handling other than those cited in Smith's survey. For instance, Hao and Shen (2008) describe the simulation of material handling in a pull production. A Kanban-based material handling system is investigated to make it in line with the pull production line and both discrete-event and agent-based technologies to model how complex material handling processes in an assembly line are implemented.

16.2.1.3 Cellular Manufacturing System Design

Cellular manufacturing is the application of group technology to manufacturing systems. Basically, this means producing groups of similar products in a cluster of machines configured to provide all of the processing requirements for those products (Angra et al., 2008). In other words, parts that need similar processing are grouped together and processed as a group in a dedicated grouping of machines called a *cell*.

Smith (2003) describes many applications of simulation in this area. For example, Taj et al. (1998) implements simulation for the verification of manufacturing cell design. Welgama and Mills (1995) use simulation for the analysis of two different cell designs for a chemical manufacturing plant based on operator and material handling utilizations. Logendran and Talkington (1997) conduct a simulation study for the comparison of performance functional and cellular layouts where there are machine breakdowns. In addition to papers cited in that literature survey, there are other examples of simulation such as Chan and Ip (1995), who describe how simulation and Analytical Hierarchy Process (AHP) can be used together to choose the best alternative manufacturing layout design by taking into account the financial and nonfinancial attributes. They use four different layout alternatives including group and FMS technology.

16.2.1.4 Flexible Manufacturing System Design

An FMS is a group of numerically controlled machines connected by an automated material transport system, all under the control of a computerized control system (Jain et al., 2008). FMSs promise a reliable, flexible, and fast manufacturing process. FMS design is also another specific area where simulation has been extensively used. Some specific papers cited in Smith (2003) will be mentioned below.

Brown and Rathmill (1983) show the general application of simulation for FMS design and operational planning. Kiran et al. (1989) use simulation in the evaluation of design alternatives for FMSs at a specific facility. Another study by Caprihan and Wadhwa (1997) evaluates the effect of routing flexibility on a FMS's cycle time.

In addition to the papers mentioned above, Kumar and Sridharan (2007) describe a simulation-based experimental study of a tool sharing problem in an FMS.

An attempt has been made to present the details of a simulation study conducted to investigate the effect of tool request selection rules in combination with part launching rules under three different scenarios of operation of a typical single-stage multimachine FMS in a tool sharing environment. Deroussi et al. (2006) describe the combination an optimization method and a DES method in order to answer to the complexity of scheduling of operations on the machines to minimize the makespan in an FMS. Um et al. (2007) describe the usage of aspect-oriented simulation to determine the optimal buffer size in FMS to improve the system performance.

16.2.2

Manufacturing Operation

Manufacturing operation decisions are generally short- to medium-term decisions as opposed to manufacturing system design applications involving long-term decision making (Smith, 2003). In other words, manufacturing operation decisions are generally related to the day-to-day or week-to-week operations of the system, whereas the design decisions generally span months to years. We divide this class of applications into operational planning and scheduling applications, real-time control applications, operating policy selection applications, and performance analysis applications. The following sections discuss these application classes.

16.2.2.1 Operational Planning and Scheduling

The operational planning and scheduling class includes capacity and production planning and production scheduling (Smith, 2003). *Planning* in this context involves developing a production plan, an inventory plan, and determining the required amounts of manufacturing resources and inputs (e.g., tooling, employees, raw materials, etc.) required to execute the plan. *Scheduling* in this context involves determining the specific timing of the manufacturing activities comprising the production plan. Manufacturing planning and scheduling have long been studied in the general operations research field and are common areas for the application of simulation as there is significant uncertainty that must be considered during analysis.

Andersson and Olsson (1998), Schriber and Stecke (1987), and Schmitt (1984) describe various applications of simulation for capacity planning where resources are constrained and the production requirements are uncertain. The constrained resources can include machines, people, and/or transportation capacity. Many of these applications involve the development of mathematical formulations of the underlying problems and the use of simulation to evaluate alternatives, as these formulations are generally not solvable.

Early operational scheduling work involved using simulation models to evaluate priority scheduling rules (Berry, 1972; Stecke and Solberg, 1981; Tang et al., 1993). The basic idea is to enumerate a set of priority scheduling or *dispatching* rules and to then use a simulation model of the manufacturing system to evaluate the perfor-

mance under different combinations of the rules. Other scheduling-related work involves the job shop scheduling problem (Koh et al., 1996), static and dynamic flexible flowline scheduling (Bengu, 1994), and FMS machine grouping and loading (Stecke and Raman, 1994). Kim et al. (1998) and Sandell and Srinivasan (1996) describe applications of simulation for operational planning and scheduling in the semiconductor industry. Semiconductor manufacturing involves complex process routings and uncertainty in process yields, making it an ideal area for simulation-based analysis.

More recently, Duvivier et al. (2005) describe the usage of simulation and optimization to assess different scheduling strategies and performance measures. Kumar et al. (2007) describe using Excel and DES (Witness) in order to build a decision support system that can calculate throughput ratios and material requirements under different ratios.

16.2.2.2 Real-Time Control of Manufacturing Systems

The use of simulation for real-time control of manufacturing systems is a relatively new area. Most of the early work in this area focused on using a simulation model to evaluate alternate operational strategies based on the current state of a manufacturing system. Wu and Wysk (1988) describe such a use of simulation as a “look-ahead” evaluator for a flexible manufacturing system. Using this methodology, the simulation is initialized from the state of the system and is then used to analyze various operational strategies based on the system’s current state. So when the real system reaches a point in time where an operational decision needs to be made (e.g., a machine fails or another unexpected event occurs), the simulation model is used as a decision support tool.

Smith et al. (1994) describe a related methodology where the simulation model is connected to the control system of the FMS. The simulation is then used not only to evaluate operational alternatives in look-ahead mode, but also to control the system in real-time mode. Bodner and Reveliotis (1997) describe similar developmental work using an object-oriented simulation paradigm. Smith (2003) discusses several other research applications of this type of simulation.

Recently, there has been significant interest in using DES and emulation models during the *commissioning* process of programmable logic controller (PLC)-based control systems (Miles and Siddeley, 1989; Mueller, 2001; Schiess, 2001; Versteeg and Verbraeck, 2002; Vorderwinkler et al., 1999; Smith and Cho, 2008). Commissioning is the process of a control system customer verifying that the developed control system meets the design specifications. The traditional commissioning process can be quite expensive as it generally involves test running the control system on the actual hardware. As such, the manufacturing system is running and, hence, requires the human operators, but is not producing products.

The idea behind “soft commissioning” (Vorderwinkler et al., 1999) is that a DES mimics the actual hardware of the manufacturing system so that the PLC-based control system thinks that it is actually connected to the real system. This allows the control system testing (and part of the commissioning process) to be done in

the laboratory without the use of the real system. Mueller (2001) describes such a simulation-based commissioning system called Polysim. Polysim uses the commercial simulation package AutoMod and has been successfully used for several control system development projects. Smith and Cho (2008) illustrate how the commercial simulation package Arena can be used in a similar manner to interface directly with a PLC control system.

From an ABM application standpoint, there are many applications that fall into the control classification. For instance, Bussman and Sieverding (2001) describe an agent-based system generated for a DaimlerChrysler engine assembly plant in Stuttgart, Germany. This paper shows how agents might be used beneficially to increase the flexibility in the control of a production line. The assembly process has five sections: engine block assembly, cylinder head assembly, final assembly, test field, and shipping area. The assembly system contains almost no interprocess buffers between sections of the assembly line. The researchers first analyzed the existing condition to realize the problems with the system. The first big problem with the existing system was that it was very dependent on the disturbances. In other words, in case of machine breakdown or supply shortage, the whole line had to stop working because of the resulting blockage for the upstream machines and starvation for the downstream machines.

The second problem of the existing system is that it cannot be easily adjusted to a higher-level production in case of an increase in demand. The response to the increase in demand would be possible by either running overtime or building a second assembly line. Both of those options were not feasible because running overtime was limited and adding a second line would have greatly increased the unit production cost. Therefore, the goal of this case is to increase the robustness and adjustability of the assembly line by making a big change in the control of the assembly line while keeping a high volume and low cost per product.

The researchers started improving the system in three phases. In the first phase, the existing layout of the assembly line was expanded by flexible buffers located along the assembly line so that they could be used as additional buffering in case of a disturbance in the main line. The transportation between the main assembly line and flexible buffers would be done by an AGV. In the second phase, the researchers introduced the multifunctional stations to the assembly line. Multifunctional stations were able to replace more than one station in the line in case of a disturbance or bottleneck. In the third and last phase, the multifunctional stations were converted to or replaced by more dedicated assembly stations and were connected.

After making these changes to the assembly line in three phases as indicated above, the researchers needed a more sophisticated control system. This was because a part in the original case had no option but to be processed by the next downstream machine. However, in the new case, a part might be processed by the next downstream machine or a multifunctional station or it might be stored in a flexible buffer, which requires a new control mechanism.

In order to provide this kind of control, the researchers used a methodology designed for agent-oriented production control. For this purpose, they defined some

agents such as multifunctional station, AGV, and so on, and they set the rules for agent communication. Finally, the researchers had a chance to compare the original system to the new system. The results revealed that the new system outperformed the original system in terms of productivity. In addition, the new system had higher production volume or, in other words, scalability in case of demand increase.

16.2.2.3 Operating Policies

According to Smith (2003), operating policies deal with the configuration of system inputs such as part mix, worker assignments, and part dispatching rules concurrently. Some of the papers from the survey will be described here to give the idea of simulation implementation in operating policies. Bai et al. (1996) conduct a simulation study to examine the performance of an example production system under different control policies. Evans et al. (1992) use simulation to compare alternative order release, inventory control, and dispatching policies.

Similarly, there are some newer studies of simulation. Creighton and Nahavandi (2003) describe a case study showing that DES can be used to support a robust production design. The simulation model was used to identify critical design parameters and decide on the operating policies that will increase the robustness of the system. Lavoie et al. (2006) describe a production system consisting of multiple tandem machines with random failures and try to identify the production rates of the machines in order to minimize the total inventory and backlog costs. Chen and Cochran (2005) explain how manufacturing rules such as line balancing, on-time delivery, and bottleneck optimization can affect a system output such as effective WIP, on-time delivery, and bottleneck loading, and the results show that each rule outperforms the others on a different performance criterion. Sabuncuoglu and Lahmar (2003) examine the interaction of grouping policies (aggregation and disaggregation) in an FMS by taking into account variables such as buffer size, routing flexibility, and so on, and they state that although aggregation lowers the setup times, it does not always provide the best performance.

16.2.2.4 Performance Analysis

Every simulation application basically includes performance analysis due to the fact that simulation models are descriptive (Smith, 2003). However, in this section we discuss papers that explicitly deal with performance analysis rather than any of the other purposes of simulation mentioned above. There are also many papers listed under this subcategory in the survey by Smith (2003). Chan (1995) conducts a simulation study to assess the performance of an automated manufacturing system as a function of demand. Similarly, Klitz (1983) uses simulation to predict and optimize the performance of an automated manufacturing line. Some other papers related to performance analysis are given below.

Pradhan et al. (2008) describe the performance analysis for a Markovian type manufacturing system with product failures and reentrant flows by comparing simulation results with analytical models. Eken and Ornek (2008) describe how

DES and design of experiments can be combined for performance analysis (flow times) with different layout alternatives and other predefined manufacturing parameters and analyze their interactions. Wang et al. (2007) describe how system performance depends on varying the numbers of workers and the numbers of workstations in a system of linear walking worker assembly lines.

Roux et al. (2008) describe combining simulation and optimization in order to analyze maintenance strategy performance. Feyzoglu et al. (2005) describe how to size system resources while satisfying system requirements by implementing simulation and regression metamodeling. Ayag (2007) implements AHP in the first place in order to eliminate some of the alternatives of a new machine that is going to be used in the existing manufacturing plant. After choosing the best alternatives, a simulation study has been conducted to model the current system and analyze the results to choose the best machine alternative providing the minimum unit investment cost ratio.

Dhouib et al. (2008) describe the analysis of nonhomogeneous production lines with no intermediate buffers by comparing the simulation results with four different analytical techniques to see if the analytical results can explain the behavior of these types of manufacturing systems. Owen and Blumenfeld (2007) analyze the relationship between the operating speed and the quality and throughput of a manufacturing system. The results state that up to a certain point, increasing the operating speed also increases the throughput rate. However, after a certain speed, the throughput rate goes down due to the scrap, rework, or repair.

Li et al. (2009) describe a data-driven approach to bottleneck detection in serial lines by taking into account the fact that bottleneck stations cause blocking of the upstream machines while causing the downstream machines to starve. The authors use DES for verification of their model. Gershwin and Werner (2007) analyze a closed-loop system with finite buffers by examining the effects of different numbers of machines, buffer sizes, and so on by building an analytical method and simulation model. A simulation model was used in order to validate the analytical model and its performance.

So far, the use of simulation in the manufacturing domain has been presented by providing some earlier applications in many areas of manufacturing. However, there are also other fields of manufacturing where simulation is also used such as the human factor in manufacturing. For instance, Baines et al. (2004) describe the integration of existing human performance models into DES models of manufacturing systems. For this purpose, they chose a part of the assembly line of a car manufacturing system since it is labor intensive. Then they implemented two models, one indicating the production decrease of workers due to aging and the other one dealing with how performance varies during the day.

Stratman et al. (2004) describe how DES and worker mix (permanent vs. temporary) can affect direct production cost and quality cost. The performance of the manufacturing system is explicitly captured in the model by the production labor and quality costs by analyzing models consisting of different labor force compositions and how their locations in the line greatly affect the performance measure of the manufacturing systems. Lavoie et al. (2006) describe a production system

consisting of multiple tandem machines with random failures and try to identify the production rates of the machines in order to minimize the total inventory and backlog costs.

16.3

Agent-Directed Simulation for Manufacturing Systems

Recent work in manufacturing systems control has focused on using decentralized and distributed control schemes to address flexibility and scalability concerns associated with centralized methods. Centralized methods often involve classical control-theoretic techniques for the analysis and design of small-scale systems. For instance, traditional approaches to shop-floor scheduling utilize a single centralized scheduler, which, according to Parunak (1993), is analogous to use of a sequential computer program, which is effective in the absence of disturbances on the shop floor. On the other hand, since real manufacturing systems are not deterministic and are comprised of many physically distributed materials and resources that operate in a concurrent dynamic fashion, shifting control close to units improves resiliency and adaptability, while avoiding central bottlenecks. Therefore, modeling such concurrent and distributed systems in terms of autonomous, decentralized, and cooperative units that are flexible enough to negotiate and, if necessary, alter plans to improve the reliability, safety, and adaptivity of the overall system is advocated in this section. This introduces the use of agent-directed methodologies into the hierarchical system structure defined previously.

16.3.1

Emergent Approaches

The primary issue is to develop methodologies and mechanisms to organize an independent, autonomous, self-organizing, and coordinating collection of control units. This has led researchers to devise solutions and spectrum of decentralized control architectures that range from hierarchical adaptive organizations to completely decentralized frameworks, in which units make their own decisions based on local information and perception of their environments.

The hierarchical approach involves decomposing the system into subsystems connected to each other via weak subordination relations. Spatial (Mesarovic et al., 1970; Singh, 1980) methods that focus on multilayered strategy and temporal decomposition (e.g., frequency decomposition) (Gershwin, 1989) are common techniques. Others (Jones and Saleh, 1990) have proposed architectures that combine temporal and spatial methods.

Besides the hierarchical approach, significant research has been conducted to utilize distributed problem solving techniques, where competitive (combination of cooperative and competitive) solutions are devised in terms of loosely coupled collection of decentralized problem solvers. For instance, the metaphor of negotiation, which is based on the work of Davis and Smith (1983) on distributed sensor

systems, is leveraged to develop manufacturing system models of cooperative behavior. The well-known *contract-net protocol*, which involves a distributed task allocation and coordination strategy, is extended by Parunak (1987) in YAMS (yet another manufacturing system). Along with the negotiation metaphor, the system uses an open and extensible system model, where problem solvers are included and/or removed on demand.

16.3.2

Agent-Based Manufacturing

The agent paradigm advocates the use of autonomous entities that are capable of perceiving their environment to not only react, but also proactively change their behavior through deliberative reasoning to adapt themselves to changes in their environment. As such, agent technology focuses on advanced methodologies to facilitate communication, collaboration, coordination, and conflict resolution among multiple agents, as well as deliberation mechanisms so that agents can effectively operate under uncertainty in response to environmental perturbations. Since manufacturing applications are characteristically modular, decentralized, and open to change in environmental circumstances, they provide a fertile environment to explore the use of agents. As such, recently there has been significant research on the application of multiagent systems and distributed artificial intelligence techniques in manufacturing systems' modeling, design, and development domain. As shown in Table 16.1, these applications range from product design (e.g., PACT (Cutkosky et al., 1993)), ACDS (Darr and Birmingham, 1996), RAPPID (Parunak, 1999), en-

Table 16.1 Applications of agents in manufacturing systems modeling.

Domain	Application
Product design	PACT: An experiment in Integrating Concurrent Engineering Systems (Cutkosky et al., 1993) ACDS: Attribute Space Representation for Concurrent Engineering (Darr and Birmingham, 1996) RAPPID: Responsible Agents for Product-Process Integrated Design (Parunak, 1999)
Enterprise Integration and Supply Chain Management	ISCM: Integrated Supply Chain Management System (Fox et al., 1993) AIMS: Agile Infrastructure for Manufacturing Systems – A Vision for Transforming the US Manufacturing Base (Park et al., 1993) CIMPLEX: A Multi-agent System for Enterprise Integration (Peng et al., 1998)
Planning and Scheduling	AARIA: Agents and Internet: Infrastructure for Mass Customization (Baker et al., 1999) MASCADA: Designing Agents for Manufacturing Process Control (Bruckner et al., 1998) YAMS: Manufacturing Experience with the Contract-Net (Parker, 1987).

terprise integration and supply chain management (Fox et al., 1993; Park et al., 1993; Peng et al., 1998), and planning and scheduling (Baker et al., 1999; Bruckner et al., 1998). A comprehensive survey on the use of agent technology in modeling, design, and simulation/implementation of manufacturing systems can be found in Shen et al. (2006).

As an example of the use of agents, consider the following architecture, which is adopted from the MetaMorph I architecture developed by Maturana and Norrie (1996). The objective of the MetaMorph architecture is to enable the adaptation of structure and activities dynamically to tasks that emerge in the manufacturing system. *Mediator* agents enable coordination of resource agents by decoupling the coordination protocol from the component functionalities. Mediators wrap components to augment their behavior to participate in the desired protocol. Communication between mediators is achieved by a brokering scheme, by which messages received by mediators are selectively broadcast to a group of agents. Coordination within the MataMorph architecture through a process of subtasking, coordination cluster creation, and task execution. Global system objectives are sought while enforcing local constraints via dynamically formed clusters of resources that emerge on demand arising from the requirements of the manufacturing system.

As shown in Figure 16.1, the approach is used to simulate a distributed concurrent design and manufacturing system in terms of enterprise, design, resource, and simulation mediators. Each resource mediator provides a protocol for the coordination of machines, tools, and other entities on the shop floor. Enterprise integration protocols enable interoperation across agents designed in the form of mediators.

One motivation for the use of agents in representing manufacturing systems is that agents provide a natural metaphor in conceptualizing entities that have state-dependent behavior. Instead of viewing the system in terms of events, activities,

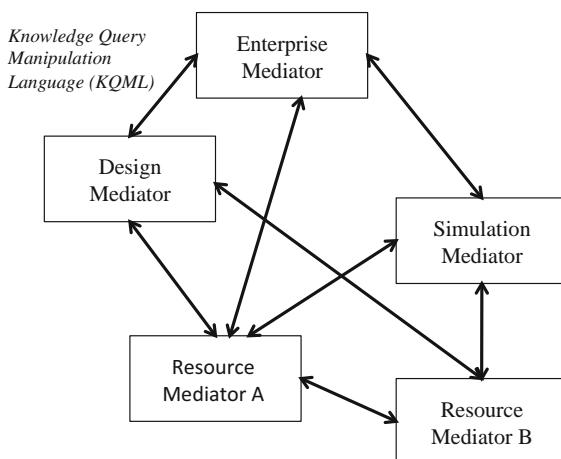


Fig. 16.1 MetaMorph architecture (Shen et al., 2000).

and their cause-and-effect relations, agents enable the interpretation of manufacturing system structures and behaviors in terms of self-contained entities that interact with each other to provide the desired functionality. That is, agents are used to represent entities in the physical world, such as machines, workers, tools, products, parts, and operations. On the other hand, increased communication, collaboration, task allocation, and coordination strategies that are needed to govern the agents and their behavior lead to performance issues and agent management complexities.

16.3.3

The Holonic Approach: Hierarchic Open Agent Systems

The shortcomings of the centralized approach are clearly revealed under the rubric of the FMS domain. Specifically, centralized control techniques are presented as the main reasons for systems that are inflexible, fragile, and difficult to maintain. The research on application of distributed control techniques led many to have the impression that an autonomous, distributed, and cooperative approach is required to address these issues. That is, manufacturing system designs that are capable of reconfiguration, robust (e.g., capable of recovering from disturbances), and responsive are sought in terms of distributed problem solving schemes. However, in practice, early results were not as successful as expected. As a result, to address increasingly stringent user requirements involving high quality, low production cost, and manufacturing complexity due to intricate dependencies between manufacturing systems, supply chain management issues, and enterprise integration aspects, research in holonic agent-based manufacturing systems emerged as a plausible strategy.

Holonic manufacturing systems (HMSs) emerged as one of the several topic areas under the Intelligent Manufacturing Systems (IMS) program (Christensen, 1994). Holonics (Koestler, 1967) is based on the synergistic combination of the general philosophy of living systems and social organizations with advances in distributed artificial intelligence. The objective of the HMS approach to manufacturing system modeling, design, simulation, and implementation is to attain the benefits of organisms and societies (e.g., robustness, adaptability in the face of change, efficiency, stability). An HMS consists of autonomous, self-reliant manufacturing units called *holons*. Holons cooperate to achieve global manufacturing system objectives. An important concept pertaining to HMS is the notion of functional decomposition. Dealing with complexity involves decomposing the system into smaller parts; therefore holons may aggregate other holons, leading to holarchies, or groups of autonomous and cooperative basic holons and recursive holons that are themselves holarchies. Holons are defined by the HMS consortium as autonomous and cooperative building blocks of manufacturing systems for transporting, transforming, storing, or validating information and physical agents.

Koestler (1967) originally proposed the *holon* concept as an attempt to explain the general principles of open hierarchical complex systems. A holon is defined as a self-contained *whole* while acting as part of a whole in a hierarchically ordered system. A hierarchical system is a system that is composed of interrelated subsys-

Holarchy

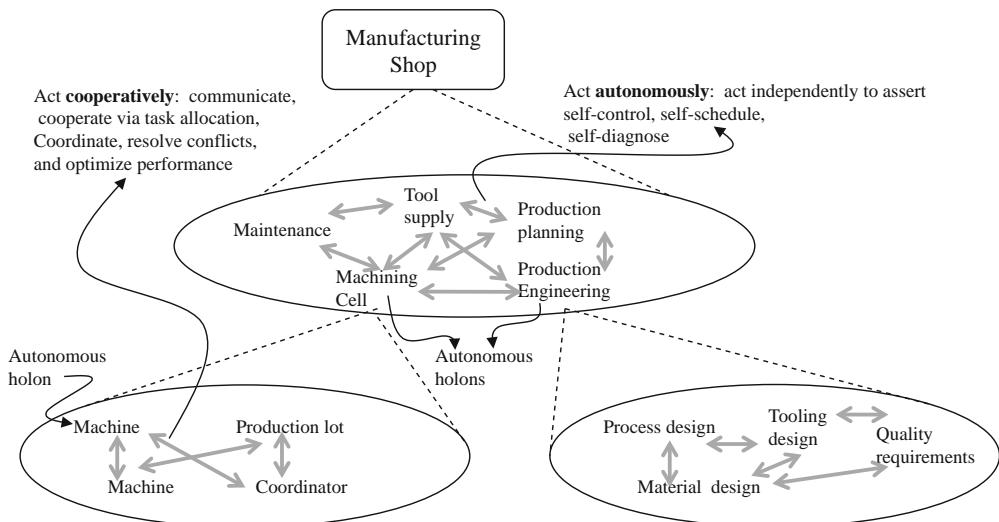


Fig. 16.2 Autonomous holarchies.

tems, each of the latter in turn hierarchical in structure until we reach the lowest level of elementary subsystems or components. Unlike traditional hierarchical systems, holarchies embedded in a hierarchical structure are not necessarily related via subordination relation.

A composite (aggregate) holon, also called a *whole* or *holarchy*, can be seen as an operational closure that encapsulates interactions among its constituent holons. For instance, in Figure 16.2, the manufacturing shop is a holarchy that contains maintenance, tool supply, machining cell, production engineering, and production planning holons, where machining cell and production engineering is represented as holarchies comprised of active, autonomous holons at a lower level. In such hierarchically ordered complex systems, holons possess hierarchic awareness, where each composite holon is aware of both its own goals attained by its constituent holons and its goal as part of the context in which it is embedded. This awareness results in both self-assertiveness (differentiation) and integrative tendencies. These opposing tendencies, autonomy and cooperation, reflect the partness and wholeness attributes of the holarchy, respectively.

A significant aspect of the holonic approach is the specification of a design solution as a hierarchic network of holons. The emergent equilibrium state of the network and its structure may be interpreted and used as a criterion to assess the characteristics of the solution. The degree of integrated differentiation of the network structure is considered as a measure of complexity and effectiveness of the solution. Some degree of differentiation through autonomy is preferable to facilitate improvement of robustness. However, it is also expected that holarchies maintain effective transactions with the other holarchies or holons that represent other sus-

systems. This improves the stability of the solutions generated by the holarchy. The failure to achieve a balance between differentiation and integration results in a decrease in robustness, as well as coherence in attaining the objectives of the manufacturing system. This can take form in different ways: (1) limited differentiation and consequently limited development of special mechanisms that are relevant to a specific functionality; (2) excessive dependence on highly centralized holarchies, where some holons become highly subordinate to characteristics of competing holarchies; (3) decreased exchanges between holons, which results in restricted flows of information that limit the usefulness and extensibility of local solutions; and (4) increased conflicts between holons across distinct holarchies and consequently limited coherence of the overall solution.

The types of manufacturing systems envisaged by these concepts promise many advantages. Practical realization of these advantages requires further research and exploration in better understanding the dynamics and mode of operation of holarchies, as well as their implementation in terms of agent technologies.

16.4 **Summary**

Manufacturing accounts for a significant portion of the world's economic output, and the corresponding manufacturing systems can be extremely complex. Increased global competition has forced the organizations that operate these systems to do so as cost-effectively and as flexibly as possible and to have significant interactions with other organizations within their manufacturing supply chain. This has impacted not only the organizations that operate manufacturing systems, but also those organizations responsible for designing and implementing these systems. Given the complexity of the systems and the needs for optimized performance and flexibility, significant and ongoing analysis is required during the design and operation of these systems.

This chapter has focused on the applications of simulation for manufacturing systems design, analysis, and operation. Emphasis has been given to traditional discrete-event applications as well as to more recent agent-directed simulation applications. Traditional DES has long been a staple in the manufacturing systems analysis domain. More recently, agent-directed methodologies have been used to address some of the limitations of centralized decision making in hierarchical control environments.

Applications of simulation in the manufacturing domain can be broadly classified as either analysis related or operations related. Analysis-related applications focus on estimating and ultimately improving the performance of the underlying manufacturing systems. Simulation has been broadly applied to systems in this area since the early 1970s, and the area is dominated by the traditional DES applications. Operations-related applications, which generally focus on either short-term policy selection or real-time control, appeared in these areas in the 1980s, with significant growth occurring in the 1990s. Agent-directed simulation appears to be

perfect for these areas as short-term policy selection and control both involve explicitly considering the “current state” of the system, complex interaction between model entities, and identification and consideration of emergent behavior.

Given the pace with which globalization has impacted the worldwide manufacturing sector, it is unlikely that the need for tools to support analysis and control of complex manufacturing systems will subside any time soon. In fact, these needs will likely grow as more specialized manufacturing becomes common and emerging economies begin to fully participate in the worldwide manufacturing sector. As such we fully expect the use of simulation (discrete-event and agent-directed) to increase significantly over the next several years.

References

- M. Andersson and G. Olsson. A simulation-based decision support approach for operational capacity planning in a customer order driven assembly line. *Proceedings of the 1998 Winter Simulation Conference*, pp. 935–941, 1998.
- S. Angra, R. Sehgal, and Z.S. Noori. Cellular manufacturing time-based analysis to the layout problem. *International Journal of Production Economics*, 112(1), 427–438, 2008.
- Z. Ayag. A hybrid approach to machine-tool selection through AHP and simulation. *International Journal of Production Research*, 45(9), 2029–2050, 2007.
- S.X. Bai, V. Nambiar, C.H. Peredo, and Y. Tseng. Evaluation of production control system policies via simulation experiment design and analysis. *5th Industrial Engineering Research Conference*, pp. 429–434, 1996.
- T. Baines, S. Mason, P. Siebers, and J. Ladbrook. Humans: the missing link in manufacturing simulation? *Simulation Modelling Practice and Theory*, 12(7–8), 515–526, 2004.
- B. Baker, V. Parunak, K. Erol. Manufacturing over the Internet and into your living room: Perspectives from the AARIA project. *EEE Internet Computing*, Fall 1999.
- F.L. Bellifemine, G. Caire, and D. Greenwood. *Developing multi-agent systems with JADE*. John Wiley & Sons, 2007.
- G. Bengu. Simulation-based scheduler for flexible flowlines. *International Journal of Production Research*, 32(2), 321–344, 1994.
- W. Berry. Priority scheduling and inventory control in job lot manufacturing systems. *AIEE Transactions*, 4(4), 267–277, 1972.
- D.A. Bodner and S.A. Reveliotis. Virtual factories: an object-oriented simulation-based framework for real-time fms control. *IEEE Symposium on Emerging Technologies and Factory Automation, EFTA*, pp. 208–213, 1997.
- J. Brown and K. Rathmill. The use of simulation modelling as a design tool for fms. *Proceedings of the 2nd International Conference on FMS*, pp. 197–214, 1983.
- S. Bruckner, J. Wyns, P. Peeters and M. Kollingbaum. Designing agents for manufacturing process control. *Proceedings of Artificial Intelligence and Manufacturing Research Planning Workshop*, pp. 40–46, 1998.
- P.J. Buckley. The impact of the global factory on economic development. *Journal of World Business*, 44(2), 131–143, 2008.
- Bureau of Economic Analysis. Gross domestic product by industry accounts. Retrieved on November 07, 2008 from http://www.bea.gov/bea/industry/gpotables/gpo_action.cfm, 2008.
- S. Bussman and J. Sieverding. Holonic control of an engine assembly plant: an industrial evaluation. *IEEE International Conference on Systems, Man, and Cybernetics*, 5, 3151–3156, 2001.
- M. Cantamessa. Agent-based modeling and management of manufacturing systems. *Computers in Industry*, 34(2), 173–186, 1997.
- R. Caprihan and S. Wadhwa. Impact of routing flexibility on the performance of an

- fms- a simulation study. *International Journal of Flexible Manufacturing Systems*, 9(3), 273–298, 1997.
- F.T.S. Chan. Using simulation to predict system performance: a case study of an electro-phoretic deposition plant. *Integrated Manufacturing Systems*, 6(5), 27–38, 1995.
- F.T.S. Chan and R.W.L. Ip. Multi-attribute analysis of flexible manufacturing systems via simulation. *IEEE International Conference on Systems, Man and Cybernetics*, 2, 1327–1332, 1995.
- Y. Chang, R.S. Sullivan and J.R. Wilson. Using slam to design the material handling system of a flexible manufacturing system. *International Journal of Production Research*, 24(1), 15–26, 1986.
- H. Chen and J.K. Cochran. Effectiveness of manufacturing rules on driving daily production plans. *Journal of Manufacturing Systems*, 24(4), 339–351, 2005.
- J.C. Chen, C.C. Chien, H.H. Tseng, C.C. Wang and T.C. Chang. A systematic approach for ic fab design. *Proceedings of the Sixth International Symposium on Semiconductor Manufacturing*, 4, 15–18, 1997.
- J. Christensen. Initial architecture, standards, and directions. *The 1st European Conference on Holonic Manufacturing Systems*, pp. 1–20, 1994.
- J.K. Cochran and H.A. Kaylani. Optimal design of a hybrid push/pull serial manufacturing system with multiple part types. *International Journal of Production Research*, 46(4), 949–965, 2006.
- D. Creighton and S. Nahavandi. Application of discrete event simulation for robust system design of a melt facility. *Robotics and Computer Integrated Manufacturing*, 19(6), 469–477, 2003.
- M. Cutkosky, R. Englemor, R. Fikes, T. Gruber, M. Gennsereth, W. Mark, J. Tenenbaum and J. Weber. PACT: An experiment in integrating concurrent engineering systems. *IEEE Computer*, 26, 28–37, 1993.
- T. Darr and W. Birmingham. An attribute-space representation and algorithm for concurrent engineering. *Ai EDAM*, 10(1), 21–35, 1996.
- R. Davis and R. Smith. Negotiation as a metaphor for distributed problem solving. *Artificial Intelligence*, 20, 469–477, 1983.
- L. Deroussi, M. Gourgand and N. Tchernev. Combining optimization methods and discrete event simulation: A case study in flexible manufacturing systems. *2006 International Conference on Service Systems and Service Management*, 1, 495–500, 2006.
- K. Dhouib, A. Gharbi and S. Ayed. Availability and throughput of unreliable, unbuffered production lines with non-homogeneous deterministic processing times. *International Journal of Production Research*, 46(20), 5651–5677, 2008.
- B. Dubiel and O. Tsimhoni. A simulation-based decision support approach for operational capacity planning in a customer order driven assembly line. *Proceedings of the 37th conference on Winter simulation*, pp. 1029–1037, 2005.
- D. Duvivier, V. Dhaevers and A. Artiba. Simulation-based performance measurement and analysis: an industrial application. *International Journal of Computer Integrated Manufacturing*, 18(5), 402–407, 2005.
- B.Y. Eken and A.M. Ornek. A simulation based experimental design to analyze factors affecting production flow time. *Simulation Modelling Practice and Theory*, 16(3), 278–293, 2008.
- G.W. Evans, W.E. Biles and S.M. Alexander. Large scale simulation model for analyzing the production of pipe valves and fittings. *Simulation*, 59(6), 366–374, 1992.
- O. Feyzoglu, H. Pierrevat and D. Deflandre. A simulation-based optimization approach to size manufacturing systems. *International Journal of Production Research*, 43(2), 247–266, 2005.
- M. Fox, J. Chionglo and M. Barbuceanu. The integrated supply chain management system. *Technical report, Department of Industrial Engineering*, University of Toronto, 1993.
- S.B. Gershwin and L.M. Werner. An approximate analytical method for evaluating the performance of closed-loop flow systems with unreliable machines and finite buffers. *International Journal of Production Research*, 45(14), 3085–3111, 2007.
- S. Gershwin. Hierarchical Flow Control: A Framework for Scheduling of and Planning Discrete Events in Manufacturing

- Systems. *Proceedings of the IEEE*, 77(1), 195–209, 1989.
- D. Gianni. Bringing discrete event simulation concepts into multi-agent systems. *Tenth International Conference on Computer Modelling and Simulation*, pp. 186–191, 2008.
- D. Gien and S. Jacqmart. Design and simulation of manufacturing systems facing imperfectly defined information. *Simulation Modelling Practice and Theory*, 13(6), 465–485, 2005.
- A. Greasley. Using simulation for facility design: A case study. *Simulation Modelling Practice and Theory*, 16(6), 670–677, 2008.
- Q. Hao and W. Shen. Implementing a hybrid simulation model for a kanban-based material handling system. *Robotics and Computer-Integrated Manufacturing*, 24(5), 635–646, 2008.
- G.K. Hutchinson. The design of an automated material handling system for a job shop. *Computers in Industry*, 4(2), 139–146, 1983.
- M. Jain, S. Maheshwari and K.P.S. Baghel. Queueing network modelling of flexible manufacturing system using mean value analysis. *Applied Mathematical Modelling*, 32(5), 700–711, 2008.
- A. Jones and A. Saleh. A multi-level/multi-layer architecture for intelligent shop-floor control. *International Journal of Computer-Integrated Manufacturing*, 3(1), 60–70, 1990.
- Y.D. Kim, D.H. Lee, J.U. Kim and H.K. Roh. A simulation study on lot release control, mask scheduling, and batch scheduling in semiconductor wafer fabrication facilities. *Journal of Manufacturing Systems*, 17(2), 107–117, 1998.
- A.S. Kiran, A. Schloffer and D. Hamkins. Integrated simulation approach to design of flexible manufacturing systems. *Simulation*, 52(2), 47–52, 1989.
- J.K. Klitz. Simulation of an automated logistics and manufacturing system. *European Journal of Operational Research*, 14(1), 36–40, 1983.
- A. Koestler. *The Ghost in the Machine*. Arcana, London, 1967.
- K-H. Koh, R. DeSouza, and N.-C. Ho. Multi-processor distributed simulation for job-shop scheduling: boon or bane? *International Journal of Computer Integrated Manufacturing*, 9(6), 434–442, 1996.
- S. Kumar, D.A. Nottestad and J.F. Macklin. A profit and loss analysis for make-to-order versus make-to-stock policy supply chain case study. *The Engineering Economist*, 52(2), 141–156, 2007.
- S.N. Kumar and R. Sridharan. Simulation modeling and analysis of tool sharing and part scheduling decisions in single-stage multimachine flexible manufacturing systems. *Robotics and Computer-Integrated Manufacturing*, 23(4), 361–370, 2007.
- M. Kuroda and T. Tomita. Robust design of a cellular-line production system with unreliable facilities. *Computers & Industrial Engineering*, 48, 537–551, 2005.
- P. Lavoie, J.P. Kenne, and A. Gharbi. Production control and combined discrete/continuous simulation modeling in failure-prone transfer lines. *International Journal of Production Research*, 45(24), 5667–5685, 2006.
- L. Li, Q. Chang and J. Ni. Data driven bottleneck detection of manufacturing systems. *International Journal of Production Research*, 47(18), 5019–5036, 2009.
- P.C. Lockemann and J. Nimis. Flexibility through Multiagent Systems: Solution or Illusion? *SOFSEM 2004: Theory and Practice of Computer Science*, pp. 211–224. Springer, Berlin/Heidelberg, 2004.
- R. Logendran and D. Talkington. Analysis of cellular and functional manufacturing systems in the presence of machine breakdowns. *International Journal of Production Economics*, 53(3), 239–256, 1997.
- F. Maturana and D. Norrie. Multi-agent architectures for distributed manufacturing. *Journal of Intelligent Manufacturing*, 7(3), 257–270, 1996.
- M. Mesarovic, D. Macko, and Y. Takahara. *Theory of Hierarchical Multi-level Systems*. Academic Press, 1970.
- T.I. Miles and H. Siddeley. Using discrete-event computer simulation to test control systems. *Proceedings of the 1989 Winter Simulation Conference*, pp. 848–858, 1989.
- J.M. Moore. *Plant Layout and Design*. MacMillan & Co., New York, NY, 1962.
- G. Mueller. Using emulation to reduce commissioning costs on a high-speed bottling line. *Proceedings of the 2001 Simulation Conference*, pp. 1461–1462, 2001.

- J.H. Noguera and E.F. Watson. Analyzing throughput and capacity of multiproduct batch processes. *Journal of Manufacturing Systems*, 23(3), 215–228, 2004.
- J. Odell. Agent technology: What is it and why do we care? Retrieved on September 19, 2008 from http://www.jamesodell.com/Cutter-Exec_Rpt-July_2007.pdf, 2007.
- J.H. Owen and D.E. Blumenfeld. Effects of operating speed on production quality and throughput. *International Journal of Production Research*, 2007.
- M. Ozden. A simulation study of multiple-load-carrying automated guided vehicles in a flexible manufacturing system. *International Journal of Production Research*, 26(8), 1353–1366, 1988.
- H. Park, J.A. Tenenbaum and R. Dove. Agile infrastructure for manufacturing systems: A vision for transforming the US manufacturing base. *Defense Manufacturing Conference*, 1993.
- H. Parunak. Manufacturing experience with the contract-net. *Distributed Artificial Intelligence*, Pitman, London, 1987.
- H. Parunak. Autonomous agent architectures: A non-technical introduction. *Industrial Technology Institute Report*, 1993.
- H. Parunak. Industrial and practical applications of DAI. In: G. Weiss (ed.) *Multi Agent Systems: A Modern Approach to Distributed Artificial Intelligent Systems*, MIT Press, 1999.
- Y. Peng, T. Finin, Y. Labrou, B. Chu, J. Long, W. Tolone and A. Bougannam. A multi-agent system for enterprise integration. *Proceedings of PAAM'98*, pp. 533–548. 1998.
- S. Pradhan, P. Damodaran and K. Srihari. Predicting performance measures for markovian type of manufacturing systems with product failures. *European Journal of Operational Research*, 184(2), 725–744, 2008.
- A.A.B. Pritsker. Applications of slam. *IIE Transactions*, 14(1), 70–78, 1982.
- O. Roux, M.A. Jamali, D.A. Kadi and E. Chatelet. Development of simulation and optimization platform to analyse maintenance policies performances for manufacturing systems. *International Journal of Computer Integrated Manufacturing*, 21(4), 407–414, 2008.
- I. Sabuncuoglu and M. Lahmar. An evaluative study of operation grouping policies in an fms. *International Journal of Flexible Manufacturing Systems*, 15(3), 217–239, 2003.
- R. Sandell and K. Srinivasan. Evaluation of lot release policies for semiconductor manufacturing systems. *Proceedings of the 1996 Winter Simulation Conference*, pp. 1014–1022, 1996.
- M. Savsar. Flexible facility layout by simulation. *Computers and Industrial Engineering*, 20(1), 155–166, 1991.
- C. Schiess. Emulation: debug it in the lab, not on the floor. *Proceedings of the 2001 Winter Simulation Conference*, pp. 1463–1465, 2001.
- T.G. Schmitt. Resolving uncertainty in manufacturing systems. *Journal of Operations Management*, 4(4), 331–346, 1984.
- T.J. Schriber and K.E. Stecke. Using mathematical programming and simulation to study fms machine utilizations. *Proceedings of the 1987 Winter Simulation Conference*, pp. 725–730, 1987.
- W. Shen, D. Norrie and J. Barthes. *Multi-Agent Systems for Concurrent Design and Manufacturing*, Taylor and Francis, 2000.
- W. Shen, Q. Hao, H.J. Yoon, and D.H. Norrie. Applications of agent-based systems in intelligent manufacturing: An updated review. *Advanced Engineering Informatics*, 20(4), 415–431, 2006.
- M. Singh. *Dynamic Hierarchical Control*. North-Holland, Amsterdam, 1980.
- J.S. Smith and Y. Cho. Offline commissioning of a plc-based control system with arena. *Proceedings of the 2008 Winter Simulation Conference*, pp. 1802–1810, 2008.
- J.S. Smith, R.A. Wysk, D.T. Sturrock, S.E. Ramaswamy, G.D. Smith and S.B. Joshi. Discrete event simulation for shop floor control. *Proceedings of the 1994 Winter Simulation Conference*, pp. 962–969, 1994.
- J.S. Smith. Simulation for manufacturing systems design and operation. *Journal of Manufacturing Systems*, 22(2), 157–171, 2003.
- K.E. Stecke and N. Raman. Production planning decisions in flexible manufacturing systems with random material flows. *IIE Transactions*, 26(5), 2–17, 1994.
- K.E. Stecke and J.J. Solberg. Loading and control policies for a flexible manufacturing

- system. *International Journal of Production Research*, 19(5), 481–490, 1981.
- J.K. Stratman, A.V. Roth and W.G. Gilland. The deployment of temporary production workers in assembly operations: a case study of the hidden costs of learning and forgetting. *Journal of Operations Management*, 21(6), 689–677, 2004.
- S. Taj, D.S. Cochran, J.W. Duda and J. Linck. Simulation and production planning for manufacturing cells. *Proceedings of the 1998 Winter Simulation Conference*, pp. 973–978, 1998.
- L.-L. Tang, Y. Yih and C.-Y. Liu. A study on decision rules of a scheduling model in an fms. *Computers in Industry*, 22(1), 1–13, 1993.
- I. Um, H. Lee and H. Cheon. Determination of buffer sizes in flexible manufacturing system by using the aspect-oriented simulation. *International Conference on Control, Automation and Systems*, pp. 1729–1733, 2007.
- C. Versteegt and A. Verbraeck. The extended use of simulation in evaluating real-time control systems of agvs and automated material handling systems. *Proceedings of the 2002 Simulation Conference?*, pp. 1659–1666, 2002.
- M. Vorderwinkler, T. Eder, R. Steringer and M. Schleicher. An architecture for soft-commissioning, verifying control software by linking discrete event simulation to real control systems. *Proceedings of the 1999 Simulation Conference*, pp. 191–198, 1999.
- Q. Wang, G.W. Owen and A.R. Mileham. Determining numbers of workstations and operators for a linear walking-worker assembly line. *International Journal of Computer Integrated Manufacturing*, 20(1), 1–10, 2007.
- P.S. Welgama and R.G.J. Mills. Use of simulation in the design of a jit system. *International Journal of Operations and Production Management*, 15(9), 245–261, 1995.
- E.J. Williams and H. Celik. Analysis of conveyor systems within automotive final assembly. *Proceedings of the 1998 Winter Simulation Conference*, pp. 915–920, 1998.
- E.J. Williams and A. Gevaert. Pallet optimization and throughput estimation via simulation. *Proceedings of the 1997 Winter Simulation Conference*, pp. 744–749, 1997.
- S.D. Wu and R.A. Wysk. Multi-pass expert control system – a control/scheduling structure for flexible manufacturing cells. *Journal of Manufacturing Systems*, 7(2), 107–120, 1988.

17

Organization and Work Systems Design and Engineering: from Simulation to Implementation of Multiagent Systems

Maarten Sierhuis, William J. Clancey, and Chin H. Seah

17.1

Introduction

In this chapter we present a specific approach to analyzing and designing work systems we call *work systems design* (WSD) that is based on a theory of modeling and simulating work practice. The theory is made explicit in an agent-based modeling and simulation (M&S) environment called *Brahms* (see also Chapter 12). In the next section we start by explaining what WSD is. We then give a short overview of the Brahms language. This overview is in addition to the Brahms description in Chapter 12. It gives a more detailed description of some of the language constructs for modeling work systems. Next, we describe the *from simulation to implementation* software engineering methodology. In this we turn an agent-based simulation of a work system into a distributed multiagent system (MAS). It is here where Brahms differs from other agent simulation environments and where agent-based M&S meets agent-oriented software engineering (Luck and Padgham, 2008). After the description of the methodology, we describe NASA's OCAMS project in which this approach was successfully applied. The OCAMS MAS has been in operation 24/7 for over 6 months as of the writing of this chapter, in NASA's Mission Control Center (MCC) for the International Space Station (ISS) in Houston, TX. In this project we first modeled and simulated the current work system in Brahms as an agent-based simulation of people, artifacts, and environment. We then designed the OCAMS system as a MAS simulation interacting with a simulation of people agents. We end this chapter with some conclusions about the approach and experiences from the OCAMS project.

17.2

Work Systems Design

A work system is a “natural” setting for those who work within it, because every setting is natural for those who frequent it. In this chapter we focus on workplaces

for work systems. A workplace is where the work system comes alive, where the daily work is continuously being performed, based on familiar past performance as well as changes unknown until faced. In other words, work is like a symphony, well rehearsed but always different. It is this “symphony” we are interested in composing (designing) changes, using an engineering method that allows us to predict the impact of designed change on the current system.

A work system covers more than just the people working inside it, just like a whole that is bigger than the sum of its parts. From the perspective of designing and engineering a work system, we see the system as a sociotechnical system of people, artifacts, and (computer) systems and their (joint) activities, organization, and communication. Everything is interconnected (Emery, 1960). The design of a work system sometimes involves the completely new design of people’s tasks, procedures, and activities. However, often a change to a work system is created by the introduction of a new software system that changes the activities of people. It is this change that we want to understand up front in the design of software systems so that we can predict how the system should operate within the work system and how it changes the work.

17.2.1

Existing Work System Design Methods

There are a number of existing methods for analyzing and/or designing work systems. Most of these methods are referred to as “soft system” methods. In these methods working with the people in the workplace is one of the big differences from more traditional (software) engineering methods. A number of methods were developed as byproducts of the “Scandinavian approach” to systems design, called participatory design. Other approaches are byproducts of task analysis performed mostly by cognitive psychologists. Below are four existing approaches from the participatory design realm:

- *Design at Work: Cooperative Design of Computer Systems* (Greenbaum and Kyng, 1991) gives detailed examples, theory, and methods for participatory design. It gives the Scandinavian perspective of defined observational studies of workplaces as a theoretically grounded activity in software engineering.
- *Contextual Design* (Beyer, 1998) can be seen as a method for “contextual inquiry”, including how to observe and work with customers; conduct interviews; model work as organizational flow, task sequences, artifacts, culture, stakeholders, and physical environment; and redesign work.
- *Soft Systems Methodology (SSM)* (Checkland, 1990) describes an engineering method for understanding the “soft” issues of a system, through examples of how different stakeholders use different “lenses” looking at the same problem. Checkland, himself a civil engineer, realized that engineered artifacts were part of a human work system he called an activity system. The process of designing an activity system is a *holonic* modeling process that involves many parties (customers, users, designers, policymakers, etc.).

- *Cognitive Work Analysis* (CWA) (Vicente, 1999) is another method for analyzing work systems and designing software systems based on detailed mapping of information flows and tasks. It was developed by Rasmussen and colleagues (Rasmussen et al., 1994); work models are detailed cognitive models rooted in cognitive task analysis for tool design, and hence observation must be systematically organized to understand the domain.

What all these approaches have in common is that they are mostly *ad hoc* modeling approaches. Most of them are pen-and-paper based and have no specific M&S tool that helps guide what to do; although some of the approaches lend themselves to one or more modeling methods such as system dynamics modeling (Sterman, 2000) for SSM, and GOMS (Caroll, 2003) for CWA.

17.2.2

A Brief History of Work Systems Design

WSD has its roots in business anthropology, which started in the 1980s as the Scandinavian approach to system design (Ehn, 1989; Greenbaum and Kyng, 1991). The use of photography and video in studying workplaces was pioneered in cultural anthropology. Today, digital photography and video make capturing people's work practices in the workplace much easier. Descriptive textual notes of a setting are replaced by high-resolution digital photos or a short video recorded on a digital camera. These recordings are extremely useful in the analysis of how people use space, interact with artifacts, and communicate with each other.

Improvement of procedures and tasks through the design of new technology used in the workplace has taken place throughout the last 50 or so years. Studying schools and other natural settings by developmental psychologists (Lave, 1988) and modeling workflow by management consultants and organizational analysts (Senge, 2006) have led to major fads in business process reengineering. Often it is not the academic studies that create change in the development of methods, but the application of academically rooted and developed methods to real-world problems that move a field forward. Similarly, this happened with the coming of expert systems and knowledge acquisition (Buchanan, 1984). Where computer scientists and business consultants failed, anthropologists, together with computer scientists, succeeded in developing a participatory software engineering practice (Nardi, 1996; Kling and Star, 1998).

For a more complete history of how WSD came to be, we refer the reader to Clancey (2006). At this point in time, we are applying M&S to the understanding of work practices – how people work. This is where social science is finally meeting systems engineering, and a truly holistic human-centered engineering approach – an approach in which people are at the center and people, environment, and systems are understood as an interacting system – is created. This is the topic of the next section.

17.3

Modeling and Simulation of Work Systems

In this chapter we argue that we need to move the design of human work systems from a seemingly mystical social-science art form to systems engineering with, more or less, formal methods, tools, and guidelines. To do this, we need to show how the social-science “methods”, mentioned in the previous section, can be made useful as data-gathering methods, not for ethnographic analysis, but for the purpose of work systems design.

What is needed is a theory of analysis and design of human activity systems. This theory will then form the foundation upon which we can develop a systems engineering approach that has clear guidelines and procedures to follow, a sort of cookbook for the analysis and design of work systems. We argue that modeling, and especially simulation, focuses the attention on systems engineering methods. This is mostly because a computer simulation requires a formal description of the system in order to simulate changes to the system over time, based on the operationalization of theories.

17.3.1

Designing Work Systems: What Is the Purpose and What Can Go Wrong?

If we develop a methodology with appropriate methods for designing work systems, what is its purpose? How many new work systems get developed from scratch? The answer is, not many. There are situations, for example at NASA, where a complete new work system gets designed and put in place for a short duration of, say, three months to ten years. Space mission operations organizations are such an example. However, in most cases design mostly entails changes to existing systems. More often than not there is a current system that exists and the objective is to change just a small part of the current system to make it more efficient or effective, or both. In the first, not-so-frequent, case, M&S is used to design the future system as a computer model and then to simulate it to understand how the system will behave in the future. We use the methods to understand a system in the future before it is actually there. In the second, more frequent, case, the objective is to understand the change trajectory of an existing system and understand how the designed change fits into the current system.

In both cases we are talking about predicting future behavior of people and machines. The danger of an M&S approach is the well-known “garbage-in-garbage-out” dilemma. In the 1990s this issue became well known in the process reengineering fad (Davenport, 1993). Expensive, external business consultants were hired to reengineer a company’s work processes to make them “leaner and meaner”. On top of bringing in outside consultants, who knew very little of the company’s work process, most of the time this was done just prior to or during a company merger. The approach became synonymous with “cutting cost by cutting heads”, while in the meantime management “forgot about the people” (Davenport, 1995). The approach was not driven from understanding the work of the people within the

system, but was mainly driven from management's view of cutting costs. The result was often that the models and workflow simulation tools that were used were geared toward showing high-level tasks and information flow, and the objective function of the simulation effort was to cut down on processing time and simplify information flows to cut down time as it was modeled.

Clancey et al. (1988) have shown how business process-flow modeling approaches from the 1990s lacked the capabilities of modeling people's work. Together, Clancey and Sierhuis have been working on an agent-based modeling approach for modeling people's work at the practice level – how work really gets done. This work practice modeling approach is the topic of the next section and the main topic of this chapter. However, before we explain why it is that agent-based M&S is a powerful method for work system design and engineering, we turn briefly to the fact that management often does not like M&S of work systems, regardless of the fact that, if done right, it allows for an "objective" and "quantifiable" view of the work system's improvement.

17.3.2

The Difficulty of Convincing Management

It seems contradictory to the modeler and engineer that management would be uninterested in getting objective and quantifiable measures about their organization in order to improve the work product. However, we purposefully used quotations marks around the words objective and quantifiable at the end of the previous section.

First, it has to be understood that the modeler can never be completely objective. It is important to recognize that a model of a work system (or any system for that matter) is built from a particular point of view, namely, that of the modeler. It is extremely important for any modeling effort to have a well-defined objective. This objective needs to be developed with the participation of management and workers from the work system that is to be modeled. Management will and should drive the when, what, and why of the modeling effort. Without management (or customers, i. e., those who pay for the effort) approving and standing behind the when, what, and why of a M&S effort, the project will not succeed. Besides the so-called buy-in from management, it is as important to get buy-in from the workers whose work is going to be modeled, because they will be the ones that benefit or not from the outcome of the effort. Without their help the accuracy of the model will be compromised and the garbage-in-garbage-out phenomenon will creep into the effort.

The second important aspect of getting buy-in is to convince management that the simulation can quantify important variables that are part of the why of the effort. Often the whys of a M&S effort of a work system is about increasing efficiency, which most often will translate into reducing cost. However, this is not always the case. The example we use in this paper is based on an actual NASA project in which the why of the modeling effort, as put forward by management (the customer), was indeed related to efficiency, however not to reduce cost (because no

jobs were lost), but to increase the efficiency of the work system by introduction of a designed agent-based workflow system to reduce manpower for one particular organizational task. The overall objective was to add this manpower to other, understaffed tasks.

The key to convincing management of the benefits of M&S for WSD is to (a) use it as a decision-making method for management and (b) use it as a design and engineering method to provide an implementation of the decision that is supported in (a). As mentioned earlier, the quality of the model of the work systems depends on the amount of participation one gets from the workers in the organization that is being modeled. This participation goes two ways; (c) without the participation of the workers the quality and outcome of the simulation should always be questioned, and (d) without the participation of the workers the implementation of the resulting design will not be accepted by those who have to live with it.

17.4

Work Practice Modeling and Simulation

Work practice M&S is a model-based method for developing a time-based representation of the practices of people. Whereas work practice modeling is a process of developing static representations of a work system at the practice level, simulation adds a time dimension to such static models allowing us to view the model at different moments in time, and thus providing a way to see how the work happens in practice. In this section we describe what is meant by work practice. The modeler in Figure 17.1 develops a model of the work using the representational power of the Brahms language (Section 17.5). Model creation is an elaborate process of data collection and work description that leads to a static model of the situated activities of the individuals involved. Using the Brahms simulator, the model is simulated

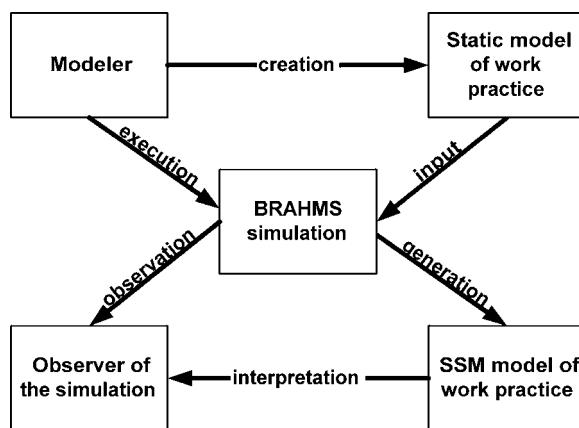


Fig. 17.1 Relation of a work model to a description of the work practice.

and a dynamic behavioral model of the work (i. e., a situation-specific model of the practice) is generated. The observer of the simulation model can observe the model during and after the simulation, interpreting the work practice model.

17.4.1

Practice vs. Process

Many researchers in the social sciences use the word *practice* as if it is a well-defined concept that everyone knows, described as “work as experienced by those who engage in it” (Button, 1996). Practice is also called “lived work” – “what work consists of as it is lived as part of organizational life by those who do it” (Button, 1996). In short, practice is doing in action (Suchman, 1987). However, it is difficult to describe what a practice is. People notice when something is not a practice, and can often describe why – “we normally don’t do things this way.” It can be said that a group of people has developed a practice, but when asked to describe what it consists of, we find it difficult to describe in words. As such, practice is part of our tacit knowledge (Polanyi, 1983). An ad hoc definition of *practice* that we use is:

The collective performance of contextually situated activities of a group of people who coordinate, cooperate, and collaborate while performing these activities synchronously or asynchronously, making use of knowledge previously gained through experiences in performing similar activities.

Practice is to be contrasted with formal process specification of what work is to be done. In the workplace itself, processes are often idealized and constitute shared values. Narratives that people record or present to authorities cater to these policies or preferences and create an inherent conflict in the work system between what people do and what they say they do. Two fundamental concepts related to the practice vs. process distinction are *behavior vs. function* and *activity vs. task*. Process models (e. g., workflow diagrams) are idealized functional representations of the tasks that people in certain roles are expected to do. In contrast, practice concerns chronological, located behaviors, in terms of everyday activities, for example, reading e-mail, meeting with a client, and sorting through papers. Activities are how people “chunk” their day, how they would naturally describe “what I am doing now” (Clancey, 2002, 2006).

17.4.2

Modeling Work Practice

In the model of technical rationality, the notion of a practice is automatically associated with the application of scientific knowledge in “major” professions (Schon, 1982). Not only are we claiming that practical knowledge is an important category of knowledge, but the concept of work practice allows us to view practical knowledge within the scope of all kinds of practitioners (not only within those of “major” professions). Here we focus on creating a framework that allows us to investigate, collect data about, and model the work practices of any group of individuals from

any type of profession. Even more, we focus the attention on work situations where multiple individuals from different professional backgrounds collaborate.

Work practice is constituted by the way people act and interact in their daily tasks as part of their job, socially and psychologically situated within their environment. It is situated action (Suchman, 1987) described in terms of activities and their context. It is how people act and interact in order to accomplish what they have to do. In the next sections, we give definitions of the important elements: community of practice, activity, communication, artifacts, and geographical environment, which are important elements in a theory on modeling work practice.

17.4.2.1 Community of Practice

People who are engaged in a work practice together belong to a community that has an identity (Wenger, 1999). Together this group of people is engaged in choreographed activities, acting either together or on their own. For example, consider the interplay of activities of people working and dining in a restaurant. There are different roles that are played, the waiters, the chef, the dishwashers, the maître d'homme, and so on. Even the dinner guests are part of the practice. They all engage in interplay, a kind of theatrical improvisation in real time, an unwritten play, so to speak, unrehearsed, but still they never forget their lines. They seem to know what the play is about, reacting to each other, never stepping out of character. They all seem to know their parts. They react to and communicate with each other. They have all played their parts before they have ever met each other, because their actions are based on similar previous experiences working and eating in restaurants. This is what the activity of working in a restaurant and going to eat in a restaurant is all about. It is a conceptual choreography. Everyone knows their roles, because they have done it so many times before. They are part of a community of practice that exists inside and outside the restaurant. This type of community of practice focuses on a group of people who produce something together.

a. Community of practice *A community of practice is a group of individuals, each with different individual skills and knowledge, performing complementary activities while producing something together that collectively can be seen as a unity within a practice.*

We define a second type of community of practice (b). The distinction between the first definition and the second is the type of people that belong to a community. The first definition (a) includes individuals playing different roles and performing different activities. The second definition includes people with similar skills and knowledge, playing the same role and performing similar activities. This type of community of practice includes the professional communities, such as the Java programmers at company X, the architects at company Y, or the group of waiters at a restaurant, and so on. However, it does not by definition have to be a professional community. For example, we could also talk about the practice of the group of people meeting each other regularly at the water cooler. Such communities are more informal or social and do not have to include people from the same professional

background. The point is that this definition of community of practice focuses on people that play similar roles and perform similar activities.

b. Community of practice *A community of practice is a group of individuals playing similar roles, each with similar skills and knowledge, that allow them to perform the same activities, that collectively can be seen as a unity within a practice.*

Both definitions are useful and hold true at the same time. The reason for making a distinction is for the purpose of identifying these types of communities of practice and the ability to talk about their practice as a whole. For the purpose of modeling, it is useful to make a distinction in the practice of a community in terms of different groups of people in an organization performing different activities, or in terms of a group of people performing similar activities. By describing a community of practice as a group to which individuals belong, we can represent people's practice in terms of the sum of the communities (groups) they belong to.

17.4.2.2 Activity

An important concept in modeling practice is that of an *activity*. In describing the practice of a group of people, we describe the individual and group activities over time as a *day in the life* (DITL) model of the group. To understand activities we must first understand that human action is inherently social. The key is that "action" is meant in the broad sense of an "activity", and not in the narrow sense of altering the state of the world (Engestrom, 2000). Describing human activities as social means that the tools and materials we use, and how we conceive of what we are doing, are socially and culturally constructed (Vygotsky, 1978; Leontev, 1978). Although an individual may be alone, as when reading a book, there is always some larger social activity in which he or she is engaged. For instance, the individual is reading the book, as relaxation, while on vacation. Engaging in the activity of "being on vacation", there is an even larger social activity that is being engaged in, namely, while on vacation still "working for the company" and while working also "being a parent", and so on. The point is that we are always engaged in a social activity, which is to say that our activity, as human beings, is always shaped, constrained, and given meaning by our ongoing interactions within a business, family, and community. An activity is therefore not just something we do, but a manner of interacting. Viewing activities as a form of engagement emphasizes that the conception of activity constitutes a means of coordinating action, a means of deciding what task to do next, what goal to pursue, in other words, a manner of being engaged with other people and things in the environment. The idea of activity has been appropriately characterized in cognitive science as intentional, a mode of being. The social perspective adds the emphasis of time, rhythm, place, and a well-defined beginning and end.

Conceptually, we can view activities as the "what we are doing at each moment in time." Goals can be viewed as the "why we are doing what we are doing," while tasks can be viewed as the "how we are doing what we are doing." In other words, goals and tasks are being executed within activities, or better, activities at the meso

level are our social conception of goals and tasks at the micro problem-solving level. Viewing work as the collective activities of individuals (Engestrom, 1999) allows us to understand why a person is working on a particular task at a particular time, why certain tools are being used or not, and why others are participating or not. This contextual perspective helps us explain the quality of a task-oriented performance.

We can be in more than one activity at the same time. This is situated action, an activity that is not fully planned in detail, and can be interrupted and resumed (Suchman, 1987); think about putting on your pants in the morning, and the phone rings. While there is not a control program that runs and controls our activities, a situation that suddenly comes up has to be dealt with, without articulated task knowledge. While switching context, the higher-level activity is still being engaged in. Therefore, it is such higher-level activities that constrain us from switching context from one lower-level activity to another lower-level activity and back.

People choose which activity they engage in, but they cannot choose this for others. Therefore, when people suddenly enter our space to interact, we juggle the activities we engage in. We suspend the current activity, start a new one, stop a third one never to come back to it again, and so on. We act in the situation and react to our environment. This is how the work practice of an organization is formed, and work happens or does not happen. If we are interrupted all the time during our work activities, we start acting a certain way, conscious or unconscious. We might hide, so that interruptions are minimized, or we might just do those activities that do not require a lot of time or can be interrupted at any moment. In short, the situation and the environment determine our activities, which in turn form our work practice.

Activity *An activity is a collection of actions performed by one individual, socially constructed, situated in the physical world, taking time, effort, and application of knowledge. An activity has a well-defined beginning and end, but can be interrupted.*

17.4.2.3 Communication

In order for two or more people to collaborate, they need to communicate. In Searle's speech act theory (Searle, 1969), the meaning and intent of *speech acts* are formalized in terms of sending and receiving communicative acts triggering response actions. A speech act has at least four distinct types of acts (Searle, 1969) that are all part of the same collaborative activity:

1. Uttering words is performing an *utterance act*. This is modeled with a time-consuming communication activity the sender is engaging in.
2. Referring and predication is performing a *propositional act*. This is modeled as the content of the message (beliefs of the sender) being communicated in the communication activity.
3. Stating, questioning, commanding, promising, and so on is performing an *illocutionary act*. This is modeled as the type of communication, defining the type of response expected from the receiver(s) of the communication.

4. The consequence or effect on actions, thoughts, and beliefs of the hearers is the *perlocutionary act*. This is modeled as the activity the receiver(s) engages in because of receiving the communication. Together with the utterance act of the sender this defines the collaborative activity both sender and receiver(s) engage in.

Searle went as far as defining a taxonomy of types of speech acts in which he classified all types as embodying one of five illocutionary points: assertives, directives, commissives, expressives, and declarations (Searle, 1975). Speech act theory analyzes communication in terms of its illocutionary point, force, and propositional content. Using this type of communication analysis we can model the sequence of communications in a collaboration activity between sender and receiver, as well as the intention and meaning of the speech act. In our theory for modeling work practice, sending and receiving information is minimally a coordination activity. However, in analyzing the way collaboration occurs in practice, we also need to analyze communication in terms of how it actually happens in the real world, thereby modeling collaboration as it really occurs. Speech act theory abstracts communication in terms of patterns of commitment entered into by the speaker and the hearer. While this is important, in modeling communication as it happens in practice we also need to take into account whether a communication activity between two people actually happens or does not happen. We need to include the communication tools used in the activity, because the type of tool has an impact on when and how the hearer receives the speech act. Today, communication is more and more efficient and certain communication tools are used globally. Phones, voice mail, e-mail, and fax are communication tools that are more and more taken for granted in the way that we use them. However, it should not be taken for granted that we all have created our own practice around the use of these tools in certain situations. We emphasize the point that collaboration is very much defined by our practice surrounding our communication tools, and that we, therefore, need to include the use of communication tools in modeling how people actually coordinate their collaboration in the real world. We need to include a model of the workings of communication tools (phone, e-mail, etc.) and how they are used in practice.

Communication *A communication is the activity of directionally transferring information (in the form of beliefs), held by one individual called the sender, to one or more individuals called the receivers, using a specific communication tool (face-to-face, telephone, e-mail, fax, document, etc.). After the transfer activity is complete, and successful, the receivers will hold the same information (beliefs) as the sender of the information and can now react to it.*

17.4.2.4 Context

Work is performed within a three-dimensional geographical environment. The restaurant we have dinner at, the office we work in, and the Moon crater Apollo astronauts explored are all examples of places and spaces that enable, while at the

same time constrain our work. The artifacts we use in our work, such as communication and information tools, are also located in a three-dimensional space. We are constrained to our three-dimensional world, and it defines very much how we can perform our work. For example, when the phone rings, we cannot hear it if we are not in the same room as the telephone. We also cannot observe specific changes in a location when we are not there. For example, if someone turns off the light in a room and you are not there, you will not observe this and therefore will not be aware of the fact that the light in this room is now off. To show the effect of the environment on practice, we have to include a model of the environment, including the people and artifacts in it, in a model of work practice.

Environment *The environment is a description of the physical environment and the people and artifacts located within it and performing activities.*

People use and/or create artifacts in almost all activities they engage in. When in the activity of hammering a nail, we use a hammer and a nail, and we end up with a nail in whatever artifact we have hammered it in. If we try to understand this activity in the context of performing it in the real world, we cannot leave out the artifacts. The artifacts constrain the way we perform activities. It is part of our context, and we have no choice but to interact with the physical world in order to act. We need to include these artifacts in our model of work practice. Leaving them out would miss the opportunity to understand the reason for performing activities. In other words, the artifacts are as important in the work practice as the people are.

Artifact *An artifact is a physical or digital information object in the world.*

Important in modeling work practice is how the artifact is used and conceptually understood within the activity. George Mead's social-behaviorist notion of *instances of the universal* (Mead, 1934), as well as Heidegger's notion of *breakdown* and *readiness-at-hand*, explain the role of objects – artifacts – in an activity. Mead, as well as Heidegger, uses the hammer and the activity of hammering as an example in which the hammer is the object that turns into a tool – as an extension of the hand. Mead's idea is that the concept "hammer" is the universal and the object used in the specific activity is the instance of the universal. Therefore, for Mead, the role of the hammer is *socially bound* to the activity and is not a property of the object itself. If the person who is hammering uses a piece of wood to hammer in the nail, that piece of wood becomes the instance of the universal during its use in the activity, and thus plays the role of a hammer. In other words, the object is transformed into the tool used to hammer in the nail. Heidegger, in essence, says the same. Only he speaks to it through the understanding that objects and their properties are not inherent in the world, but arise only in an event of breakdown in which the object becomes present-at-hand. To the person hammering, the hammer as such does not exist. It is part of the readiness-at-hand, and it is taken for granted in the activity, without the user's identification as an object. It is only in the breakdown, for example, when the person cannot find the hammer when he wants to hammer in the nail, that the object is present for the user. Whichever notion speaks to you, the issue that is important in modeling work practice is how artifacts are used, created,

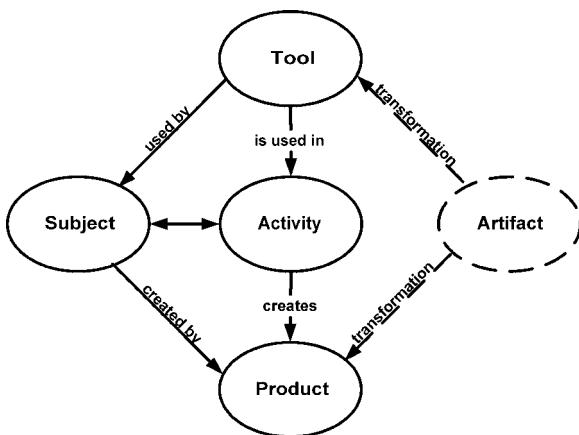


Fig. 17.2 Mediated relationship of artifacts in activities.

and conceptually understood by people within an activity. Figure 17.2 shows this relationship.

It is the use of the artifact in the activity – its role – that transforms the artifact into a *tool* or *resource*, or a *product* of the activity, used or created by the subject. Outside the activity the artifact is just an object in the world. To the observer, and to the modeler, the object is necessary for the activity to be performed.

Tool/Resource *When an artifact is being used in an activity, it becomes a tool or resource in the performance of the activity.*

Product *When an artifact is created or changed in an activity, it becomes a product of the activity.*

17.5

The Brahms Language

The name *Brahms* refers to the agent-oriented language (AOL), the multiagent M&S environment, and the MAS development and execution environment. The Brahms language was developed to incorporate and operationalize the theory of modeling work practice described in the previous section. A Brahms model or program can be written using a standard text editor or using one of the two available interactive development environments for the Brahms language; the Composer (Figure 17.11) or the Brahms Eclipse plugin. We refer the reader also to Chapter 12, Section 12.5, for a comparative description of Brahms with other agent simulation languages. Here we give some more details of the Brahms language and its syntax. For a more detailed description of the Brahms language we refer the reader to Sierhuis et al. (2009).

17.5.1

Simulation or Execution with Brahms

A Brahms model or program is executed by the Brahms virtual machine (BVM). All components of Brahms, including the compiler, the BVM, and both IDEs, are developed in Java, and thus run on most systems that run Java. The BVM can be run in one of three modes:

1. *Simulation mode*: In this mode the BVM is a multiagent discrete-event simulator. All agentd and behavioral objectd are independent processes (Java threads) that process events using their own belief-based inference and activity subsumption engine (Brooks, 1986). A Brahms agent is a *belief-desire-intention* (BDI) agent (Sierhuis, 2007). Each agent has a private set of beliefs that are used to infer new beliefs (in *thoughtframes* and *workframes*) and perform activities (in *workframes*). All creation of new beliefs and facts, as well as the start and end of activities, are new events for the agent that need to be scheduled on the agent's local event queue. There is a centralized scheduler that schedules the distribution of events to and from agents (in the case of agent-to-agent communication and world fact creation), based on a discrete simulation clock. Although the grain size of the simulation clock can be set by the user, by default one clock tick is assumed to be 1 s of simulated time. The centralized scheduler makes sure that all events generated by all agents are processed in the right order, making sure that each agent processes the correct events in each event cycle. See Figure 12.6 in Section 12.5.2 for a depiction of the central event scheduler and its interaction with a Brahms agent and the central world state.
2. *Real-Time or RT mode*: In this mode the BVM's central scheduler is bypassed and the simulation clock is turned off. Each agent and behavioral object now executes in real time. Since they are all independent Java threads, they are all independently executed parallel processes. All incoming events are processed as fast as the CPU schedules the priority of each agent Java thread in the Java VM. Since there is no centralized event scheduler and no simulation clock, activity execution time is managed by the system clock.
3. *Distributed RT mode*: This mode is equal to the RT mode, with the additional capability of allowing multiple BVMs to connect to each other over a network. Agents and behavioral objects running in different BVMs can find each other using a distributed directory service. Communication between agents is managed using a hidden communication layer. Different communication protocols are supported: TCP/IP, UDP, SSL, SOAP, Corba, and so on. This allows a Brahms program to be distributed over a network. For a more detailed description of the DRT model we refer the reader to Sierhuis et al. (2009).

To explain the Brahms language concepts, in the next subsections we use the example of a student who gets hungry while studying and needs to go to an ATM to get cash to pay for lunch.

17.5.2

Modeling People and Organizations

Modeling of people and organizations is done with the language concepts *agent* and *group*. Imagine that all students, while they are studying, monitor how hungry they are. The level of “hungriness” is measured by a belief about a real-valued attribute. Let’s assume that all students will study until their hungriness level rises to 21. At that moment, the student determines that he or she needs cash to go to lunch. This will make the student stop the “study” activity.

The program source code in Table 17.1 shows the Brahms code for a model of the above description. First, we see the group *Student* declared. The group has two *attributes*, *howHungry* and *needCash*, the first one being of type *double* and the second of type *boolean*. Then, there is one *relation* called *hasCash* defined. This relation makes it possible for a student to have cash on him or her. Cash is represented by a class type *Cash* (see next subsection).

Next in Table 17.1 is the definition of every student’s *initial beliefs*. For each agent that is a member of the group *Student*, initial beliefs are inherited from the group and are created at initialization time and added to the agent’s local belief set. In the source code below agent *Alex_Agent* is a member of the group *Student*, and the belief (*Alex_Agent.needCash = false*) is created at initialization (i.e., sim time = 0). This means that at initialization time agent *Alex_Agent* will believe that it does not need any cash.

Next is the definition of the activity *study*. In Table 17.1 you can see that activity *study* is defined as a *primitive activity* with a *max duration* of 3000 clock ticks, or, by default, 5 min of simulated time. This means that when an agent executes this activity, the simulation engine makes sure that the activity takes 5 min. How long the activity actually takes during a simulation is undefined and is dependent on the priority of other active activities (see Chapter 12.5 for a more detailed discussion on activity scheduling and execution). Important to note is that a primitive activity only takes a certain amount of time. What happens while the agent is in a primitive activity is not further specified. In this case, the agent is simply studying for 5 min at a time.

The last, but most important, piece of code in Table 17.1 is the definition of the *workframe* (WFR). WFR *wf_study* defines when the agent starts and stops executing the *study* activity. The *when-clause* defines the *preconditions* for the start of the WFR. In the example in Table 17.1, the preconditions state that the agent must believe that the time of the Campanile clock is less than 20, that it does not need any cash, and that the hungriness level is below 21. Only then will the WFR *wf_study* be executed and the *body* or *do-part* of the WFR be performed. Here only the *study* activity will be performed. After the activity is completed, the WFR is done. However, since the WFR *repeat variable* equals *true*, as long as the agent’s beliefs match the preconditions the agent will keep executing an instance of the WFR. Thus, the agent will repeatedly keep studying for 5 min, until at any moment in time the agent *detects* the *fact* that it needs cash. The *dt_veryHungry detectable* is active while the WFR is active, and if at any moment the agent detects the fact that it needs cash or

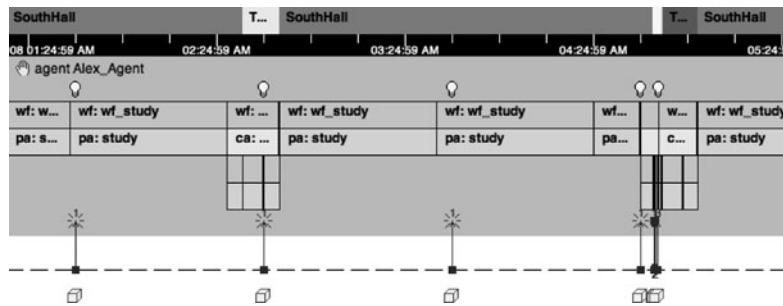


Fig. 17.3 AgentViewer screenshot of agent Alex_Agent execution.

believes that it needs cash through some other means (e.g., reasoning or communication with another agent), it will *abort* the current *study* activity. Because one of the preconditions is then also false – the agent now believes it needs cash – no new *wf_study* will get executed. This is shown in Figure 17.3, where one can see other WFRs interrupting the WFR *wf_study*.

The last line in Table 17.1 defines agent *Alex_Agent* as a member of the group *Student*, which means it inherits all concepts defined in that group. It should be noted that an agent can be a member of any number of groups, as well as that groups can be members of parent groups. In this way, we can model any organization by modeling the organizational, functional, and social groups and what people do when they are a member of these groups. By using group inheritance, the practice in an organization can be modeled by modeling the community of practice as groups with attributes, beliefs, activities, workframes, and thoughtframes. People that belong to a community of practice are modeled as agents that are members of the corresponding group.

Figure 17.3 shows the execution of multiple *study* activities by agent *Alex_Agent*.

17.5.3

Modeling Artifacts and Data Objects

As described in Section 17.4.2.4, a model of work practice includes the artifacts and data objects used in the context of an activity. In the student model described in Table 17.1, we can see a number of artifacts being used: the object *Campanile_Clock*, which is the representation of the Campanile bell tower on the University of California at Berkeley campus, and the *Cash* object of the student, representing the money the student has in his or her pocket. In the complete model there are other objects as well, such as the ATMs of the bank, the student's ATM card as artifact, as well as the student's bank account as a data object with a PIN code, and so on. Table 17.2 shows the source code of the classes for the ATMs and bank account, as well as some of the object instances. It shows that the Brahms language includes the concepts of *classes* and *objects*, besides those of agents and groups. Brahms is thus not only agent-based, but also object-oriented.

Table 17.1 Partial source code for Student group and agents.

```

group Student {
    attributes:
        public double howHungry;
        public boolean needCash;

    relations:
        public Cash hasCash;

    initial_beliefs:
        (current.needCash = false);

    activities:
        primitive_activity study() {
            //equals 5 mins of simulated time
            max_duration: 3000;
       }//end study

    workframes:
        workframe wf_study {
            repeat: true;
            detectables:
                detectable dt_veryHungry {
                    when(whenever)
                        detect((current.needCash = true), dc:100)
                        then abort;
                }//end dt_veryHungry
                when (knownval(Campanile_Clock.time < 20) and
                    knownval(current.needCash = false) and
                    knownval(current.howHungry < 21))
            do {
                study();
            }//end do
        }//end wf_study
    }//end Student

    agent Alex_Agent memberof Student { }

```

Important to note are the *initial_facts* defined for the *Alex_Account* object in Table 17.2. This object is a data object and the initial facts are created at object creation time. These represent the facts in the world that exist about student Alex's bank account. Agents can have beliefs about those attributes that differ from the facts in the world. This is how agent *Alex_Agent* can have a wrong belief about his bank account's PIN code, and can lose his ATM card after typing in a wrong PIN code three times at the ATM.

In Brahms, the world facts are separate from the agent's beliefs about those facts. There is only one world fact set. Agents can detect these facts using *detectables*, such as the one shown in the WFR in Table 17.1.

Table 17.2 Partial source code for classes and objects.

```

class Atm {
    display: "Atm";
    cost: 0.0;
    resource: true;
    attributes:
        public int currentAccountPin;
        public boolean pinChecked;
        public boolean pinIsWrong;
        public boolean pinAsked;
        ...
    relations:
        public Bank ownedbyBank;
    ...
}//end Atm

object Boa_Atm instanceof Atm {
    location: Telegraph_Av_113;
    initial_facts:
        (current ownedbyBank Boa_Bank);
}//end Boa_Atm

class Account {
    display: "Account";
    cost: 0.0;
    resource: true;
    attributes:
        public double balance;
        public string typeof;
        public int code;
        public int pin;
    relations:
        public Bank openedWithBank;
}//end Account

object Alex_Account instanceof Account {
    display: "Alex_Account";
    initial_facts:
        (current.balance = 20.00);
        (current.typeof = checking);
        (current.code = 1212);
        (current.pin = 1111);
        (current openedWithBank Boa_Bank);
}//end Alex_Account

```

17.5.4

Modeling Communication

Communication in Brahms is modeled as speech act or communicative act activities (Searle, 1975; FIPA, 2002). The Brahms language includes default *communi-*

cation and *broadcast* activities. In a *communication* activity, the performing agent specifies to which agents or objects it is communicating. In a *broadcast* activity, the performing agent does not have to specify the receivers of the communication. In that case all agents in the same location (Section 17.5.5) as the performing agent will receive the communication. What is being communicated is the beliefs specified in the so-called *transfer definitions* defined in the activity. It should be noted that an agent needs to have the beliefs in its belief set for it to be able to communicate them to another agent or object. This is to represent that we, as people, cannot communicate what we don't know.

Table 17.3 shows the definition of communication activity *communicatePIN* and use of this activity in the WFR *wf_communicatePIN*, all part of the *composite activity* called *useATM*. In this composite activity we model how the student uses an ATM. Figure 17.3 shows the communication of student *Alex_Agent* with the object *Boa_Atm*, showing the communication as lines from the agent to the object on the right in the figure. For a more detailed description of communications using the Brahms FIPA-based Communicator library, we refer the reader to Sierhuis et al. (2009).

17.5.5

Modeling Location and Movement

In Brahms, agents and objects can be situated in a model of the physical world. The world is represented independent of the capability of agents. An *area definition* is used for defining a class of *area* instances, used for representing geographical locations. Area definitions are similar to classes in their use. Examples of area definitions are “Building” and “City”. An example of an area is “Berkeley”. Areas can be decomposed into subareas. For example, a building can be decomposed into one or more floors. A floor can be decomposed into offices. The decomposition can be modeled using the “part-of” relationship. A *path* connects two areas and represents a route that can be taken by an agent or object to travel from one area to another. The modeler may specify distance as the time it takes to move from area1 to area2 via the path. The BVM automatically generates location facts and beliefs for agents and objects moving from one area to another.

Agents and objects can be located in an *initial location* (i. e., an area). Agents and objects can move to and from areas using a *move* activity. For example, Figure 17.3 shows the movement of agent *Alex_Agent* from the areas *SouthHall* to the areas of the restaurant and bank branch on Telegraph Avenue, and back to *SouthHall*. When agents and/or objects come into a location, the BVM automatically creates a location fact (*agent.location = <current-area>*). Agents always know where they are and they notice other agents and objects. When agents come into a location, the BVM automatically gives the agent a belief about its new location (same as the location fact), and also gives the agent a location belief for all other agents and objects currently in that location. When an agent or object leaves a location, the location fact and beliefs are retracted from all agents that are in that location the moment the agent or object leaves. Agents and objects can carry (through the

Table 17.3 Partial source code for communication of beliefs.

```

group Student {
    ...
    activities:
    ...
        composite_activity useATM() {
            activities:
            ...
                communicate communicatePIN(Atm at3, Account bka) {
                    max_duration: 20;
                    with: at3;
                    about: send(bka.pin = unknown);
                    when: end;
                } //end communicatePIN
            ...
            workframes:
            ...
                workframe wf.communicatePIN {
                    repeat: true;
                    variables:
                        forone(Account) bka;
                        forone(BankCard) bkc3;
                        forone(Atm) at3;
                        forone(Bank) ba3;
                        forone(Building) bd3;
                    when(knownval(current hasBankCard bkc3) and
                        not(current contains bkc3) and
                        knownval(current hasAccount bka) and
                        knownval(current.chosenBank = ba3) and
                        knownval(at3 ownedbyBank ba3) and
                        knownval(current.pinCommunicated = false) and
                        knownval(current.location = at3.location) and
                        knownval(at3 contains bkc3))
                    do {
                        communicatePIN(at3, bka);
                        conclude((current.pinCommunicated = true),
                            bc:100, fc:0);
                    } //end do
                } //end wf.communicatePIN
            ...
        } //end communicatePIN
    ...
} //end useATM
...
} //end Student

```

containment relation) other agents and objects. Contained agents and objects are not “noticed” until they are put into the area by the containing agent or object.

The geography model is a conceptual model – it does not represent the geography as a graphical three-dimentional model – representing geography as a hierarchical

Table 17.4 Brahms geography model.

```

// Area definitions
areadef University extends BaseAreaDef { }
areadef UniversityHall extends Building { }
areadef BankBranch extends Building { }
areadef Restaurant extends Building { }

// ATM World
area AtmGeography instanceof World { }

// Berkeley
area Berkeley instanceof City partof AtmGeography { }

// inside Berkeley
area UCB instanceof University partof Berkeley { }
area SouthHall instanceof UniversityHall partof UCB { }
area Telegraph_Av_113 instanceof BankBranch partof Berkeley {
}
area Telegraph_Av_2405 instanceof Restaurant partof Berkeley {
}

// initial location
agent Alex_Agent memberof Student {
    location: SouthHall;
} //end Alex_Agent

```

model of areas, with attributes representing facts about these areas and agents and objects as inhabitants of these areas. Areas can have attributes and relations, and define initial facts. Facts about areas can represent the state of a location, for example, the temperature in an area. The BVM automatically generates facts about the “part-of” relationships in the geography. Agents can detect these facts and thus learn (i. e., acquire beliefs) about the areas in their environment.

The example geography model in Table 17.4, defines a simple geography for the University of California at Berkeley. This model defines the university building SouthHall, where student Alex is initially located. Furthermore, the model defines a bank branch and a restaurant in the city of Berkeley.

17.5.6

Java Integration

Brahms currently has two ways of interfacing with Java using the Brahms JAPI:

- *Java activities* are primitive activities written in Java. To write a Java activity, the modeler needs to define the Java activity in the Brahms model and implement the activity by writing the activity using the Brahms JAPI. To do this you need to create a Java class that extends from the *AbstractExternalActivity* abstract class in the JAPI. The *AbstractExternalActivity* is an interface for external activities implemented in Java, called by Brahms Java activities. The

external activity can perform any Java action. This abstract class provides access to parameters passed to Brahms Java activities and allows for adding bindings to unbound variables passed to Brahms Java activities through parameters. Most importantly, you need to define the *doActivity* method to execute the Java activity.

- *External agents* are Brahms agents written in the Java programming language. To write an external agent you will need to define the agent as an external agent in the Brahms model and then write the external agent in Java using the Brahms JAPI. To do this you need to create a Java class that extends from the *AbstractExternalAgent* abstract class in the JAPI. The *AbstractExternalAgent* is an interface for external agents implemented in Java, loaded into the virtual machine to participate in a Brahms simulation or real-time agent execution. The external agent can perform any Java action. This abstract implementation provides access to the concepts loaded in the virtual machine and the world state to allow for communications with these concepts and to allow for world state changes to be triggered by this agent.
- *Java objects* can be referenced using Java class types as Brahms attribute types. This allows referencing Java objects from within the Brahms language.

17.6

Systems Engineering: From Simulation to Implementation

As we gained experience with applying our theory of modeling work practice using the Brahms environment, we developed a worldview and a set of practices for the development of software systems. We collectively named this set of practices *human-centered computing*, in the sense that the golden rule is that people in a work system should always be *at the center* in the development of new technology. Over the years, we have developed a set of practices that we are now starting to formulate as a MAS software development methodology. At the center of this methodology is our work practice M&S approach outlined in this chapter. The development of software is rooted in the analysis M&S of the work system in which the new software system is to be introduced. Due to the nature of the Brahms environment, an agent-based simulation model can be relatively easily converted to a real-time MAS. This fact has made it possible for us to first design a MAS as an agent-based simulation in Brahms, and subsequently convert this model to a MAS that can be executed by the BVM in real time. Hence, we are speaking of a methodology that allows us to go *from simulation to implementation* of a MAS. In this section we describe this methodology in more detail.

A methodology consists of a number of elements that build on each other like a pyramid (Figure 17.4). Below we describe how each of these elements are filled in in our approach. As we apply our methodology to more software engineering projects, new experiences will surely make us revisit each of these elements and make changes to the approach. It is thus important to note that the description we

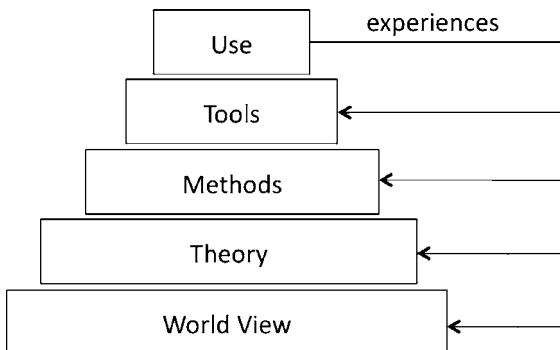


Fig. 17.4 The methodology pyramid.

provide here will not be the final version of our approach. One should see this as but the first attempt to formalize a set of practices we have developed over time and applied with success at NASA. In Section 17.7 we present our recent experience with applying our own methodology in a project for NASA's Mission Control. Here we first provide a more formal description of our methodology, as applied.

- *Worldview:* Our worldview is our human-centered slogan that people should always be at the center of any technology development project. This sounds trivial, but it is not the rule in most software development methodologies, or in any systems engineering approach for that matter. This human-centered slogan is the view on which our theory is based.
- *Theory:* The theory of our methodology is the combination of theories on which our theory is based, such as activity theory, situated cognition, and participatory design, and of course our theory of modeling and simulating work practice described in this chapter.
- *Methods:* The methods we apply are those that come from anthropology and knowledge engineering (e.g., participant observation, video and photography, interviews, and knowledge acquisition), as well as those coming from participatory design and cognitive psychology (e.g., story boarding and task analysis). Our main method is that of work practice M&S using the Brahms agent-based activity approach and the way we develop different models over the course of the project. We start with a model of the *current work system*, then move to a participatory design of the MAS in Brahms and incorporate this in a *model of the future work system*. The future work system model is then turned into an MAS.
- *Tools:* Besides the obvious tools of digital photo and video cameras, recorders, and so on, our main tool is the Brahms environment for developing agent-based models of the work practice and the MAS.
- *Use:* Our methodology has been used in different ways over the last 10 years at NASA. Every time we use the methodology on a project, we gain new understanding and experiences that we use to update our theories, methods, and tools.

17.6.1

A Cyclic Approach

The *from simulation to implementation* methodology uses a cyclic approach, as shown in Figure 17.5. The project starts with doing participatory observation of the current work system. This activity can take up to a couple of months, depending on the objective of the project. The next step is to model and computationally simulate the current work system in a *descriptive* Brahms agent-based model. Even though Figure 17.5 shows these two activities to be sequential, it should be understood that in reality these activities happen in parallel. It is the agent-based modeling of the work system that drives what parts of the work practice needs more observation.

After the simulation of the current work system is completed, the next activity is to design the work system change. This design activity needs to be participatory with the workers from the organizations in the work system. The MAS system is designed collaboratively given the constraints of the possible changes. The new design is then implemented in a Brahms MAS model and simulated. This MAS is incorporated in a simulation of the future work system that is changed from the current work system model. This way the impact of the newly designed MAS on the future work system can be analyzed. Also, changes to the current work system can be identified and analyzed, and the same metrics as those generated by the current simulation can be compared with those of the future simulation.

After the future work system simulation is completed, the next activity is to implement the MAS in a real-time environment. Using the Brahms environment, the Brahms model of the designed MAS can be easily changed to a real-time executable MAS. It is in this activity that the “from simulation to implementation” transition

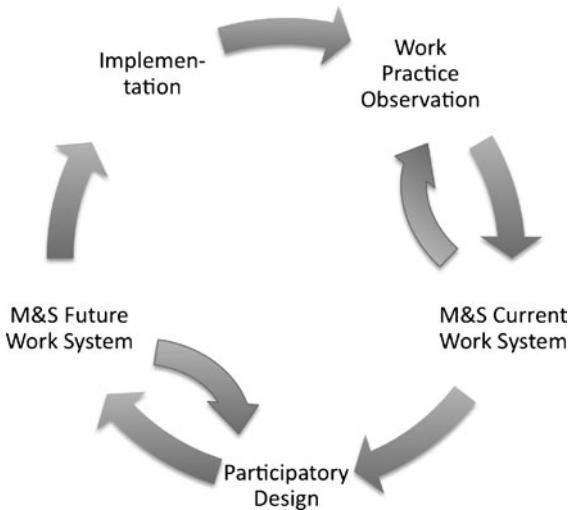


Fig. 17.5 Simulation to implementation cycle.

takes place. Ethnographers, system designers, and Brahms modelers are replaced by Brahms and Java programmers.

17.6.1.1 What to Include and When to Stop

The cycle is complete after implementation of the MAS in the work system. Continuous improvement to the work system is obtained by starting the next version of the cycle. The model of the future work system in the previous cycle now becomes the model of the current work system in the next cycle. However, because in the previous cycle the model was a *prescriptive model* of the future work system, new work practice observations and changes to the future model will result in a *descriptive model* of the current work system in the next cycle.

17.6.2

Modeling Current Operations

As with any M&S project, it is very important to define the objectives of the modeling effort up front. This should be done in collaboration with management and end users. The end product of M&S of the *current work system* is an agent-based model of a “day in the life” of the people in the work system. The output of the simulation should be a detailed activity timeline of all relevant roles and people in the organization, as well as the generation of metrics and statistics that answer the specific questions that are being asked.

17.6.2.1 Developing a Day-in-the-Life Model

A day-in-the-life (DITL) model is a model of the chronological activities of people. It is developed based on observation in the workplace where one can see what is happening minute to minute as the workday progresses. The trick is to observe without making immediate assumptions. The only form of interpretation the observer does is to abstract moment by moment action into high-level activities that can be identified to have a defined beginning and end, for example, leaving home to go to work (the “going to work” activity), coming to work and leaving work to go home (this is the highest-level “working” activity), the start and end of a meeting, the reading of e-mail, the answering of a telephone call, and so on.

Once the observer has identified an abstract model of the chronological activities of the day of a person that plays a particular role, the next step is to model these activities as an activity model for the agent representing that person. It is important to observe not just one person playing that role a number of different people playing the same role, as to develop not a person-specific model, but an activity model for that role in the organization. Once a role is modeled as one or more groups in Brahms, the same number of agents being members of these groups can be created in the simulation. Figure 17.6 shows the *observation-analyze-model* cycle of the development of a DITL model for the current operations of a work system.

Other important aspects to observe, analyze, and model are communications between people and how this communication takes place, be it fact-to-face, by phone,

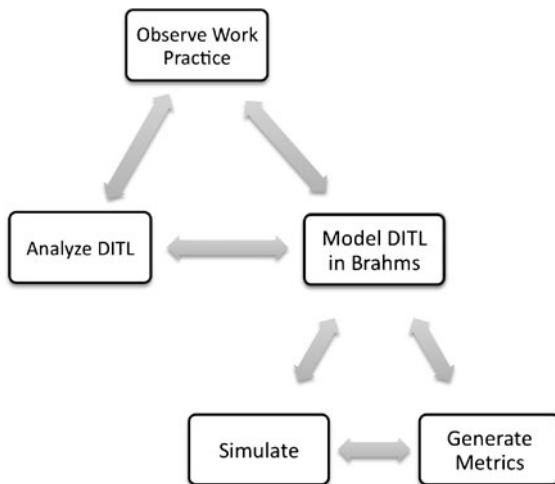


Fig. 17.6 Current operations modeling process.

e-mail, paper, or electronic documents, and so on. The use of space and artifacts in the workplace and the movement within the environment – for example, using a USB stick to transfer files from a computer on one's desk to another computer in another place. All these objects, information creation, and transfer need to be modeled in the DITL model, as does movement of agents within geography model.

17.6.2.2 Simulating a Day-in-the-Life Model

After a version of a DITL is created in Brahms, it is important to make sure the model can be simulated and results can be shown to the end user. Statistics and metrics generation need to be incorporated in the model from the beginning, such that the why and how questions can be answered with the simulation. What statistics to keep track of is an important decision in any effort.

The bottom of Figure 17.6 shows this *model-simulate-generate* cycle. The top and bottom cycles work in concert, and the development of a current operations model is based on cyclical phases of more and more detail in the DITL. The issue of when to stop going through this cycle is an important one. At the start of any M&S effort the objective of the effort needs to be clearly defined. The objective should be formulated in terms of what needs to be learned from the output of the simulation, for example, “What we are interested in is understanding how task X is performed by the workers in group Y in terms of who is doing what and when, what knowledge is used, and how long it takes. From this we need to design an automated system that can do this task.”

17.6.3

Modeling Future Operations

After the current operations modeling phase is finished, and the decision is made to change the current work system and/or create some automation that will be implemented in the current work system – thereby also changing the work system – the next phase in the simulation-to-implementation cycle is the M&S of the future operations.

Figure 17.7 depicts the future operations modeling process. We start with a participatory design task, where the new automation is designed in close participation with the end users of the system. It cannot be stressed enough how important the participatory approach is to this effort. Participatory design in this context means that the design team includes workers from the organization in which the system is going to be used. System and user interface requirements should be developed together.

When this is done, the design of the new system is implemented as a MAS in Brahms. This implementation is at first a simulation of the behavior of the system. The simulation of this future MAS system should use as input the same input as was used in the current operations simulation and generate the same output. This is accomplished by developing a DITL simulation of the future operations that includes a simulation of the MAS, based on the simulation of the current system. In particular, the people – end users – in the future work system need to be included as agents in the future DITL simulation model. Those aspects of the practice that need to change need to be modeled in this future model. Similar statistics as in the current operations simulation model will need to be generated by this future

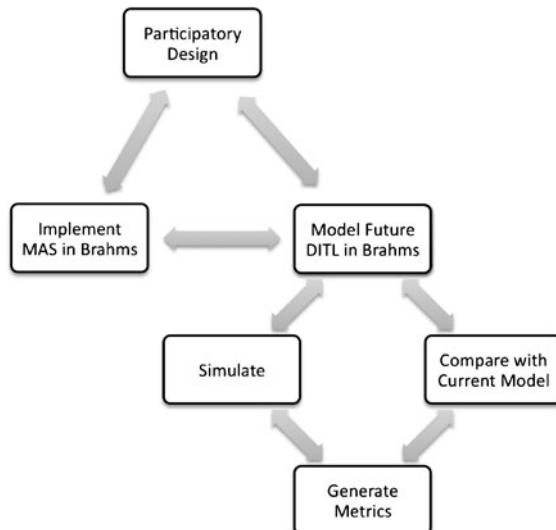


Fig. 17.7 Future operations modeling process.

operations simulation model. This will allow later on for a comparison between the current and future operations model.

The trick in this phase is to model the future MAS as close to complete as possible. This will not only help in the transition from simulation to implementation of MAS in the next phase, it also helps in prototyping the system in the participatory design process. Using the capability in the Brahms language to develop Java agents, designed user interface agents can already be implemented as part of the future simulation. This will allow for a close to realistic simulation. The simulation needs to be verified and validated in participatory fashion with the workers from the organization.

17.6.4

MAS Implementation

The last step in the from-simulation-to-implementation cycle is the transformation of the MAS as a simulation in Brahms into a distributed real-time MAS that implements the designed automation for the future work system. This transition process is depicted in Figure 17.8.

Depending on the needed interfaces of the system, this transition can be easy and fairly fast as long as the MAS simulation was done in detail. However, most of the time the MAS needs to be integrated with existing legacy systems and networks in the organization. The integration with existing external systems and networks will have been abstracted and simulated in the future operations model that includes a simulation of the MAS. The main effort of the transition from a MAS simulation to an actual executable MAS in the work system is the development of external system and network interfaces. For example, in the future simulation of the OCAMS system, described in the next section, the FTP-ing of files performed by an FTP agent was modeled as a simulation of the FTP of file objects on a simulated file system. In the MAS simulation model, the file system was modeled as Brahms areas where file objects were inhabitants of that area. The file FTP was simulated as a movement of file objects from one area to another. When the MAS system was implemented, this FTP agent became a Java agent that implemented the actual FTP transfer of files over the network system.

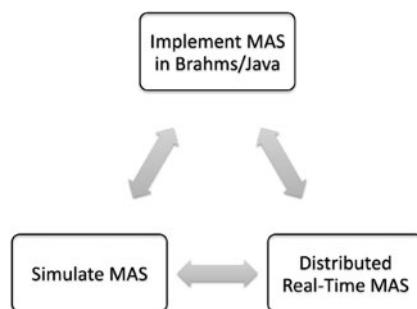


Fig. 17.8 Moving from simulation to implementation of the MAS.

When using appropriate agent communication protocols in the simulation, a simulated agent can easily be replaced by a real-time agent, making the transition from simulation to implementation rather seamless. When done correctly, the simulation of the MAS will have allowed for unit testing and some integration testing depending on the number of simulated agents, leaving a more complete integration testing and final system testing. Thus, the simulation of the MAS and the future operations simulation not only help in the design validation of the system and determining the impact on the current work system, it also cuts down in the development and testing of the actual MAS.

The Brahms environment makes this easy transition from simulation to implementation possible because of its ability to both run the BVM in simulation mode and in (distributed) real-time mode. In addition, the Brahms language is closely related to Java, and it is very easy to implement agents or activities with the existing Java API. As far as we know this is a unique feature that no other agent simulation language or agent-oriented language possesses.

17.7

A Case Study: The OCA Mirroring System

The Lyndon B. Johnson Space Center (JSC) near Houston, TX is the National Aeronautics and Space Administration's (NASA) center for human spaceflight activities. JSC houses the Shuttle and ISS Mission Control Centers (MCC), one of the most complex and best-known organizations for command and control of human spaceflight. The ISS MCC is charged with managing, commanding, and controlling every aspect of the ISS, from docking with the Space Shuttle and Soyuz spacecrafts to uplinking and downlinking all information to and from the ISS. The ISS MCC, as most MCCs, is divided into a “front room” – the room where the main flight controllers are located – and a “back room” – where the support flight controllers are located. Each flight controller in the front room has several support flight controllers in the back room. Together the people in the front and back room are organized into flight control groups for the different subsystems for the ISS, overall named the Flight Control Team (FCT).

Over a six-month period, computer scientists and ethnographers in the Work Systems Design & Evaluation Group of the Intelligent Systems Division at NASA Ames observed, studied, and simulated the work practices of the Orbital Communications Adapter (OCA) officer group to identify possible process improvements. Using statistics generated from a Brahms agent-based current operations model, the team, over the next 3-month period, designed and simulated a Brahms agent-based workflow system that now automates the process of creating a ground-based replica of the ISS file system (the mirror LAN). Future operations simulation statistics predicted a reduction in mirroring time from 6% to 0.4% of the OCA officer’s shift – a more than tenfold reduction. Using the above-described simulation-to-implementation methodology, agents were then converted, in a 1-month period, into a distributed MAS tool called OCAMS. Using three distributed Brahms BVMs,

these agents manage the workflow on multiple computers and servers using secure communications provided by the Brahms collaborative infrastructure (CI). The tool automatically writes large parts of the OCA handover log.

17.7.1

Mission Control as a Socio-Technical Work System

The main task of the MCC is to manage human spaceflight, from liftoff to landing at the end of a mission. The different groups of flight controllers manage all aspects of the mission. In the case of the ISS, this means monitoring and commanding all aspects of the ISS, as well as planning, scheduling, and managing the daily activities of the astronauts onboard the ISS. Besides the ISS MCC, the Shuttle MCC is housed by the same JSC organization (the Mission Operations Directorate) in the same building complex. This large organization is divided into divisions, branches, and groups with a total of around 3200 people. Most systems and subsystems of the ISS and Shuttle are managed by separate flight control groups. These groups consist of both front-room and back-room positions for a specific subsystem, such as “Guidance & Control”. Besides having general MCC software tools available that every flight controller group uses, each group is responsible for developing their own software tools to manage their subsystem and training each other to use these tools. Each group is thus a separate organization of flight controllers, trainers, and software developers. Most senior flight controllers do it all; they help train their younger colleagues, set requirements for tools and tool improvements, and “sit on console” – are in the flight control room working a shift as a flight controller.

There is a big difference between the Shuttle MCC and the ISS MCC. Whereas the Shuttle only flies a couple of times a year for a mission of around 14 d, the ISS is manned 24/7, 12 months a year. To do this there are three shifts – called orbits – in the ISS MCC. Certified flight controllers are the only ones that can sit on console, and most who are certified will be on console 1 week for a particular shift. Then they are “off console” for a week, which means that they are in their offices working on their next certification level (i. e., training) or are working on developing their next-generation tools and procedures.

While on console, the flight controller is solely responsible for his or her the for which he or she is certified. Depending on the stage of a mission – an ISS mission, called an expedition, takes about 6 months – the work on console can be highly stressful or very quiet. Besides the upfront known stage of a mission, a day in the life of a flight controller can never be predicted with certainty, because at any moment in time something can happen that requires their utmost attention, flexibility, and inventiveness. It is this excitement that most flight controllers like. Because of this high-stress environment, most flight controllers are young engineers.

The MCC is both a work system that manages and controls two of the most complex systems in the world (both the ISS and the Shuttle) and a work system that deals with both the most advanced software and computer systems (dedicated AI tools running on Linux systems) and the most old-fashioned software and computer systems (old mail servers running on Windows 2000 machines with 512 KB

memory). Indeed, the MCC is a hodgepodge of software and computer systems. Flight controllers need to be able to live within this sociotechnical work system, which often means that the young flight controllers are very inventive and flexible, seemingly working miracles with the tools and system that exist within their work environment.

17.7.2

The OCA Officer's Work System

The Orbital Communications Adapter (OCA) officer is a back-room flight controller and a member of the Operations Planner (OPSPLAN) group. The ISS OCA officer is responsible for manually uplinking and downlinking all files to and from the ISS. These files include schedules, procedures, commands, e-mail, photographs, health data, newspapers, and so on.

17.7.3

Simulating the Current OCA Work System

As part of the current OCA work process, the OCA officer spends time after each file uplink/downlink activity mirroring the same files to the mirror LAN. This cumbersome activity is both error prone, because of the manual “sneaker net” being used, and time consuming, because many uplink/downlink activities have to be duplicated on the mirror LAN using a laborious manual process. Figure 17.9 is a process flow depiction of the current mirroring process.

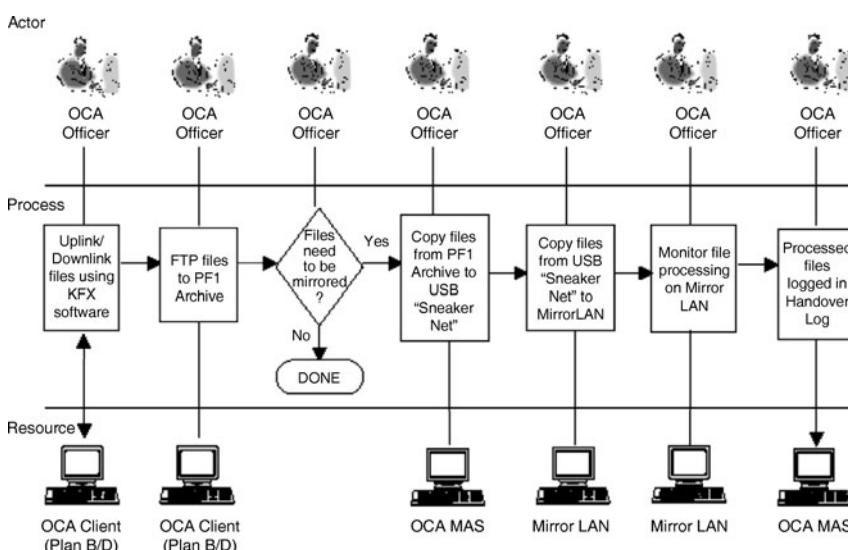


Fig. 17.9 Current OCA mirroring work process.

17.7.3.1 The Mirroring Process

After the OCA officer has uplinked or downlinked files to or from the ISS, he or she needs to create a mirror of the files on what is called the mirror LAN. The mirror LAN network is a ground-based duplicate of the network onboard the ISS. This mirroring activity starts with the OCA officer – after the files have been FTP-ed from the machine on which the files are downlinked or uplinked to the archive server – deciding which of the files need to be mirrored. Not every file uplinked or downlinked needs to be mirrored to the mirror LAN. After FTP-ing the files from the uplink/downlnk computer, the OCA officer moves in his or her chair back to the MAS computer from which he/she copies the files from the archive server via his/her workstation to a USB stick. After the files are copied to the USB stick the OCA officer moves in his or her chair to a space station computer (an IBM laptop) connected to the mirror LAN. The OCA officer copies the files from the USB stick to folders on the mirror LAN. Some of the files are processed by special batch files that run on the mirror LAN. For these files the OCA officer monitors the batch process, which can take up to 15 min, to watch for possible errors that can occur. If there are errors, the error files need to be sent to the flight controller responsible. This is done via e-mail on the MAS PC. After all files have been mirrored, the OCA officer logs all actions and problems in a Word document called the handover log.

17.7.3.2 Simulating the Mirroring Process

The current OCA mirroring work process was modeled in Brahms based on our 3-month observation with two OCA work practice observers. We took pictures, videos, and written notes. Our modeling data were based on timing of OCA officer activities during actual work.

Figure 17.10 shows the UML class diagram generated from the Brahms model (the Brahms compiler can generate standard UML interface – umi – format from Brahms source code). The Brahms group inheritance in Figure 17.10 shows the organizational model of the MOD organization of the OCA officers group. You can see that the OCA officers group is part of the DO4 Flight Planning Branch, which in turn falls under the DO Operations Division.

Figure 17.10 also shows the activity-based groups that the OCA officer group is a member of. Activity groups are abstract groups the modeler creates to define the common activities in the model. You can see the three activity groups in Figure 17.10 that are all members of the highest-level Schedule group. The OCA Mirroring group is the group in which the mirroring activity is modeled. Figure 17.10 shows the activities and attributes inherited from each group.

Figure 17.11 shows the subactivities of the composite mirroring activity in the *Agent Model*. The OCA agents inherit this activity from the OCAMirroringGroup, also shown in the UML class diagram of Figure 17.10. This activity describes what the OCA does to mirror a file. On the right-hand side of Figure 17.12 one can see part of the *Geography Model*, while on the left-hand side is the *Object Model*. The Geography Model shows the ISS OCA room areas in the Planning MPSR room, where the OCA officer and computers are located. The OCA officer agents can

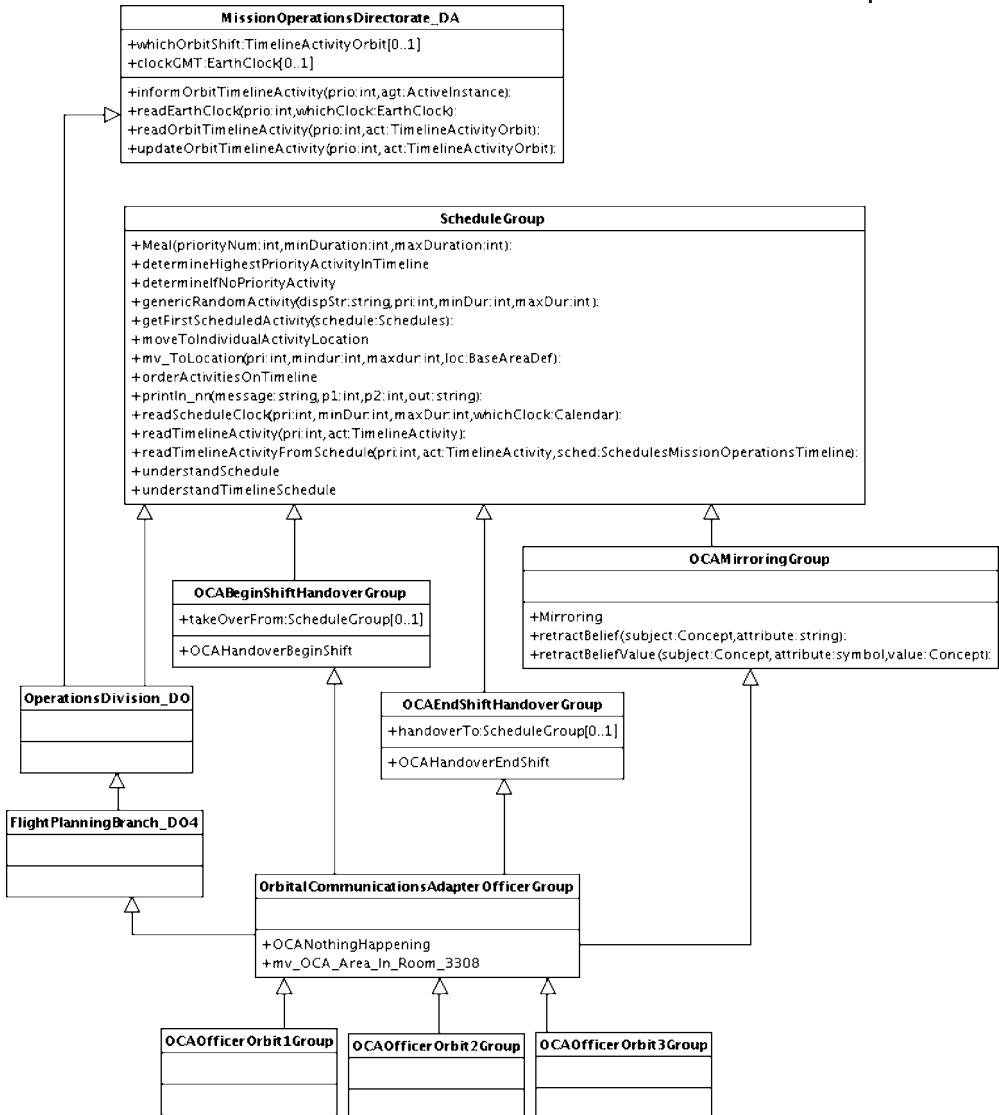


Fig. 17.10 Current operations sim statistics generated for each OCA officer on every shift over a simulated month of work (time in minutes).

move between room and room areas according to the paths specified. The Object Model on the left shows only a part of the object model. Together these three parts of the Brahms model constitute the OCA current operations model.

Figure 17.13 shows a part of the simulated OCA officer activities timeline for Orbit 3 (3rd shift). On the top right (1) you can see agent OCA Orbit 3 executing the

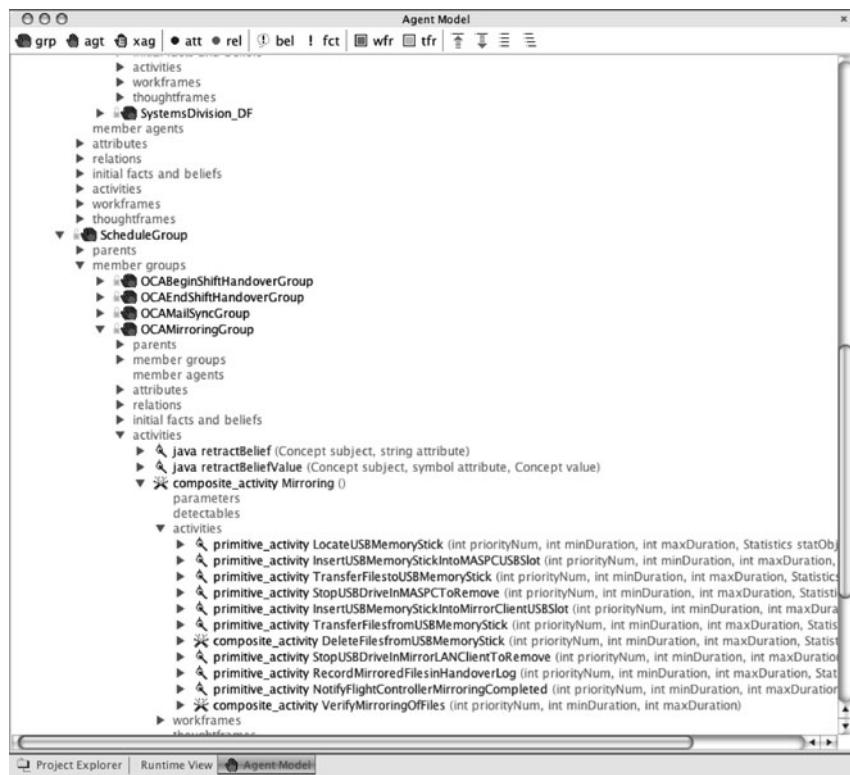


Fig. 17.11 Agent model of current OCA operations Brahms model.

mirroring activity with its subactivities. Below that (2), the timeline also shows the actual file object being moved (due to agent OCA Orbit 3's copying subactivities) from folder "V": to a folder on the mirror LAN via a USB "stick" (the colored bars above the black timeline show location movements of the OCA officer agent (1) and file object (2)).

In the logging activity the OCA officer writes the particulars of files uplinked, downlinked, and mirrored in a Word document (not shown in Figure 17.13). This document provides the log of what happened during the shift and is used for the next OCA shift to get a grasp of what happened in the previous shift. The verifying activity is where the OCA officer verifies the processing of specific types of files (e.g., the ISS astronaut activity timeline updates files) by the mirror LAN. As part of the copying processes, the mirror LAN server executes and processes the "dropped" files. The OCA officers have to verify that these batch processes execute correctly and the files are "absorbed" without errors. In case of errors, the OCA officer needs to deliver the error files to the responsible flight controller in the MCC. Of course, all this needs to be logged in the handover log.

Figure 17.14 shows, as part of the output of the OCA current operations simulation, the amount of time the OCA Officer spends on mirroring files in a month.

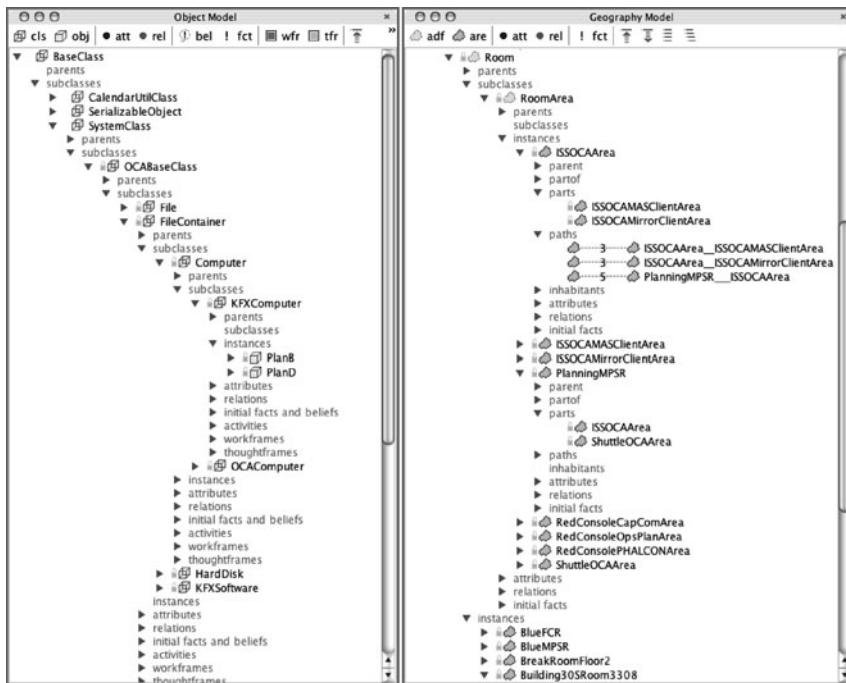


Fig. 17.12 Object and geography model of current OCA operations Brahms model.

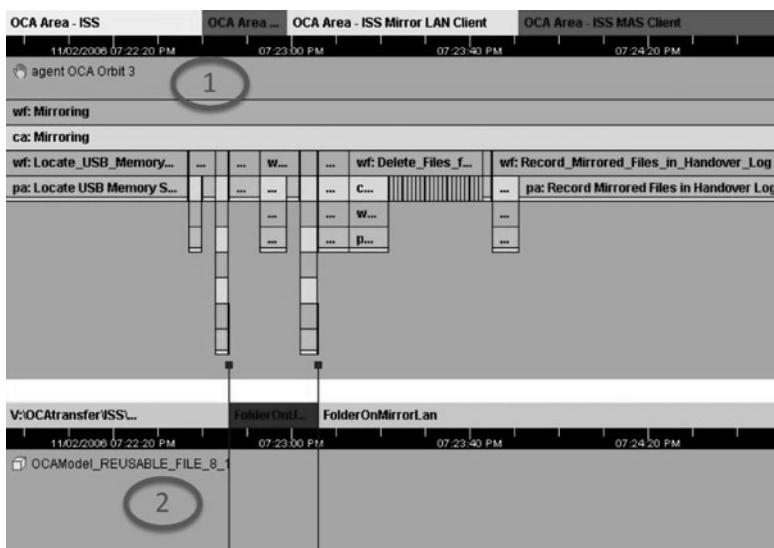


Fig. 17.13 Simulation timeline of OCA officer's mirroring activity during 3rd shift (Orbit 3).

Shift #	Communicating	Configuring Resource	Deleting	Logging	Moving	Searching	Transferring Files	Verifying	Total Time	% of Month
OCA Orbit 1	7	33	13	346	36	62	43	209	751	4.64%
OCA Orbit 2	10	45	19	412	51	91	56	272	958	6.49%
OCA Orbit 3	7	35	15	449	39	68	99	320	1035	7.02%

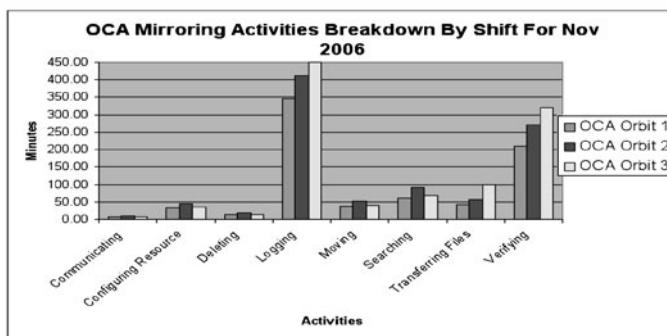


Fig. 17.14 Current operations sim statistics generated for each OCA Officer on every shift over a simulated month of work (time in minutes).

The table above the bar-graph shows the percent time each shift (orbit) spends on mirroring files of the total shift time. The actual numbers differ slightly each month, depending on how many files are being uplinked and downlinked. The statistics generated in Figure 17.14 are from actual uplink and downlink data from November 2006. Striking in these results is the fact that the highest time-cost for the OCA Officer are the (handover) logging and mirroring verification (verifying) subactivities.

17.7.4 Designing the Future OCA Work System

After the Current OCA simulation phase, a participatory design phase with a design team from the OCA Flight Officers group was started. The Future OCA Operations Work Process from Figure 17.14 was designed as a MAS that includes both the OCA Officer and the OCAMS system. All of the Mirroring activity that in the current work process is done by the OCA Officer is now replaced by the OCA Personal and OCA Mirroring agents. Part of the design was the design of a graphical user interface (GUI) through which the OCA Officer interacts with the OCA Personal agent.

The OCAMS agents perform all the mirroring and logging activities. The OCA Officer agent only has to select uplinked and downlinked files in the GUI, and review the OCAMS activity when complete. The GUI agent displays the mirroring status back to the OCA Officer who verifies the mirroring by OCAMS. All OCAMS activity is also reported in the handover log, which the OCA Officer uses to verify mirroring was completed correctly.

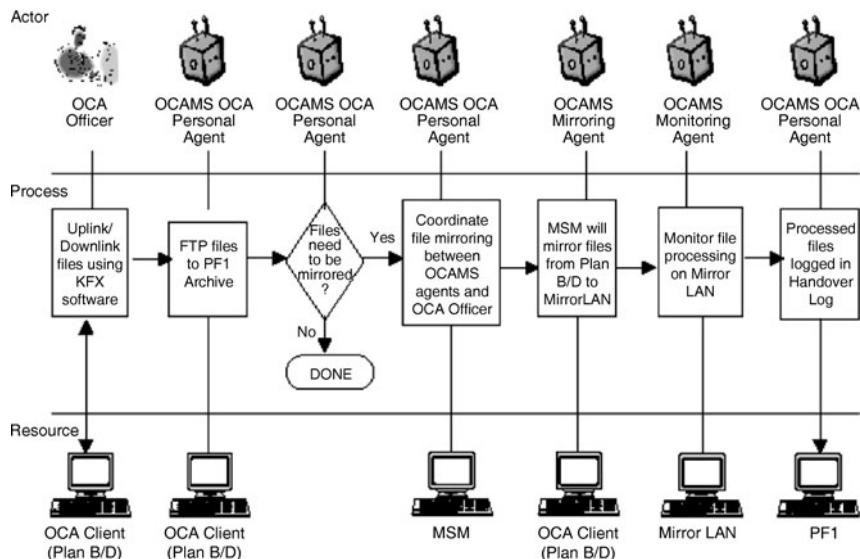


Fig. 17.15 Future OCA mirroring work process.

17.7.5

Simulating the Future OCA Work System

The future operations simulation model was then implemented as a Brahms MAS simulation, including a simplified work practice model for the OCA officer agent and the GUI agent (Figure 17.16).

This *Future OCAMS MAS simulation model* was simulated with Brahms using the same November 2006 data. This allowed us to generate the same statistics as were generated in the current OCA model. Comparing Figure 17.17 with Figure 17.14 there is more than tenfold decrease in mirroring time for the OCA officer. In particular, the logging and verifying activities decreased significantly, giving the OCA officer time to work on other tasks. Based on these results, the decision was made to develop and implement the OCAMS MAS.

17.7.6

Implementing OCAMS

In the last phase we moved *from simulation to implementation* of the OCAMS MAS. This step was accomplished within 1 month with three developers. Architecturally, the OCAMS system is divided into three separate distributed agent systems, each of which is running in a separate Brahms Hosting Environment (BHE) (Figure 17.18). These BHEs can run on any desired computer and network configuration, making the architecture easily adaptable to the computer architecture and network safety concerns of the ISS MCC.

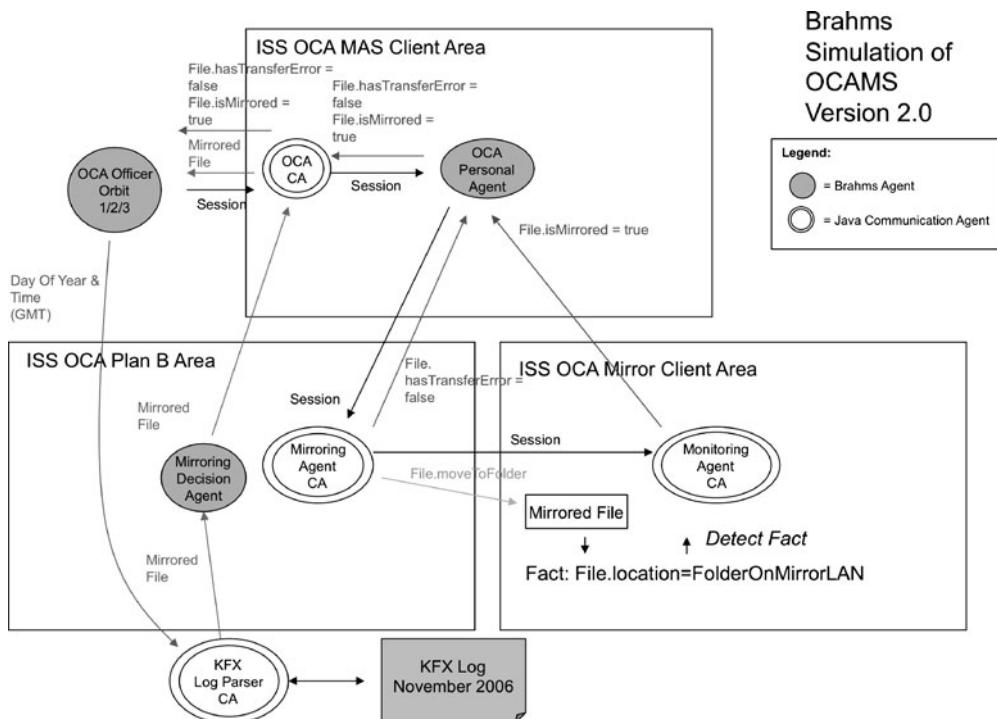


Fig. 17.16 Future OCAMS MAS simulation model.

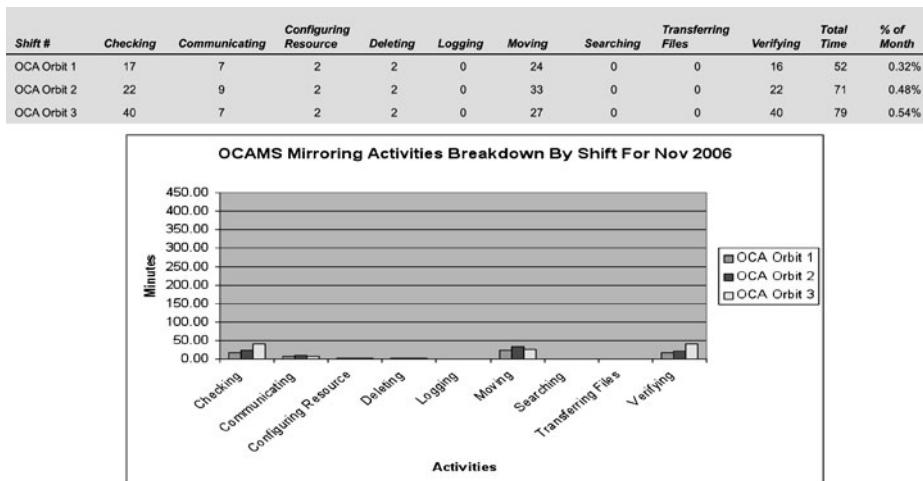


Fig. 17.17 Future operations with OCAMS: sim statistics generated for each OCA officer on every shift over a simulated month of work (time in minutes).

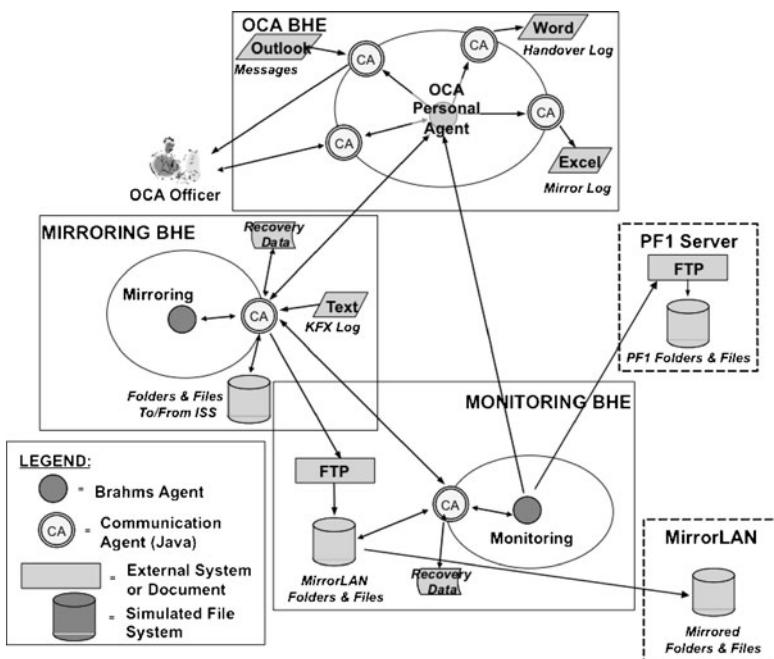


Fig. 17.18 OCAMS MAS architecture

The OCAMS Release 1 MAS consists of nine individual software agents (Figure 17.18). Three agents are rule-based BDI agents written in the Brahms language, while six are Java-based agents written using the Brahms Java API. These Java agents are referred to as a “communication agent”, because their purpose is to communicate with external systems or files outside of the OCAMS system.

The agents are currently divided over three agent subsystems (BHEs) but could easily be loaded in a different number of BHEs. The OCA BHE consists of the central OCA Personal Agent that coordinates all work with the other agents in the system, four human interface agents, one GUI agent and three Microsoft Office agents. The mirroring BHE consists of the agents that interface with the KFX file uplink and downlink software currently used by the flight controllers and decides which file to mirror on the mirror LAN. The monitoring BHE is the workhorse of OCAMS. It has agents that monitor FTP and copy files to the mirror LAN and monitor for errors that might occur in the FTP and copying between file systems on the network, as well as problems with monitoring the specific file processing that needs to occur on the mirror LAN depending on the mirrored file type. Brahms agents (the solid circles in Figure 17.18) are used where belief- and rule-based decision making is needed. From our experience in developing BDI-based MASs, we developed some basic rules of thumb for deciding whether an agent should be a BDI agent (i.e., written in the Brahms language) or an imperative agent (i.e., written in the Java language).

17.8

Conclusion

In this chapter we presented an agent-based engineering methodology for analyzing and designing work systems. The methodology is based on a theory for modeling and simulating work practice. This theory is founded in social science and informatics research mainly in Europe, as well as psychological theories of activities and situated action. The theory is operationalized in the Brahms multiagent environment. Brahms has been developed with the idea that an agent-based simulation based on BDI and Java agents can easily be transformed into a real-time MAS. This approach has led to the creation of our *human-centered work systems design* methodology, in which we use work practice observations and participatory design to develop a simulation of an organization's work practice and model the designed software system as a MAS in Brahms, first as a simulation that later is transformed into a distributed MAS.

This methodology has been developed over almost 10 years of research at NASA Ames Research Center. The approach has been successfully applied in NASA's Mission Control Center in Houston, TX. The OCAMS system, described as a case study in this chapter, has been in operation for more than 6 months at the time of this writing. Currently, Release 2 of OCAMS has been delivered to our customer. In addition to automating the mirroring activity, Release 2 automates the archiving activity as well. There are in total five releases planned for OCAMS in the next 2 years.

Not only is the OCAMS MAS the first MAS deployed in the ISS MCC, the system was delivered within budget, on time, and without any serious problems the moment it was turned on. The system enjoys 100% usage by all OCA officers. The success of OCAMS and the ease with which it was deployed within Mission Control is due to the *from simulation to implementation* methodology.

Acknowledgements

This work was supported by the National Aeronautics and Space Administration, in particular by the Mission Operations Directorate (MOD) of NASA Johnson Space Center and the Intelligent Sciences Division at NASA Ames Research Center. We would like to thank the OCA team at NASA JSC for their support and acceptance of our approach and of the OCAMS system. Brahms has been funded by NASA, the former NYNEX Corporation (now Verizon), and the former Institute of Research on Learning. Last but not least, we thank all those who have been part of the Brahms development team over the last 16 years.

References

- Beyer, H. and Holtzblatt, K. (1998) *Contextual Design: Defining Customer-Centered Systems*, Morgan Kaufmann Publishers, San Francisco, CA.
- Brooks, R. (1986) A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2(1), 14–23.
- Buchanan, B.G. and Shortliffe, E.H. (1984) *Rule-based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project*, Addison-Wesley.
- Button, G. and Harper, R. (1996) The relevance of 'work-practice' for design. *Computer Supported Cooperative Work*, 1996(4), 263–280.
- Carroll, J.M. (2003) *HCI Models, Theories, and Frameworks: Toward a Multidisciplinary Science*, Morgan Kaufmann.
- Checkland, P. and Scholes, J. (1990) *Soft Systems Methodology in Action*, John Wiley & Sons Ltd., Chichester, England.
- Clancey, W., Sachs, P., Sierhuis, M. and v. Hoof, R. (1998) Brahms: Simulating practice for work systems design. *International Journal on Human-Computer Studies*, 49, 831–865.
- Clancey, W.J. (2002) Simulating activities: Relating motives, deliberation, and attentive coordination. *Cognitive Systems Research*, 3(3), 471–499.
- Clancey, W.J. (2006) Observation of work practices in natural settings. *The Cambridge handbook of expertise and expert performance*, (eds K.A. Ericsson, N. Charness, P.J. Feltovich and R.R. Hoffman), Cambridge University Press, pp. 127–146.
- Davenport, T.H. (1993) *Process Innovation: Reengineering Work Through Information Technology*, Harvard Business Press.
- Davenport, T.H. (1995) The fad that forgot people. *Fast Company*, 1995(1).
- Ehn, P. (1989) *Work-Oriented Design of Computer Artifacts (2nd Edition)*, Lawrence Erlbaum Associates, Hillsdale, NJ.
- Emery, F.E. and Trist, E. (1960) Socio-technical systems. *Management Sciences, Models and Techniques*, (ed. C. Churchman), Pergamon, London.
- Engeström, Y. (1999) Expansive visibilization of work: An activity-theoretical perspective. *Computer Supported Cooperative Work*, 8, 63–93.
- Engeström, Y. (2000) Activity theory as a framework for analyzing and redesigning work. *Ergonomics*, 43(7), 960–974.
- FIPA (2002) Fipa communicative act library specification. FIPA (2002) Fipa communicative act library specification.
- Greenbaum, J. and Kyng, M. (eds) (1991) *Design at Work: Cooperative design of computer systems*, Lawrence Erlbaum, Hillsdale, NJ.
- Elle, P. (1989) *Work-Oriented Design of Computer Artifacts (2nd Edition)*, Lawrence Erlbaum Associates, Hillsdale, NJ.
- Kling, R. and Star, S.L. (1998) Human centered systems in the perspective of organizational and social informatics. *Computers and Society*, 28(1), 22–29.
- Lave, J. (1988) *Cognition in Practice*, Cambridge University Press, Cambridge, UK.
- Leont'ev, A.N. (1978) *Activity, Consciousness and Personality*, Prentice-Hall, Englewood Cliffs, NJ.
- Luck, M. and Padgham, L. (eds) (2008) *Agent-Oriented Software Engineering VIII: 8th International Workshop, AOSE 2007*, Springer, Honolulu, HI.
- Mead, G.H. (1934) *Mind, Self, and Society: From the standpoint of a social behaviorist*, University of Chicago Press, Chicago, IL.
- Nardi, B.A. (ed.) (1996) *Context and Consciousness: Activity Theory and Human-Computer Interaction*, The MIT Press, Cambridge, MA.
- Polanyi, M. (1983) *The Tacit Dimension*, Peter Smith, Magnolia, MA.
- Rasmussen, J., Pejtersen, A.M. and Goodstein, L.P. (1994) *Cognitive Systems Engineering*.
- Schön, D. (1982) *The Reflective Practitioner: How Professionals Think in Action*, Basic Books.
- Searle, R.J. (1969) *Speech Acts*, Cambridge University Press, Cambridge, UK.
- Searle, R.J. (1975) A taxonomy of illocutionary acts. *Language, Mind, and Knowledge*, vol. 1–29, (ed. K. Gunderson), University of Minnesota, Minneapolis, pp. 344–369.
- Senge, P.M. (2006) *The Fifth Discipline: The Art and Practice of the Learning Organization*, Random House, Inc.

- Sierhuis, M. (2007) "it's not just goals all the way down" – "it's activities all the way down". *Engineering Societies in the Agents World VII, 7th International, Workshop, ESAW 2006, Dublin, Ireland, September 6-8, 2006, Revised Selected and Invited Papers*, volume LNCS 4457/2007 of *Lecture Notes in Computer Science*, (eds G.M.P. O'Hare, A. Ricci, M.J. O'Grady and O. Dikenelli), Springer, Dublin, Ireland, pp. 1–24.
- Sierhuis, M., Clancey, W.J. and v. Hoof, R.J. Brahms: An agent-oriented language for work practice simulation and multi-agent systems development. *Multi-Agent Programming, 2nd Edition*, (eds R.H. Bordini, M. Dastani, J. Dix, A. El Fallah Seghrouchni), Springer, to Appear.
- Sterman, J.D. (2000) *Business dynamics: systems thinking and modeling for a complex world*, Irwin/McGraw-Hill.
- Suchman, L.A. (1987) *Plans and Situated Action: The Problem of Human Machine Communication*, Cambridge University Press, Cambridge, MA.
- Vicente, K.J. (1999) *Cognitive Work Analysis: Toward Safe, Productive, and Healthy Computer-based Work*, Lawrence Erlbaum Associates.
- Vygotsky, L.S. (1978) *Mind in Society: The Development of Higher Psychological Processes*, Harvard University Press, Cambridge, MA.
- Wenger, E. (1999) *Communities of Practice: Learning, meaning, and identity*. Learning in doing: Social, cognitive and computational perspectives, Cambridge University Press, Cambridge, MA.

Index

a

- adaptation 41
- adaptive experience management 41
- agent-based modeling 269
- agent-based simulation 112
- Agent Communication Language 274
- agent-directed simulation 112, 224
 - agent-based modeling 119
 - deliberative behavior 127
 - goal-directed behavior 125
 - interaction 126
 - strategic action 126
- agent-based simulation 112
- agent simulation 112, 118
- agent-supported simulation 112
- agents for simulation 112
- agent execution cycle 273
- agent-implemented experimental frames 363
- agent-implemented test infrastructure 380
- agent-oriented language 291
- agent simulation 118
- agent society 78, 100
- agent-supported simulation 112
- agent systems 111, 120
 - network-centric systems 173
- agent taxonomy 96
 - artificial agent 97
 - biological agent 97
 - computational agent 97
 - robotic agent 97
 - software agent 97
 - entertainment agent 97
 - information agent 97
 - task-specific agent 97
- agenthood 84, 86, 100
- agents 113
 - software agents 113
- agents for simulation 112

agents systems

- agent organizations 120
- AI-directed simulation 31
 - AI-based simulation 31
 - AI-supported simulation 31
- analysis 318–319
- analytic hierarchy process 185
 - multicriteria decision making 185
- anticipation 41
 - anticipatory system 41
- architectural frame 84
- artificial intelligence 20
- Ascape 271, 277
- autonomic computing 37
 - self-configuration 37
 - self-healing 37
 - self-optimization 37
- autonomic simulation system 47
- autonomic simulation systems 41
- autonomy 77, 245

b

- behavioral change 249
- belief-desire-intention 82
- Brahms 291, 475

c

- Capability Maturity Model 184
- CASESim 300
- cellular automata 436
- cellular manufacturing 457
- centralized synchronous approach 271
- cognitive networks 179
- cognitive system 176
- cognitive systems 399, 404, 406
- collaboration 434
- combined model ensemble 49
- command and control 402, 422

- Common Operational and Tactical Picture 179
 communication 175, 270, 434
 community of practice 482
 complexity 113, 219
 computational paradigms 115
 - goal-directed knowledge processing 116
 - agent-based paradigm 116
 - AI-based paradigm 116
 - event-based paradigm 116
 - intentional knowledge processing 115
 - procedural knowledge processing 115
 computer-aided modeling 11
 conflict resolution 434
 control 153
- d**
 decision making 258
 decision support 87
 decision support simulation systems 399, 401, 405, 411
 decision support systems 399–400, 404–405
 decomposition analysis and resolution 372
 deliberative agents 89
 Design 318–319
 DEVS 11
 DEVS model 363
 discrete-event models 436
 dynamic composability 135
- e**
 emergent organizational behavior 241
 evaluation environment 184
 - indicator hierarchy 185
 event scheduling 436
 evolutionary algorithm 342
 experimental frame 366
- f**
 facility layout 456
 false alarms 408–409, 411
 flexibility 82
 flexible manufacturing system 457
 functional analysis 153
 functional engineering simulator 324
 functional validation testbench 324
- g**
 GA 341, 343
 genetic algorithm controller 53
 Global Information Grid (GIG) 381
- h**
 holon 466
- holonic manufacturing system 466
 House Resolution 487 3
 human-automation 406
 human behavior 88
 human complex systems 336
 hybrid agent architectures 43
- i**
 imitation 7
 inference engine 176
 information assurance 179
 integrated development and testing
 - methodology 388
 integration 318
 intelligence, surveillance, and reconnaissance 179
 intelligent agents 111
 interoperation 135
 - broker agent 138–139
 - facilitator agent 137
 - mediator agent 137
 introspection layer 44
- j**
 joint mission thread 378
- k**
 knowledge base 176
 knowledge generation activity 12
 knowledge processing 9
- l**
 layered approach 271
 learning in agents 129
 life cycle 158
- m**
 M&S 3
 - decision support 4
 - education 4
 - experience 4
 - experimentation 6
 - understanding 4
 management 149
 manufacturing systems 452
 - cellular manufacturing 457
 - facility layout 456
 - flexible manufacturing system 457
 - holonic manufacturing system 466
 - holon 466
 - material handling 457
 mapping 413
 Mason 271, 289
 MASs 271, 275, 291, 305, 308

- material handling 457
- metamodel 412
- metasimulator 132
- model analysis 15
- model base 375
- model-based activity 11, 13
 - model-based management 13
 - model-building 13
 - model processing 13
 - behavior generation 15
 - model analysis 15
 - model transformation 15
- model-based management 11, 15
- model behavior 19
 - point behavior 19
 - structural behavior 19
 - trajectory behavior 19
- model-driven enterprise information systems 11
- model transformation 16
- modeling 3
- multiagent systems 241, 269
 - designed 241
 - emergent 241
- multicriteria decision making 185
- multimodel 132, 411, 415, 417
 - multi-aspect multimodel 132
 - multistage multimodel 132
- multisimulation 131–132, 411, 417, 423
 - metasimulator 132
 - model recommendation 133
- multistage models 417

- n**
- net-centric infrastructure 380
- NetLogo 271, 280–281
- network-centric system 177
 - Common Open Policy Service 177
 - Directory Enabled Networking 177
 - Internet Protocol Security Policy 177
 - Network Data Management Protocol 177
 - Simple Object Access Protocol 177

- o**
- open system architectures 180, 186
- open systems 174
 - autonomy 174
 - dynamism 175
 - heterogeneity 175
- operational view 180
- organization 238, 434
 - computational organization 238

- organizational change 248
- organizational structure 240
 - adhocracy 240
 - entrepreneurial 240
 - machine 240
 - professional 240
 - resource capacity 243
 - task complexity 243
 - volatility 243
- organizational utility 247
- organization theory 240
- organizational design 240
- organizational structure 237

- p**
- partial model ensemble 48
- particle swarm optimization 50
- perception 99, 401, 410, 412
- performance estimation 318–319
- Petri-net 436
- pruned entity structure 385

- q**
- quality 173
- quality assessment 184
- quality assurance 176

- r**
- reflection 81
- reliability 407, 424–425
- reorganization 238, 244
- RePast 271, 283
- requirements analysis 153, 318–319
- requirements management and verification 326
- resolution 412–413
- risk 165
- robustness 41

- s**
- scope 412–413
- self-organization 40, 50
- self-organizing system 40
- sense-making 80
- service-oriented architecture 306
- simulation 3
- simulation-based augmented reality 13
- simulation-based decision support 38
- simulation for agents 112
- situated environment 78, 100
- situatedness 77
- sociotechnical work system 504
- software agents 113
- software engineering 20, 184, 433

- software process simulation models 436
 - software processes 433
 - software systems engineering 433
 - software validation facility 324
 - spacecraft qualification and acceptance 325
 - specification 318
 - standards 153
 - EIA-STD-632 153
 - IEEE-1220 153
 - ISO 155
 - ISO-IEC-IEEE-STD-15288 153
 - ISO/EIC 15288 159
 - MIL-STD-499B 153
 - Swarm 271, 286
 - synthesis 153
 - SysML 326
 - system analysis 153
 - system architecture 180
 - open system architectures 180
 - operational view 180
 - system view 180
 - system communication 409
 - system complexity 408
 - system concept simulator 324
 - System dynamics 436
 - System Entity Structure 369
 - system sciences 11
 - complex systems 11
 - discrete-event systems 11
 - system theory 11
 - system view 180
 - systems engineering 20, 147, 149, 220, 317
 - agent-based systems engineering 148
 - detailed design 162
 - life cycle 158
 - preliminary design 161
 - process 157
 - requirements analysis 160
 - risk management 165
 - simulation system engineering 222
 - simulation systems engineering
 - agent-directed simulation systems 226
 - systems of systems 86, 329
- t**
- technical view 180
 - technology 149
 - test 318
 - trust 400, 406–407
- u**
- updating knowledge base 274
- v**
- validation 174, 318
 - Vee model 331
 - Vee Model of systems engineering 372
- w**
- work practice 434, 475
 - work system design 475
 - Brahms 475
 - cognitive work analysis 477
 - contextual design 476
 - sociotechnical work system 504
 - soft systems methodology 476
 - work practice 475
 - community of practice 482

WILEY SERIES IN SYSTEMS ENGINEERING AND MANAGEMENT

Andrew P. Sage, Editor

ANDREW P. SAGE and JAMES D. PALMER

Software Systems Engineering

WILLIAM B. ROUSE

**Design for Success: A Human-Centered Approach to Designing Successful Products
and Systems**

LEONARD ADELMAN

Evaluating Decision Support and Expert System Technology

ANDREW P. SAGE

Decision Support Systems Engineering

YEFIM FASSER and DONALD BRETTNER

Process Improvement in the Electronics Industry, 2/e

WILLIAM B. ROUSE

Strategies for Innovation

ANDREW P. SAGE

Systems Engineering

HORST TEMPELMEIER and HEINRICH KUHN

Flexible Manufacturing Systems: Decision Support for Design and Operation

WILLIAM B. ROUSE

Catalysts for Change: Concepts and Principles for Enabling Innovation

LIPING FANG, KEITH W. HIPEL, and D. MARC KILGOUR

Interactive Decision Making: The Graph Model for Conflict Resolution

DAVID A. SCHUM

Evidential Foundations of Probabilistic Reasoning

JENS RASMUSSEN ANELISE MARK PEJTERSEN, and LEONARD P. GOODSTEIN

Cognitive Systems Engineering

ANDREW P. SAGE

Systems Management for Information Technology and Software Engineering

ALPHONSE CHAPANIS

Human Factors in Systems Engineering

YACOV Y. HAIMES

Risk Modeling, Assessment, and Management, 2/e

DENNIS M. BUEDE

The Engineering Design of Systems: Models and Methods

ANDREW P. SAGE and JAMES E. ARMSTRONG, Jr.

Introduction to Systems Engineering

WILLIAM B. ROUSE

Essential Challenges of Strategic Management

YEFIM FASSER and DONALD BRETTNER

Management for Quality in High-Technology Enterprises

THOMAS B. SHERIDAN

Humans and Automation: System Design and Research Issues

ALEXANDER KOSSIAKOFF and WILLIAM N. SWEET

Systems Engineering Principles and Practice

HAROLD R. BOOHER

Handbook of Human Systems Integration

JEFFREY T. POLLOCK and RALPH HODGSON

Adaptive Information: Improving Business Through Semantic Interoperability, Grid Computing, and Enterprise Integration

ALAN L. PORTER and SCOTT W. CUNNINGHAM

Tech Mining: Exploiting New Technologies for Competitive Advantage

REX BROWN

Rational Choice and Judgment: Decision Analysis for the Decider

WILLIAM B. ROUSE and KENNETH R. BOFF (editors)

Organizational Simulation

HOWARD EISNER

Managing Complex Systems: Thinking Outside the Box

STEVE BELL

Lean Enterprise Systems: Using IT for Continuous Improvement

J. JERRY KAUFMAN and ROY WOODHEAD

Stimulating Innovation in Products and Services With Function Analysis and Mapping

WILLIAM B. ROUSE

Enterprise Transformation: Understanding and Enabling Fundamental Change

JOHN E. GIBSON, WILLIAM T. SCHERER, and WILLIAM F. GIBSON

How to Do Systems Analysis

WILLIAM F. CHRISTOPHER

Holistic Management: Managing What Matters for Company Success

WILLIAM B. ROUSE

People and Organizations: Explorations of Human-Centered Design

GREGORY S. PARRELL, PATRICK J. DRISCOLL, and DALE L. HENDERSON

Decision Making in Systems Engineering and Management

MO JAMSHIDI

System of Systems Engineering: Innovations for the Twenty-First Century

ANDREW P. SAGE and WILLIAM B. ROUSE

Handbook of Systems Engineering and Management, Second Edition

JOHN R. CLYMER

Simulation-Based Engineering of Complex Systems

KRAG BROTHBY

Information Security Governance: A Practical Development and Implementation Approach