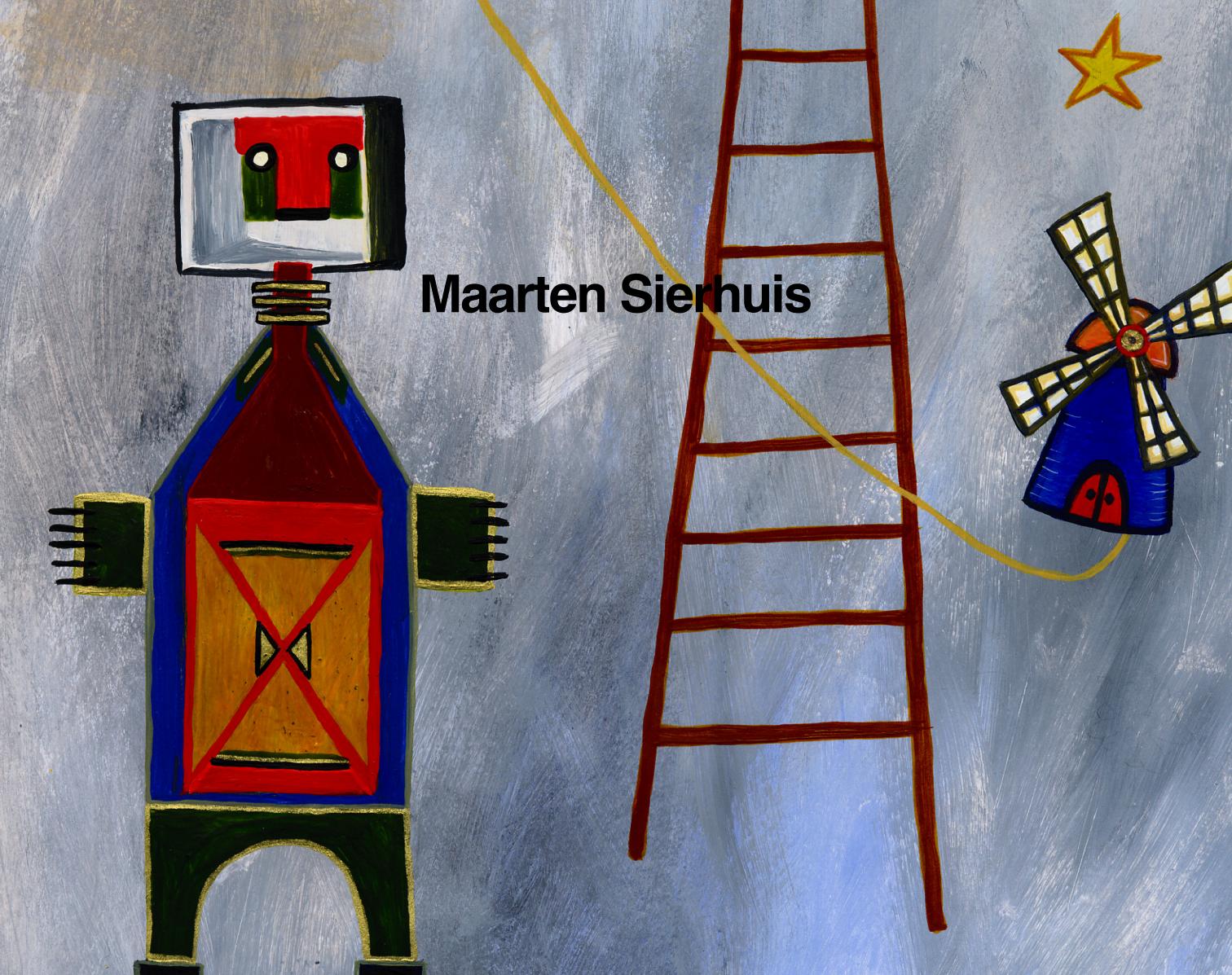




Modeling and Simulating Work Practice

BRAHMS: a multiagent modeling and simulation language for work system analysis and design



Maarten Sierhuis

Modeling and Simulating Work Practice

BRAHMS: a multiagent modeling and simulation language for work system analysis and design

Maarten Sierhuis

Modeling and Simulating Work Practice

BRAHMS: a multiagent modeling and simulation language for work system analysis and design

ACADEMISCH PROEFSCHRIFT

ter verkrijging van de graad van doctor

aan de Universiteit van Amsterdam

op gezag van de Rector Magnificus

prof. dr. J.J.M. Franse

ten overstaan van een door het college voor promoties ingestelde

commissie, in het openbaar te verdedigen in de Aula van de Universiteit

op vijdag 28 september 2001, te 14:00 uur

door

MAARTEN SIERHUIS

geboren te Apeldoorn

Promotor: Prof. Dr. B.J. Wielinga / Prof. Dr. R. de Hoog

Co-promotor: Dr. W.J. Clancey

Faculteit der Psychologie

Universiteit van Amsterdam

Promotiecommissie:

Prof. Dr. H. Akkermans Vrije Universiteit, Amsterdam

Prof. Dr. W van Aalst Technische Universiteit Eindhoven

Prof. Dr. C.W. de Dreu Universiteit van Amsterdam

Prof. Dr. J. Breuker Universiteit van Amsterdam

Dr. P. van den Besselaar Universiteit van Amsterdam

The National Aeronautics and Space Administration funded this research through the NASA Cross-Enterprise program.

For Mary Ellen, Nicholas and Megan

PREFACE

This dissertation is the result of a long traveled road, sometimes flat and smooth while at other times with steep hills and a rough surface. This is not unusual, and indeed most doctoral students go through a long and difficult process of finishing their dissertation. However, every dissertation is different with a different history and a different road that has been traveled. I will briefly share with you some of the travels leading to this dissertation.

The road towards my Ph.D. started back in 1992, the first time I met Bill Clancey at NYNEX Science & Technology. Over the next six years, under the research leadership of Bill, and the help of several scientists at NYNEX and the Institute for Research on Learning, we started the development of "Workframe," a tool for "representing how people really work." In 1994 I became the project leader for the project. The project grew to four full-time people, with a patent application in 1997 for our tool—by then called "Brahms." It was at that time that I became seriously interested in pursuing my Ph.D. It became clear to me that Brahms was a new kind of modeling and simulation tool, different from any other tool I had seen and worked with. More and more, I became interested in understanding the use of the tool and the question of how to represent "how people really work." Although we had developed the language and the tool, based on Bill Clancey and Dave Torok's first prototype, nobody had developed a significant model with the tool, and therefore nobody really understood what it meant to model and simulate a work practice using Brahms.

As often, two independent roads come together leading to the same destination. In my case, it was my previous work with Robert de Hoog and Bob Wielinga, in 1986, which led me to this thesis. Back in 1986 I worked on the PEES project (translation from Dutch: Project Experimental Expert System). The PEES project was a nationally funded research project between the University of Amsterdam (in particular the KADS-group), several commercial firms (Digital Equipment, SEMA Group), and the social security department of the city of Amsterdam. I participated in this project as a developer of an expert system for part of the Dutch social security law. This was my thesis work for finishing my Informatics degree (*Hogere Informatica Opleiding*). It was during this time that I met both Robert de Hoog and Bob Wielinga, and became familiar with the KADS methodology. Strangely enough, it was also the first time I read the research papers of Bill Clancey about meta-level issues of knowledge representation. It was then that my interest in knowledge modeling and representation languages started.

In October 1996, I contacted Robert de Hoog with the question if he would consider supporting me in doing my doctoral dissertation at the Social Sciences and Informatics group (SWI) at the University of Amsterdam. During a trip to The Netherlands I met with Robert and explained my topic. Robert agreed to support me with the knowledge that, back in the States, Bill was supporting me. My advisors were in place, and I was ready to start the work. However, being a project leader for the Brahms project at NYNEX, and having a family, made for very little extra time to work on my thesis. In October 1997, NYNEX merged with Bell Atlantic and the research at NYNEX Science & Technology was stopped. Our funding ended and the project folded. It was then that Bill started the contact with NASA Ames Research Center. By January 1998 Bill had moved to NASA Ames Research Center with the goal of re-establishing the Brahms project. In April 1998, I moved to NASA Ames as well, and it was then that had the time to finish my dissertation, as well as developing the Brahms team. Part of the result of this move lies in front of you, and for the other part I am proud to say that the Brahms team currently consists of eight researchers, Brahms modelers, and Ph.D. students.

It is obvious that my work is the result of guidance, collaborations, and friendships with many people. First of all, I am forever in debt to my friend and mentor Dr. Bill Clancey. Without his ideas for Brahms, his advise on becoming a proficient researcher in the United States, as well as his personal friendship that goes beyond a working relationship, it would have been extremely difficult to come to the end of this road. I am extremely thankful to have Bill as one of my main advisors. Secondly, I thank Prof. Dr. Robert de Hoog for his support and trust in me, while we were 9000 miles apart. I don't think I could have done this without Robert's experience as a thesis advisor, the internet, and Robert's trust in me. This has been a remarkable collaboration. Robert's positive responses on the chapters I sent him were always a source of renewed inspiration. I have been in the lucky circumstance to have the support of two world-renowned scientists from the United States and Europe. While Bill was always there to help me with the day-to-day Brahms research,

Robert was but an e-mail away with his advice and experience on how to write a doctoral dissertation. Thirdly, I want to thank Prof. Dr. Bob Wielinga for his support and the creation of the KADS methodology. Without the original KADS work I would have never been in the position to take the initial road back in 1986.

Besides my thesis promtors, I would like to thank my good friend and colleague Al Selvin for all our good discussions during our commute to and from our office in White Plains, NY, and his skepticism about Brahms. I am convinced that a researcher needs at least one skeptic in his circle of research friends, because a good skeptic will keep you on your toes, and will always question the things you are so convinced of. I am also grateful for the Brahms development team, in particular Ron van Hoof. Without Ron there would be no Brahms environment based on quality of software engineering. His capable software engineering skills, his knowledge of Java and all the state-of-the-art software tools on the market these days is truly unbelievable. Ron was always there, from 1996 till today. He listens to Bill and my needs and complaints about the state of Brahms, and quickly implements those features we need. I am also thankful for Mike Scott's incredible graphics design mind. Without Mike's design of the AgentViewer application that allows us to display the result of a multi-agent Brahms simulation in a time-line view, it would be very difficult to explain the outcome of a Brahms simulation. I thank Ted Shab for implementing this complex graphics application, in Visual Basic no less.

Las but not least, I have to thank my family. Their support has been more than anyone can expect. I therefore dedicate this dissertation to Mary Ellen Sierhuis, my best friend, partner and mother of my children. She was willing to move our home for nine years from New York to the Bay Area in California, just so I could follow my dream. Her sacrifice has been unparalleled. Without her compromise and understanding I would have never been able to do this. This, of course, also counts for my two children, Nick and Megan. Their sacrifice, although they had no choice in the matter, will never be forgotten. I am forever grateful to all three of them.

Before I end with this preface, I want to briefly add something about the current state of Brahms. At this moment Brahms is being developed at NASA Ames Research Center. Over the last three years the tool has been completely redesigned and implemented in the Java programming language, and ported to Windows, Linux, and Solaris. The Brahms simulation engine can be viewed as a virtual machine implemented on top of the Java virtual machine. We are continuing to develop Brahms tools making it easier for others to use the tool and create complex multiagent models. We are also developing Brahms as a multiagent programming environment, as well as a methodology to support a complete life cycle for developing human-centered systems. Currently, Brahms can be downloaded for free (for research purposes) from our web site <http://www.agentisolutions.com>.

Maarten Sierhuis
Fremont, CA, July 2001.

CONTENTS

LIST OF FIGURES	V
------------------------	----------

LIST OF TABLES	XI
-----------------------	-----------

1. INTRODUCTION	1
------------------------	----------

1.1 PROBLEM STATEMENT	2
1.2 RESEARCH QUESTIONS	3
1.3 FRAMEWORK FOR MODELING AND SIMULATION	3
1.4 THESIS OUTLINE	5

PART 1: THEORY

2. APPROACHES FOR MODELING HUMAN BEHAVIOR	9
--------------------------------------------------	----------

2.1 BUSINESS PROCESS MODELING	9
2.2 COGNITIVE MODELING	24
2.3 DISTRIBUTED ARTIFICIAL INTELLIGENCE	31
2.4 COMPUTATIONAL ORGANIZATION THEORY	39
2.5 CONCLUSION	43

3. THEORY OF MODELING WORK PRACTICE	47
--------------------------------------------	-----------

3.1 HISTORY OF PRACTICE	47
3.2 ON THE EPISTEMOLOGICAL LEVEL OF WORK PRACTICE	52
3.3 MODEL-BASED APPROACH	63
3.4 CONCLUSION	68

4. MODELING FORMALISM	69
------------------------------	-----------

4.1 AGENTS AND GROUPS	70
4.2 OBJECTS AND CLASSES	73
4.3 BELIEFS AND FACTS	74
4.4 ACTIVITIES AND WORKFRAMES	78
4.5 GEOGRAPHY	86
4.6 SIMULATION	89

PART 2: APPLICATIONS

5. RESEARCH DESIGN	99
---------------------------	-----------

5.1 SCIENTIFIC METHODOLOGY	99
5.2 USE OF COMPUTATIONAL MODELS IN SIMULATION	101
5.3 THE CASE STUDIES	103

6. CASE STUDY 1: APOLLO 12 ALSEP-OFFLOAD **105**

6.1	APOLLO 12 AND THE ALSEP OFFLOAD	106
6.2	THE AGENT MODEL	108
6.3	THE OBJECT MODEL	112
6.4	THE GEOGRAPHY MODEL	114
6.5	THE ACTIVITY MODEL	121
6.6	THE BEHAVIORAL MODEL	127
6.7	VOICE-LOOP COMMUNICATION	142
6.8	OBJECT INTERACTION	146
6.9	VERIFICATON AND VALIDATION	151
6.10	CONCLUSION	166

7. CASE STUDY 2: PREDICTING SITUATED ERRORS **169**

7.1	HFE DEPLOYMENT	170
7.2	PREDICTIVE MODELING	174
7.3	DEVIATIONS FROM NOMINAL PROCEDURES	184
7.4	PURPOSE OF GENERAL REPRESENTATIONS	187
7.5	VERIFICATION AND VALIDATION	188
7.6	CONCLUSION	210

8. CASE STUDY 3: DESIGNING HUMAN-ROBOT COLLABORATION **213**

8.1	VICTORIA MISSION	215
8.2	PROBLEMS WITH AUTOMATED PLANETARY SURFACE EXPLORATION	216
8.3	EVALUATION CRITERIA FOR USE OF BRAHMS IN DESIGN	217
8.4	MODEL VERIFICATION AND VALIDATION	219
8.5	MISSION OPERATIONS WORK SYSTEM DESIGN	221
8.6	MODEL SIMULATION SCENARIO	227
8.7	ROVER ACTIVITY	229
8.8	TEAM ACTIVITIES	232
8.9	CALCULATING ROVER ENERGY USAGE	237
8.10	CONCLUSION	242

9. CONCLUSIONS **249**

9.1	REFLECTIONS ON RESEARCH QUESTION	249
9.2	SCALING OF BRAHMS MODELS	257
9.3	SCIENTIFIC CONTRIBUTIONS	258
9.4	FUTURE RESEARCH	260

A. BRAHMS LANGUAGE **263**

A.1.	NOTATION CONVENTIONS (BACKUS NAUR FORM)	263
A.2.	SYNTAX OF A MODEL (MOD)	264
A.3.	SYNTAX OF AN AGENT (AGT)	266
A.4.	SYNTAX OF A GROUP (GRP)	267
A.5.	SYNTAX OF AN OBJECT CLASS (CLS)	269
A.6.	SYNTAX OF AN OBJECT (OBJ)	271
A.7.	SYNTAX OF A CONCEPTUAL OBJECT CLASS (COC)	272

A.8. SYNTAX OF A CONCEPTUAL OBJECT (COB)	273
A.9. SYNTAX OF AN ATTRIBUTE (ATT)	274
A.10. SYNTAX OF A RELATION (REL)	276
A.11. SYNTAX OF AN INITIAL-BELIEF (BEL)	277
A.12. SYNTAX OF AN INITIAL-FACT (FCT)	278
A.13. SYNTAX OF A WORKFRAME (WFR)	279
A.14. SYNTAX OF A THOUGHTFRAME (TFR)	280
A.15. SYNTAX OF A COMPOSITE ACTIVITY (CAC)	281
A.16. SYNTAX OF A PRIMITIVE ACTIVITY (PAC)	283
A.17. SYNTAX OF A CREATE OBJECT ACTIVITY (COA)	285
A.18. SYNTAX OF A MOVE ACTIVITY (MOV)	287
A.19. SYNTAX OF A COMMUNICATE ACTIVITY (COM)	289
A.20. SYNTAX OF A BROADCAST ACTIVITY (BCT)	291
A.21. SYNTAX OF A PRECONDITION (PRE)	293
A.22. SYNTAX OF A CONSEQUENCE (CON)	297
A.23. SYNTAX OF A DETECTABLE (DET)	298
A.24. SYNTAX OF AN AREA DEFINITION (ADF)	300
A.25. SYNTAX OF AN AREA (ARE)	301
A.26. SYNTAX OF A PATH (PAT)	302
A.27. SYNTAX OF A TRANSFER DEFINITION (TDF)	303
A.28. VARIABLE (VAR)	304
 REFERENCES	 306
 SUMMARY	 312
 SAMENVATTING	 314

LIST OF FIGURES

Figure 1-1. Basic entities in M&S and their relationships	4
Figure 2-1. Life Cycle Model of Breakpoint BPR™	10
Figure 2-2. Components of a workflow model	11
Figure 2-3. Modeling error conditions with a branch and a join task	12
Figure 2-4. Modeling simultaneous work	13
Figure 2-5. Spawn semantics	13
Figure 2-6. Modeling job delay	14
Figure 2-7. Order processing in the business network architecture (BNA) organization	20
Figure 2-8. The Soar production system	26
Figure 2-9. Representation of an ACT-R chunk	27
Figure 2-10. The ACT-R production system	28
Figure 2-11. The total cognitive system	29
Figure 2-12. Human and Automated Pilots interact with the DIS environment using Distributed Simulators	33
Figure 2-13. Phoenix layers	35
Figure 2-14. Screen display of Phoenix's situation-specific model	36
Figure 3-1. Relation of a model of work to a description of the work practice	52
Figure 3-2. Dimensions of behavior	55
Figure 3-3. Activity subsumption	56
Figure 3-4. Mediated relationship of artifacts in activities	62
Figure 3-5. Describing real world work practice with computational modeling	65
Figure 3-6. Modeling proces	67
Figure 4-1. Beliefs and facts Venn diagram	78
Figure 4-2. Taxonomy for groups, agents, beliefs, activities, and workframes	80
Figure 4-3. Workframe-Activity hierarchy	83
Figure 4-4. State-transition diagram for frame instantiations	92

Figure 4-5. Multi-tasking in Brahms	95
Figure 5-1. Research Process	99
Figure 6-1. SEQ Bay and RTG Cask located on the side of the LM	107
Figure 6-2. Apollo 12 Surface Checklist 47 for the ALSEP Offload	107
Figure 6-3. Apollo Agent Model design	109
Figure 6-4. Brahms source code of the agent model	110
Figure 6-5. Agent Model in the Brahms Model Builder	111
Figure 6-6. Apollo Object Model design	112
Figure 6-7. Apollo 12 LM and SEQ Bay Brahms objects	113
Figure 6-8. Apollo 12 contained artifacts	114
Figure 6-9. Apollo Geography Model design	115
Figure 6-10. Geography Model Brahms source code	115
Figure 6-11. Apollo 12 ALSEP compiled Geography Model	116
Figure 6-12. Agent initial location	117
Figure 6-13. Move activity source code	117
Figure 6-14. Pete Conrad and Al Bean moving to the SEQBayArea	118
Figure 6-15. Apollo 14 Landing site and ALSEP Offload area of activity	118
Figure 6-16. Pete Conrad's location beliefs	119
Figure 6-17. Moving contained object source code	120
Figure 6-18. Moving contained object simulation	121
Figure 6-19. Activities in practice	122
Figure 6-20. The Brahms ALSEP Offload group and activities model	124
Figure 6-21. Apollo 12 LSJ: ALSEP Offload transcription	125
Figure 6-22. The OpenSEQ BayDoor composite-activity, sub-activities, and workframes	127
Figure 6-23. The AlsepOffload workframes	127
Figure 6-24. AlsepOffload workframe-activity subsumption hierarchy	128
Figure 6-25. Open SEQ Bay door activity sequence model	129
Figure 6-26. AlsepOffloading workframe	131

Figure 6-27. AlsepOffloafing WFI for agent AlBean	133
Figure 6-28. AlsepOffloafing WFI for agent PeteConrad	134
Figure 6-29. Workframes within the composite AlsepOffload activity	135
Figure 6-30. AlBean's Step 1 Activity-Context Tree	136
Figure 6-31. PeteConrad's Step 1 Activity-Context Tree	137
Figure 6-32. AlBean's Step 2 Activity-Context Tree	139
Figure 6-33. PeteConrad's Step 2 Activity-Context Tree	140
Figure 6-34. AlsepOffload activity agent timeline	141
Figure 6-35. Extra-Vehicular Communication System	143
Figure 6-36. Voice-loop communication via LmComCircuit	145
Figure 6-37. Voice-loop library model group hierarchy	146
Figure 6-38. NASA picture AS12-47- 6913	147
Figure 6-39. Al Bean taking two photographs of Pete Conrad	148
Figure 6-40. Taking a photograph	149
Figure 6-41. The PeteConrad agent taking photographs	150
Figure 6-42. Photographs AS12-47-6783, 84, and 85 by Pete Conrad	150
Figure 6-43. Simulation model verification and validation in the modeling process	152
Figure 6-44. The conceptual model	154
Figure 6-45. Voice loop transcription data from the Apollo LSJ	169
Figure 6-46. Voice loop transcription matched to activities	155
Figure 6-47. Voice loop activity time analysis	155
Figure 6-48. Brahms compile-debug cycle	156
Figure 6-49. Brahms model development cycle	157
Figure 6-50. AgentViewer application	157
Figure 6-51. Black-box validation: comparison with the real system	159
Figure 6-52. Al Bean's RemovePkg1 and RemovePkg2 activities	161
Figure 6-53. Pete Conrad's RemovePkg1 and RemovePkg2 activities	161
Figure 6-54. Workframe with communication utterance from Apollo LSJ	164

Figure 6-55. Talk activity to validate communication	164
Figure 6-56. Voice loop communication for OpenSEQBayDoor activity	165
Figure 7-1. HFE deployment configuration	170
Figure 7-2. Apollo 16 summary time line	171
Figure 7-3. Apollo 16 HFE deployment timeline procedures for LMP	172
Figure 7-4. HFE Deployment Activity Model	173
Figure 7-5. Astronaut movement during HFE Deployment activity	174
Figure 7-6. Hardwired activity plan	175
Figure 7-7. Dynamic plan execution activity	176
Figure 7-8. ReadCuffChecklist workframe	176
Figure 7-9. Workframe for DeployingHfeProbeNo1	177
Figure 7-10. DetermineNextActivity source code	178
Figure 7-11. Voice-data schedule	179
Figure 7-12. VoiceData definition	179
Figure 7-13. Voice-data example	180
Figure 7-14. The Question & Answer conversation policy	181
Figure 7-15. Workframe SenderPolicy1	182
Figure 7-16. CommunicateVoiceData activity	182
Figure 7-17. Workframe ReceiverPolicy1	183
Figure 7-18. RequestForData Activity	184
Figure 7-19. Default error-recovery activity workframes and thoughtframes	186
Figure 7-20. CheckOutProblem workframe & activity hierarchy	187
Figure 7-21. ErrorHandling workframe & activity hierarchy	187
Figure 7-22. Reading the first activity	190
Figure 7-23. ALSEP Package Placement Activity	190
Figure 7-24. ALSEP Package Placement Activity Verification	191
Figure 7-25. PlacePkgsOnSurface Workframe	192
Figure 7-26. HfeEquipmentPreparation Activity Verification	193

Figure 7-27. DeployingHfe/2Probe Activity Verification	194
Figure 7-28. Drilling BoreStem1 in the surface	195
Figure 7-29. Drilling BoreStem2 in the surface	196
Figure 7-30. Conversation Policy Verification	197
Figure 7-31. Voice-Data Communication Verification	198
Figure 7-32. AskForDrillingEndMark workframe of CapCom agent	200
Figure 7-33. Asking For Voice-Data	200
Figure 7-34. Asking for Voice-Data	201
Figure 7-35. Default error-recovery procedure	203
Figure 7-36. Error Object Data	203
Figure 7-37. CapCom's error-recovery decision	204
Figure 7-38. CreateHfeCableError workframe	204
Figure 7-39. Polymorphic Error Handling Behavior	206
Figure 7-40. HfeDeployment group's HandleError Activity tree,	206
Figure 7-41. Workframe DeployingHfeProbeErrorHandling in the HfeDeployment Group	206
Figure 7-42. Activity HandleError in the HfeDeployment Group	207
Figure 7-43. Workframe DeployingHfeProbeNo2	208
Figure 7-44. Simulating DeployingHfeProbe ErrorHandling Procedure	208
Figure 8-1. Victoria Rover	217
Figure 8-2. Transforming objectives into experimental frames	218
Figure 8-3. Victoria work system	223
Figure 8-4. Victoria Agent Model	224
Figure 8-5. Victoria Object Model	226
Figure 8-6. Victoria Geography Model	227
Figure 8-7. Victoria Rover scenario activities	230
Figure 8-8. Simulation of first uplink command activities	234
Figure 8-9. Simulation of downlink and second uplink command activities	236
Figure 8-10. Rover energy used in drilling activity from simulation history database	240

Figure 8-11. Instrument energy used in drilling activity from simulation history database	241
Figure 8-12. Rover total energy usage during traverse into crater	241
Figure 8-13. Battery power left, based on constraint (3.0)	242
Figure 9-1. Relation of Brahms to other models of work	253

LIST OF TABLES

Table 2-1. Workflow modeling limitations	23
Table 2-2. Limitations of cognitive modeling and simulation	30
Table 2-3. Limitations of Distributed Artificial Intelligence	38
Table 2-4. Limitations of Computational Organization Theory	42
Table 2-5. Human behavior model comparison	45
Table 4-1. Frame instantiation states	91
Table 4-2. Frame state-transitions	92
Table 5-1. Use of Computational Models	101
Table 5-2. Kir's levels of system knowledge	102
Table 5-3. System problems related to model use and types	103
Table 5-4. Case study descriptions	104
Table 6-1. ALSEP experiments for Apollo missions	106
Table 6-2. ALSEP Offload activity timetable	123
Table 6-3. Open SEQ Bay door activity	126
Table 6-4. Data sources used during experiment	153
Table 6-5. Aspects of modeling work practice	156
Table 6-6. Calculated activity times based on real-world data	160
Table 6-7. Activity times for LMP Al Bean from simulation history database	160
Table 6-8. Activity times for CDR Pete Conrad from simulation history database	161
Table 6-9. Activity times for LMP Al Bean including communication delay	162
Table 6-10. OpenSEQBayDoor activity with communication	163
Table 6-11. Agent speech communication validation	165
Table 6-12. Answering the research questions	166
Table 7-1. Data sources used during experiment	188
Table 7-2. Summary of Experiment Outcome	211

Table 8-1. Simulation output variables	219
Table 8-2. Relation of data flow process to functions (from (Wall and Ledbetter 1991))	222
Table 8-3. Functional activity distribution over Victoria teams	224
Table 8-4. Victoria rover instruments used during scenario	228
Table 8-5. SATM collecting lunar surface sample activity times (from Honeybee Robotics, Ltd.)	229
Table 8-6. VictoriaRover and Science Instruments activities and times in seconds	231
Table 8-7. VictoriaRover agent output variables	231
Table 8-8. Victoria Team activities and time in seconds	237
Table 8-9. Energy usage for the rover during scenario	238
Table 8-10. Rover power consumption data	239
Table 8-11. Outcome variable evaluation	244
Table 9-1. Effects of not being able to simulate in Case Study 1	255
Table 9-2. Effect of not being able to simulate in Case Study 2	256
Table 9-3. Effect of not being able to simulate in Case Study 3	256
Table 9-4: Synopsis of the notation used	263

1. INTRODUCTION

In the past decade there has been a move in business management towards *collaborative workplaces*. There has been a shift from a hierarchy-based organizational culture toward a culture in which collaboration will be (Marshall 1995):

- A new way of working in which people collaborate and cooperate together
- A new work ethic that recognizes that work is accomplished by people, regardless of the simplicity of the work process

There are significant benefits from the implementation of collaborative work models, including (Marshall 1995):

- A move toward internal collaboration with the goal of competing externally
- A faster, higher-quality, and customer-driven decision-making model
- A reduction in cycle time and elimination of non-value added work
- Greatly increased return on investment
- Reductions in internal conflicts

After companies have spent millions of dollars and laid off thousands of people, business process reengineering (BPR) consultants admit "[We] forgot about the people"¹ (Davenport 1995). Some writers propose a new method for work design (Weisbord 1987). Taylorism has been replaced with involvement of the workers themselves in the redesign of their work. Holistic work system analysis, a combination of organizational design (OD) and socio-technical system (STS) techniques, lets the workers design their own work. It gives control back to the people who do the work and lets management manage the input to the work process, while the workers manage their work and make sure the output is consistent with the mission of the organization.

In designing and implementing collaborative workplaces in a way that embraces these changes in management views, we need to understand the way people work together, and analyze workplace culture and ethics in order to suggest improvements and design the changes. Both the way people collaborate, as well as the culture of an organization is encompassed in the communities of practice of an organization—the work practices of the people (Wenger 1997). Therefore, work practice analysis, design methods and tools need to be developed that allow analysts, designers, workers, and managers to understand not only the work process, but also the work practice of an organization.

A work practice is defined as the collective activities of a group of people who collaborate and communicate, while performing these activities synchronously or asynchronously. Most often, people view work merely as the process of transforming input to output. For example, in an automobile manufacturing process the input and output of the work is well defined. Sometimes, however, it is more difficult to describe the input and output of the work. For example, consider a soccer match between two professional soccer teams. It is rather difficult to define the input and output of this type of work, although most of us would agree that professional soccer players are working. To describe the work of a soccer team, we quickly fall to descriptions about *teamwork* and *collaboration on the field*.

I claim that the individual activities that make up the work practice not only have to do with the transformation of input to output, but more importantly with the collaboration between individuals in action, in pursuit of a goal. Imagine soccer players who collaborate their activities of kicking a soccer ball in pursuit of scoring a goal. Just focusing on the input and output of each individual activity of soccer players would not

¹ 'Next Big Thing': Re-Engineering Gurus Take Steps to Remodel Their Stalling Vehicles, *Wall Street Journal* 11/26/96

only be very difficult, if not impossible, it would also miss the opportunity to understand what is really going on. However, in this century, work has been defined as the transformation of input to output, starting with Frederick W. Taylor's view of work to Michael Hammer's view of business processes (Weisbord 1987) (Hammer and Champy 1993).

In this dissertation, I take a different view. I am interested in describing work as a practice, a collection of psychologically and socially situated collaborative activities between members of a group. I am modeling work practice to understand how, when, where, and why collaborative activities are performed, and to identify the effects of these activities, as well as to understand the reasons why these activities occur in the way they do. Therefore, the central theme is to find a representation for modeling work practice. I will define what I mean by the term *work practice*, and how it relates to collaboration and communication between people. I will also present a multiagent modeling language to represent models of work practice. These models can be simulated in order to show the effects of the activities of people and their communication, being situated in a geographical environment, and using tools and artifacts to perform their collaborative work.

1.1 PROBLEM STATEMENT

In an effort to re-design an order process at NYNEX², the designers from the Work Systems Design group at NYNEX Science & Technology³ used an off-the-shelf business process simulation tool to model the old and the new work processes. The newly created design of this work process included a coordination role, the turf coordinator (TC). The function of the TC was to keep track of the incoming T1-orders from beginning to end, at each moment coordinating the work activities between several individuals in different organizations and locations throughout Manhattan. For example, when a technician in the field was ready to test a T1 data-line installation at a customer site, the TC would coordinate a circuit test with the responsible tester in the central office. It was only at these moments that the TC was actually engaged in a collaborative work activity that was essential to the workflow process. Most of the other work activities for this role (such as meetings, tracking orders, calling people) did *not* include activities that operated directly on the work product (i.e. the implementation of a high-speed data line) (Sachs 1995).

A workflow model shows the sequential tasks through which a work product (such as an order) flows through the process. In each task, resources work for an amount of time on the work product—"touching the object." The workflow-modeling paradigm excludes work activities in which people do not directly "touch" the work product (see chapter 2). In other words, critical tasks that make a work process succeed or more productive, but in which people are not directly working on the work product, are not be represented.

Coordination roles are, in essence, roles that manage or coordinate other people's work activities. It is seldom that a TC actually "touches" the work product during the process. Consequently, it was very difficult to represent most of the TC's work in the workflow model. The result was that the workflow model did not include most of the tasks of the TC, and therefore could not be used to explain the need for this new coordination role in the newly designed T1-order process. The work system designers were frustrated because they could not explain the importance of this new role using the model, while the rationale of the overall design was primarily based on the introduction of this new coordination role.

Ironically, the company adopted the design. The job-flow time and cost statistics from the simulation of the new workflow model showed a more than significant improvement over the statistics, from the simulation of the old model. Since management decided to go ahead with the new design based on the output of the workflow simulation model, one could argue that the model was helpful in the effort. However, although the management decision was at least partially based on the simulation output, the work system designers complained about the usefulness of the model and the simulation during and after the design phase.

According to the work system designers, the model could not be used during the design sessions because the language, design formalisms and representations used to talk about the work practice and activities of

² NYNEX was one of the former Bell Telephone Operating Companies. In October of 1997, the NYNEX and Bell Atlantic corporations merged. The resulting company was called Bell Atlantic. In 2000, the Bell Atlantic and GTE corporations merged to create Verizon.

³ NYNEX Science & Technology was the research and development center of the former NYNEX telephone company. As a result of the Bell Atlantic merger, NYNEX Science & Technology was dismantled, and the Work Systems Design group ceased to exist.

the people in the work process were incompatible with that of the workflow paradigm. Although useful, a workflow model typically omits collaboration, “off-task” behaviors, multi-tasking, interrupt and resume, informal interaction, and geography. In other words, workflow omits work practice (Clancey et al. 1998). The workflow model was created as a separate activity outside the design team sessions. The person creating the workflow simulation model—the modeler—would interview design team members, and from this create the workflow model based on his interpretation of the design. There was feedback from members of the design team on the workflow model, but at no time did the model play a significant role in the design activities of the design team.

In order to have a computer model that is convincing to management, helpful in the analysis and design process, helpful in the understanding of the new design, and helpful in the communication of that design, we⁴ started our effort in developing a modeling language and simulation environment that allows us to model the work practice of people in a work process. The Brahms language and simulation environment is the result of this effort.

In this thesis, I am describing the use of Brahms as a tool for modeling and simulating work practice. I first develop a theory of modeling work practice. Then, by describing three case studies, I test if Brahms operationalizes the theory and is a valid tool for modeling and simulating work practice.

1.2 RESEARCH QUESTIONS

There are many approaches to understanding work processes and practices—workplace observation, role-playing, interviewing, peripheral participation, and training. The approach that will be researched in this dissertation is a multiagent simulation approach for analyzing work practices. The following questions will be addressed:

- 1. How can we model an organization’s work practice in such a way that we include people’s collaboration, “off-task” behaviors, multi-tasking, interrupted and resumed activities, informal interaction, knowledge and geography?**

To answer this question we need a modeling paradigm that allows us to describe these aspects. I describe a multiagent activity-based modeling language and simulation method, and present case studies that evaluate the modeling paradigm and associated simulation method.

- 2. What is the added value of computer simulation in a model-based approach?**

To answer this question I will define what I mean by “added value”, and define operational criteria to answer the question based on the case studies. My hypothesis is that simulation adds significant value to the understanding of the work system, because without simulation, the changes over time in a complex system are difficult to comprehend.

1.3 FRAMEWORK FOR MODELING AND SIMULATION

In this section, I establish a framework for modeling and simulation. Modeling is the static formal or informal representation of relationships between elements of a system, while simulation is the execution of a formal computational model over time. A computational model describes a class of systems in terms of operations on entities, in which the operations are described in computational terms:

$$\text{Computational Model} = \text{Formal Representation} + \text{Operator Calculus}$$

Whereas research in work practice is done in real-life settings, computational modeling experiments can be done at lower cost, time and effort (Stasser 1988). Scenario’s can be addressed in a systematic fashion, and replicated as many times as necessary. A computational model is also useful in case there is

⁴ "We" are the people who were involved in the contemplation and development of Brahms. With the risk of leaving someone out, this list includes: Bill Clancey, Dave Torok, Ron van Hoof, Jim Euchner, Pat Sachs, Gitty Jordan, David Moore, Madeline Ritter, Peter Henschel and Ed Thomas, from NYNEX Science & Technology and the Institute for Research on Learning. I am grateful for having worked with all these individuals, without whom I would never have been able to attempt this dissertation.

insufficient data or where data is unobtainable, such as where the problem domain is too risky in terms of safety, or when designing a system that does not yet exist. Another valuable aspect of computational modeling is the ability to incorporate theories in the model to be tested in the real system. When including new propositions to be tested, unexpected novel results may occur that can lead to insights, and further research.

These characteristics make computational models especially useful for studying human activity systems (Checkland and Scholes 1990), especially those where collaboration and teamwork are an essential part of the system. Computational models force analysts and designers to be specific about the relationship between the entities in the system; in this case the way people work together. Observations, concepts, and anecdotes from the real world need to be systematically formalized into computational operations. This enables analysts and designers to be systematic and complete in describing the behavior of a group of people as individuals.

In this thesis, I describe a new conceptual and computational formalism for representing the work practice of a group of people in an organization. I will provide empirical evidence supporting the claim that this new formalism allows us to model and simulate work practice, such that this system can then be used as a tool for studying and/or designing work practice in real-world systems.

1.3.1 Entities of a modeling and simulation framework

Figure 1-1 shows that the basic entities of the framework are *source system*, *model*, and *experimental frame*. The basic relationships between these entities are the *modeling* and *simulation* relationship (Zeigler et al. 2000).

The *source system* is the real or the virtual environment that is being modeled. The source system provides the source data that will be used for the development of the model. This data is captured through observation of the source system and is represented in the *behavior database* or, what I call, the *conceptual model*.

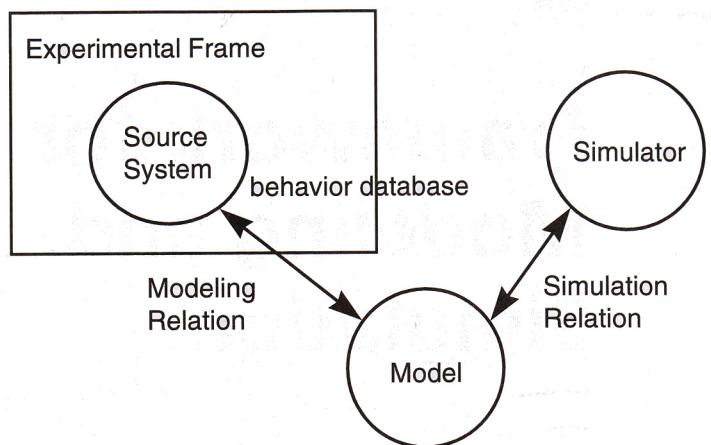


Figure 1-1. Basic entities in M&S and their relationships (borrowed from (Zeigler et al. 2000))

The *experimental frame* is the specification of the conditions under which the source system is being observed or experimented with. As such, this is the operationalization of the objectives for the modeling and simulation effort. Zeigler, et al, define two equally valid views of an experimental frame. The first view is that it defines the type of data elements that will go into the database, and therefore the observational framework for the source system. The second views a frame as a system that interacts with the source system to obtain the data of interest, and is therefore the framework that defines how to model the source system. I will show an example of this in the third case study, described in chapter 8.

A *model*, in a computational system, is a source system specification at a generative and structural level. The most common concept of a computational model is that it consists of a set of instructions, rules,

equations, and/or constraints for generating I/O behavior. Therefore, in computational models we write the model with a state transition and an output generation mechanism. The benefit of a computational model is that it has a sound computational foundation and therefore has a definite syntax and semantics that everyone can understand.

The objective in this thesis is to define a computational modeling language and output generation mechanism—a simulator—for modeling and simulating work practice. This language and simulator is called Brahms⁵, and the goal is to use and validate Brahms as an environment for modeling and simulating work practice.

1.4 THESIS OUTLINE

This thesis consists of two parts. Part one is the theory part. In this part, I am developing a theory and model-based methodology for modeling work practice. Based on this, I am hypothesizing that Brahms is a tool for developing models of human work processes at the work practice level—models of work practice. Therefore, I describe the Brahms modeling formalism and simulation capabilities in chapter 4, after the presentation of the theory in chapter 3.

However, chapter 2 first discusses existing modeling and simulation approaches for human behavior, as they are relevant to the research questions presented in this introduction chapter. I describe four approaches from different academic fields. First, I describe a business process modeling approach and tool from the business processes engineering community. This approach is relevant, because it was the source of inspiration at the start of this research. The business process modeling approach that is discussed is workflow modeling and simulation. There are significant shortcomings to this approach to making it useful for representing work practice. These shortcomings are explained and discussed.

The second approach that is discussed is cognitive modeling. This modeling approach is developed in cognitive science. It is relevant to this research, because we are interested in modeling individuals and groups of people, and their ability to act in and reason about their social and physical work context. There are some concepts that can be used from this approach. However, its limitations for modeling work practice are also discussed.

The third approach is that of the distributed artificial intelligence (DAI) community. Distributed AI focuses on multiple cognitive agents, and is thus relevant to the modeling of groups of people working together. I discuss a number of DAI systems that are out in the community, and describe the way they deal with representing the distribution of tasks amongst multiple agents, and how they deal with communication between agents, as well as the physical environment of the domain in which they operate.

The last approach discussed is that of computational organization theory. This approach is mostly rooted in organizational theory and business economy. It is relevant, because it deals with modeling organizations and their behavior. However, as is discussed, the level of modeling is at a more abstract level than the work practice level I am interested in.

Chapter 3 then describes my theory of modeling work practice. It starts with giving the reader a historical perspective of the concept of practice. I then define what I mean with work practice, and describe work practice at an epistemological level. Here I define the concepts and elements that are relevant for modeling work practice. The concepts introduced here are: community of practice, activities, collaboration, communication, artifacts and geography. I then describe a model-based approach for modeling work practice.

Chapter 4 describes the Brahms modeling and simulation language in detail. I hypothesize (in chapter 5) that the Brahms language incorporates the right worldview for modeling work practice, as described in chapter 3. The Brahms language was developed by a group of people at IRL and NYNEX Science &

⁵ The name *Brahms* stands for Business Redesign Agent-based Holistic-Modeling System, an acronym that was humorously coined by one of the original developers, Dave Torok, for our patent application. The acronym stuck, and is now simply used as the name for the environment.

Technology, which I had the pleasure to be part of. This introduction to the Brahms language is necessary to understand part two of the thesis.

Most of the work presented in this thesis was in the application of the Brahms multiagent modeling and simulation environment to three case studies. This is therefore the topic of part two. To test my hypothesis about Brahms and to proof my theory of modeling work practice, I performed three extensive case studies in which I used Brahms to model the work practice of real-world domains.

Chapter 5 describes the research design. In that chapter, I present the research approach and the case studies, and explain the motive for choosing them. If the reader is interested in jumping ahead, and get an overview of the design of the research approach chosen, as well as a quick description of the case studies, it is good to read this chapter before moving on to any other part of the thesis. After chapter 0, the next three chapters each discuss the case studies in great detail. I advise the reader to at least first read chapters 3 and 4, before starting on the case studies, because understanding the results found in the case studies requires a somewhat in-depth understanding of the theory, as well as of the Brahms language and simulation capabilities.

Chapter 6 describes the Apollo 12 ALSEP-Offload case study. In this case study, I modeled the work of the Apollo 12 astronauts on the Moon, offloading the ALSEP packages from the Lunar Module. This is an example of a *descriptive* model.

Chapter 7 describes the Apollo Heat Flow Experiment (HFE) deployment case study. In this case study, I modeled the work of the lunar surface astronauts deploying the HFE, in a more general way. This is an example of a *predictive* model.

Chapter 8 describes modeling a work system design of the mission operations of a mission to the Moon with a semi-autonomous rover. In this case study, I used my work practice modeling approach to model a design of a work system that currently does not exist. This is an example of a *prescriptive* model.

Last, but not least, chapter 9 presents conclusions. In this chapter I present my findings, based on the three case studies. I start with a reflection on the results from the case studies and how they relate to the two research questions presented in this chapter. I then discuss the cost-benefit of modeling work practice using Brahms, as well as the scaling-factor with regards to the size of Brahms models. The three case studies all model relatively small organizations. I briefly address the issue whether the work practice modeling and simulation methodology developed in this thesis, scales up to larger organizations. After that, I discuss some of my scientific contributions to the different scientific communities. This describes my own interpretation, and it should be taken into consideration that I am, obviously, biased in this regard. I end with a discussion of some future research that needs be conducted, in order to make the presented work practice modeling and simulation methodology more robust and complete.

This ends the introduction to this thesis. Next is part one and the discussion of different approaches for modeling human behavior, or chapter 5 for those who want to know more about the design of the research approach and the case studies.

PART 1

Theory

2. APPROACHES FOR MODELING HUMAN BEHAVIOR

This chapter describes a number of modeling approaches. It presents a review of existing relevant research literature to this thesis. The common theme in all the presented approaches is that of modeling human behavior and organization. It starts with a section on business process modeling. This approach is very relevant to this research, because it models work processes as sequences of tasks or flows of products in organizations. In the next section, the approach that is discussed is cognitive modeling. This approach is developed in cognitive science. It is relevant to this research, because we are interested in modeling individuals and groups of people, and their ability to act in and reason about their social and physical work context. The third section describes a distributed artificial intelligence (DAI) approach. Distributed AI focuses on multiple cognitive agents, and is thus relevant to the modeling of groups of people working together. In the fourth section, the last approach discussed is that of computational organization theory. This approach is relevant, because it also deals with modeling organizations at the agent-level.

I end this chapter with a conclusion section in which I relate all these modeling approaches to the different aspects of work practice mentioned in the first research question.

2.1 BUSINESS PROCESS MODELING

One of the most frequently used approaches to modeling human behavior in a business process is modeling the workflow through an organization. This modeling approach is often used in *business process reengineering* (BPR). In this chapter, I will describe workflow modeling, and some of its benefits, but more importantly I will discuss its weaknesses as they relate to modeling human behavior. I will use the SPARKS™ modeling and simulation tool as an example of workflow modeling. However, I argue that all of the issues related to SPARKS™ are systemic to workflow modeling in general.

Workflow modeling is a functional modeling paradigm that models the sequential tasks through which jobs⁶ flow in a business process. A workflow model typically describes the transformation of some sort of work product. In describing this transformation, workflow models focus on the time and cost parameters in each functional transformation—a task. Effectiveness is defined in terms of time and cost; that is, the number of jobs that can be processed in a specific time, and the overall cost of processing a job. Workflow modeling has gained significant popularity in the business world, due to the interest in business process re-engineering to gain more efficiency and cost reduction.

However, workflow models do not particularly focus on an individual's job performance, nor do they specify social and cultural behavior. Resources in workflow models are stochastic variables with specific characteristics, such as cost, work schedules, and other kind of parameters that can be measured. People are treated (modeled) as statistical resources, just like equipment and automation machines. As I will discuss later in this chapter, it is partly because of the limitations of such measurements that workflow models provide a very limited representation of how work is done in practice.

2.1.1 Modeling in business process re-engineering

The 1990s saw the development of many proprietary BPR methodologies. The idea for BPR was made popular in the business community by Hammer and Champy (Hammer and Champy 1993), and Davenport (Davenport 1993). The idea behind BPR is that work processes in many companies need to be evaluated, streamlined and automated in order to stay profitable and competitive (Scott Morton 1991). An example of a BPR methodology is BreakpointBPR™, developed by Coopers & Lybrand (Johansson 1992).

⁶ The term "job" is defined in section 2.1.2

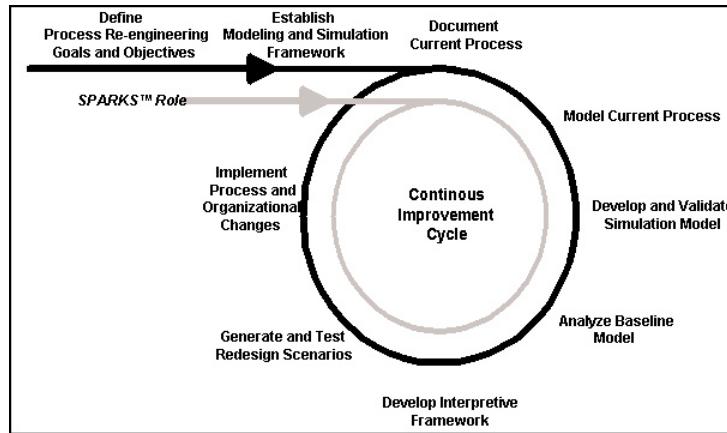


Figure 2-1. Life Cycle Model of Breakpoint BPR™ (borrowed from SPARKS™ training manual)

Figure 2-1 shows the life cycle model of the Breakpoint BPR™ methodology, and the role that a business process modeling and simulation tool plays in this life cycle. The Continuous Improvement Cycle in Figure 2-1 is primarily based on developing, analyzing, and testing a work process, using a workflow simulation tool. As changes to the business process are validated and tested through what-if scenarios in the simulation, the process moves to an implementation phase. Continuous improvement happens in the next cycle of modeling, where the current work process becomes the process that was just implemented. Detailing the methodology is outside the scope of this thesis, however the point to be made is that modeling the (current and new) work processes plays a central role in the life cycle of the methodology. Each step of the way, decisions about changing the business process are based on, and simulated in, a model. Modeling and simulation takes center stage. Design decisions are made based on statistical analysis of the efficiency (time and cost) of the simulated business process. The SPARKS™ training manual states: "SPARKS™ is [...] a computer-based tool for modeling, simulating and analyzing current business processes, and redesigning and implementing alternatives." (C&L 1994).

There are many uses of workflow modeling and simulation in a BPR project:

- As an analysis tool for the effectiveness of a business process, and as a way to identify opportunities for improvements.
- To describe the redesign of a business process, and to test and evaluate redesign alternatives. The argument is that, if we believe "the numbers" from the simulation are representative of how things work in the real world, we can use the model to choose between various alternatives.
- To analyze the impact of new technology and automation on a business process.
- To communicate, document, and train personnel about a business process.
- To manage and control a business process in real time, by implementing the model with workflow software technology.

2.1.2 Components of a workflow simulation model

Figure 2-2 shows the components of a workflow simulation model. The taskflow⁷ model represents the sequential tasks in a work process. The resource model shows the resources (people and artifacts) that are performing the task. In the input model one specifies *what* is "flowing" through the work process. This is the *work product* that is being worked on. Objects are flowing through the model as components of jobs. A *job* (e.g. an order) represents one or more objects that are worked on during the work process. The term *job* does not refer to the work of a person, as in "what is his job." Instead, a *job* is an abstract concept that

⁷ The concepts *taskflow* and *workflow* are used synonymously. However, I use *taskflow* when I speak of a static representation of a task sequence, and use *workflow* when I speak of a dynamic model that incorporates a taskflow, resource, input and timing model.

represents the work product being worked on. As such, it has discrete entry and exit criteria, and can be seen as the product of the business process. What people in the organization understand as “a job,” depends on their role and the tasks they perform in the organization. Some individuals in the organization view a job as a physical object, whereas for others it means something conceptual. For example, for people working on the job-floor of a manufacturing plant a job is the physical object that they are working on. However, for the people in the sales department, a job is a specific customer order represented by a customer order form.

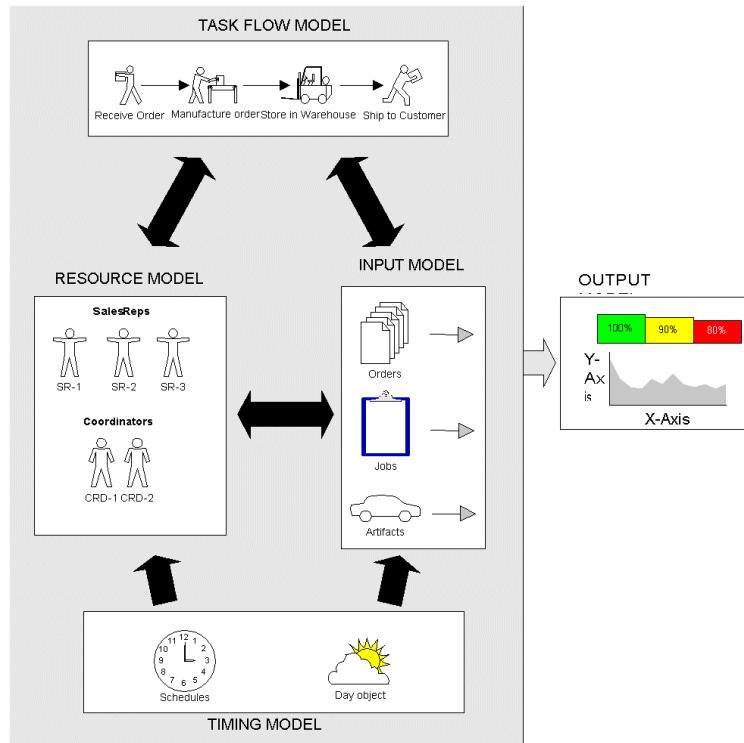


Figure 2-2. Components of a workflow model

A static model is a model that does not change with time, whereas a dynamic model changes over time through a simulation. In a static model the input, timing, resource, and taskflow models are not necessarily connected. To create a simulation, however, the models need to be interconnected. During a simulation, the input model provides the taskflow model with new jobs entering a process. The resource model provides the taskflow model with the resource units needed for each individual task. The timing model exists as part of a simulation, and provides the definition of the days and schedules used for resources, the times that jobs are entering the simulation, and the simulation clock keeping track of the time during the simulation. The output model specifies the *statistics* that are kept during the simulation and can be displayed.

2.1.2.1 Resource Model

The resource model defines the individual *resources* and groups of individual resources. Resources represent people or artifacts that perform work. Example resources include a clerk, a manager, a machinist, a machine, a drill, a computer, et cetera. The work performed by resources is not defined at the individual resource-level, or at the group-level. Instead, the work performed by individual resources is implicitly represented by the assignment of the number of resources used in a specific task in the taskflow model. Groups are only used to track statistics at the aggregate group level. When all the resources in a group are “being consumed” by tasks during a simulation run, the group has an in-box (queue) in which the work is kept until resources are available.

Individual resources in a workflow model do not exhibit individual behavior, neither task nor cognitive. Resources are statistical units assigned to tasks, which simply means that performing the task consumes the amount of resources specified at the task level.

2.1.2.2 Representing work as a taskflow

In a taskflow model work is represented in functional units, called *tasks*, through which jobs flow. Tasks represent the atomic steps of a process. Tasks are functions taking resources and jobs as input and calculating cost output, based on the time it takes to do the task and the cost of the resources for one job. Tasks are chained in a sequential flow, represented from left to right in task sequences. A number of special task-types enable representing work as a complex directed graph. In this section, I will describe some of the representational issues using workflow models.

2.1.2.2.1 Branches: Modeling different possible flows

A *branch* task is used to represent a decision point in a workflow. One specifies, usually in percentages, how many jobs will flow up one branch and how many flow down the other branch(es). *Joins* bring several branches of a flow back together into a single flow. Branch tasks and joins allow a modeler to represent the different ways a job can flow based on some attribute of the job. For example, the jobs (i.e. orders) flowing through the workflow represented in Figure 2-3 have an associated attribute representing the type of order form used for the job. In the branch task, this attribute is examined. Figure 2-3 shows the use of a branch and a join to represent an error condition.⁸

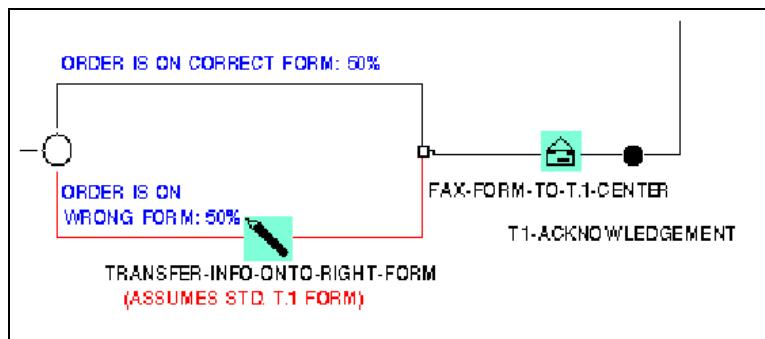


Figure 2-3. Modeling error conditions with a branch and a join task

The example shows an error condition in which the wrong form for an order is used; this happens in 50 percent of the cases. Thus, 50 percent of the jobs in the simulation will flow down through the task TRANSFER-INFO-ONTO-RIGHT-FORM. The two branches come together before the task FAX-FORM-TO-T.1-CENTER. From here on the jobs will again follow the same flow. Branches are “either-or” paths, and a job can only flow down one of the branches.

2.1.2.2.2 Spawns & Merges: Modeling simultaneous work performed by multiple people

Modelers often need to represent a situation where more than one person is working on the same job at the same time. Sometimes, this is done in isolation; for example, two people in different organizations have to perform tasks on the same job independent from each other. When people are working together at the same time modeling becomes more complicated, because their work needs to be synchronized. In workflow models, these work situations are modeled with a special kind of task, called a *spawn task*. Figure 2-4 shows the collaboration between the CO (central office) and the Field organization in troubleshooting and fixing a problem, by branching the workflow using the spawn task “co-helps-field.” The example shows the two separate tasks done by the two separate organizations at the same time. After the collaboration has been completed, the merge task “merge-shoot-trouble” recombines the spawned job.

⁸ This example, and the following examples, are from the T1 Radical Redesign model in SPARKS™, developed by Dave Torok at NYNEX Science & Technology, as part of the T1 Redesign project in 1992.

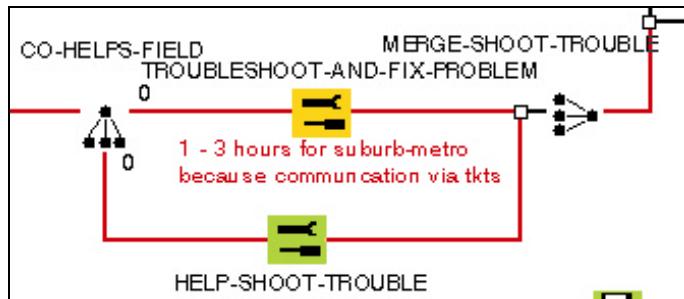


Figure 2-4. Modeling simultaneous work

A spawn task looks like a branch task, but is semantically different, because there is no decision whether the job should go one way or the other. Instead, the job is split—*spawned*—into two jobs, as if the job is flowing through both branches at the same time. Each branch of the spawn is used for representing the tasks of a different resource. At the end of the spawn there is a *merge task*. A merge task *merges* the split jobs together into the job from before the spawn. In this way, the simulation can keep track of the time spent by all resources for both workflows.

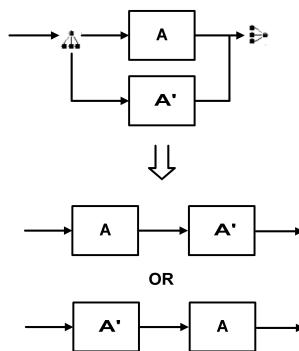


Figure 2-5. Spawn semantics

Note that during a simulation the tasks in the spawned flow are not necessarily performed at the same time. For example, in Figure 2-4 we might think that the tasks “troubleshoot-and-fix-problem” and “help-shoot-trouble” would be performed simultaneously. However, this depends on the availability of resources. Let us assume there are only two resources in the model. The first resource is assigned to the “troubleshoot-and-fix-problem.” The second resource is assigned to the “help-shoot-trouble” task. Let us also assume that at the moment a job is entering the spawn, the second resource is assigned to a task not shown in Figure 2-4. The second part of the spawn-job waits at the “help-shoot-trouble” task until the second resource is available. The second resource becomes available after its current task is completed, and the resource can start working on the “help-shoot-trouble” task. Consequently, the two tasks are performed in sequence (see Figure 2-5). This is not what is conceptually meant with the spawned flow in Figure 2-4. Of course, statistically the amount of time that was worked on the two tasks is correct, because the same amount of resources work on the tasks. However, the overall duration time for the job will now be longer than it should be, because the two tasks are done in sequence instead of in parallel. Remember that conceptually, the model is supposed to represent collaboration between two individuals. It would be better if the simulation showed that if one of the two resources, needed to accomplish the troubleshooting task, is not available, the task does not get accomplished. However, workflow simulations in Sparks™ do not give the modeler control over the assignment of resources to tasks, and availability of resources during a simulation. In short, in a Sparks™ workflow model there it is not guaranteed that two tasks are actually performed in parallel. There is no control over the execution of parallel tasks. This makes it hard to represent collaborative tasks between resources.

2.1.2.2.3 Modeling behavior with delays in the flow

Sometimes work cannot proceed because of some condition that needs to be met, but cannot be met at that moment. For example, when a sales-representative needs to call back a customer because he or she needs more information, and the customer does not answer, the job has to wait until the sales representative can talk to the customer. In workflow models, such a situation is modeled with a *delay task*.

While in a delay task a job is held and no resources are working on it—the resources are released to work on other tasks—while time is continuing. Figure 2-6 shows the example of a delay task, representing a sales rep waiting to call back a customer.

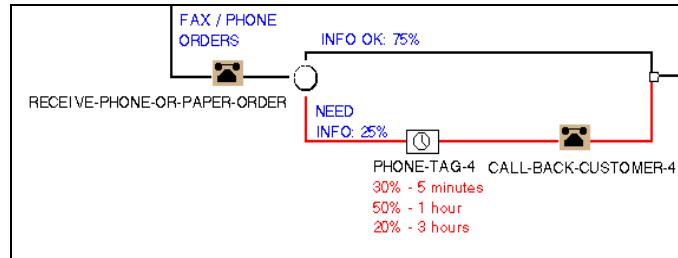


Figure 2-6. Modeling job delay

In order to model the task more accurately, the delay "phone-tag-4" in Figure 2-6 has several delay times associated with it: in 30 percent of the cases the delay takes 5 minutes, in 50 percent, 1 hour, and in 20 percent, 3 hours. Using this distribution of delay times, the model-builder intended to represent the different durations before the telephone call actually takes place. This representation of a phone call using delay tasks does not represent how phone calls happen in the real world. Phone calls are not planned tasks, but are situation dependent. Whether the sales-representative gets a hold of the customer depends on where the customer is and what the customer is doing. It will be very difficult to define a realistic distribution function for the time it takes for the sales-representative to get a hold of the customer, because it depends on the customer's situation. Also, the task of the resource being called (i.e. the customer) is not represented at all. It is not possible to associate the customer resource with the "call-back-customer-4" task, because resources are not tracked with the order, but with the task. During the simulation it is not known which customer resource should be associated with the callback task. This relates to the inherent problem of modeling collaborative tasks in a workflow model. Therefore, the model leaves out which customer is being called, and the time it takes for the customer to answer the phone call and talk to the representative is not taken into account.

Similar representational issues arise in a number of other cases; "off-task" behaviors, such as having coffee, or discussing the win in last night's basketball game (Sachs 1995). How do we represent the delay of a job, because the manager suddenly asks the salesrep for a report on the monthly order figures? In tasks that are unrelated to the flow, the resources are not working on a specific job flowing through the process. These "off-task" behaviors cannot be modeled explicitly by a workflow representation, because the moment of occurrence is not known, and because such tasks do not "touch" the jobs flowing through the model. Work consists of many such off-task behaviors. Being interrupted in the task you are doing and resuming it after some time is part of every day work practice. We can model interruptions and resumes by adding delay tasks, but this is *not* how things happen in the real world. A delay task means that after the previous task there *is always* some delay. Interruptions, such as a phone call, or going to the bathroom, are serendipitous and cannot be modeled by delay-tasks. Not being able to represent this aspect of work means that the workflow simulation can never be an accurate reflection of the *work practice*—what people are actually doing.

2.1.2.3 Input Model

To simulate work using the workflow paradigm there need to be jobs "flowing" through the model. The number of jobs and the time intervals at which these jobs "enter" the model have to be defined up front, and monitored by the simulation engine. In SPARKS™ this is done by a special type of task, called a *start task*. Start tasks input jobs of a particular job-type into the task flow model. They specify the type of the job, the time intervals the jobs are entered into the task flow (job start times), the number of jobs that will be entered (job volume), and the distribution of the number of jobs over the time interval. *Job types* are used to define and categorize different jobs, representing different types of objects processed by the resources in the model. For instance, there might be different types of jobs in a customer service department, such as X-type jobs representing orders from customer X, being processed by the X-group, and Y-type jobs representing orders from customer Y, being processed by the Y-group. By separating jobs into types, the simulation engine can gather statistics for each of the job type.

2.1.2.3.1 Jobs are conceptual

A job is an abstract concept of “what is flowing” through the process. For example, an order form is faxed, after which the information is put in a database. The database information is used by the billing system, and is re-typed into that system. Abstracting different artifacts people are working on during the course of the process creates the notion of a job. The transformation from one artifact to another is what makes things “flow,” although it is arguable that it is not the artifacts that are “flowing,” but the information contained in the artifacts. Consider a piece of paper representing an order form with customer order information. This form is faxed to a department and there the clerk enters the information from the faxed form into the companies order tracking system. The form stops “flowing” through the process, but the order information, now in the computer system, keeps flowing through the process. In most workflow models, the artifacts that make up a job are not represented, and therefore, neither are the changes between artifacts, nor the flow of information. What is represented is a conceptual object representing the job. This limited representation of what constitutes a job makes it difficult to model the context of work. For example, in modeling a “hand-off” we would need to model what is being “handed-off”. The form of the artifact that is being handed-off plays an important role in how people do the work. For example, it makes a difference whether the order form is hand-delivered, or whether the information is sent through Lotus Notes™⁹. When hand-delivered, there could be a two-way conversation around the order. Whatever the impact of this conversation, the point is that this conversation is not represented. In other words, it is abstracted away from the actual work practice in the organization. By abstracting away the media (i.e. artifacts) of a work product, the context of the work is being left out, the information, or wrong information being passed down, is not represented, and neither are the actual work activities of the individuals in the organization. This is a severe limitation in using workflow models to represent how people actually work.

2.1.2.4 Output Model

There are two types of output from simulating a workflow model. The first is showing (using animation) jobs flowing through the workflow. The second, and most emphasized in workflow modeling, is the calculation of a wide range of statistics at various levels of granularity. There are two categories of statistics; statistics for the resources in the model and statistics for the workflow process itself. Most workflow simulation tools allow the end user to customize the output model.

Statistics about the workflow process are used to capture information about the jobs passing through the model. These statistics typically include, the number of jobs passing through the model, the time spent on the tasks, and the cost associated with the process as it is represented. Statistics also include detailed information for each task, such as: the number of jobs arrived at the task, the number of jobs finished by the task, and the actual and average task times during the simulation run.

Resource statistics are used to capture information about resource utilization and costs. There are statistics at the individual resource level, as well as at the accumulated group levels. Statistics of this kind include utilization of the resource, including percentages, availability of the resources throughout the simulation, and the work loads (i.e. backlogs) of the resources.

2.1.2.4.1 Issues with statistics

The statistical output of a workflow model is completely dependent on the resource and task data for the model. This phenomenon is known as the *garbage in, garbage out* phenomenon (Banks et al. 1996). Because a workflow model is so dependent on the data used to define the cost and time estimates for resources and tasks, this problem is real and almost unavoidable, as is illustrated by the statement of the model-builder of the T1-redesign model:

I must put forth a CAUTION statement here! BE SURE that you release statistics to the world wisely and with the caveats about the meaning of specific SPARKS data. There are so many assumptions built into the model that affect the numbers' applicability to the real world. The best use of the numbers is as a relative measure between one SPARKS model of the T.1 process and another model. However, there is more and more pressure to use the numbers in an absolute sense, so do so wisely. (Torok 1992)

⁹ Lotus Notes is a trademark of IBM Corporation.

The modeler found this warning so important that he repeated it twice in his document about the T.1 SPARKS™ model.

Because of the emphasis on the dynamics of jobs in workflow models, instead of the dynamics (i.e. behavior) of resources, and because of the limitation in modeling the way people and groups actually work, most resource statistics are, therefore, inherently inaccurate. Unfortunately, most of the time there is strong pressure from upper levels of management to show "head count" across models, and to map any kind of cost savings into head count reduction (Davenport 1995). One has to be careful in basing management decisions on simulation model statistics. Data and model verification and validation is one of the most important, but at the same time most difficult aspects of simulations (Banks et al. 1996) (Zeigler et al. 2000). The goal of the validation process is twofold: 1) produce a model that represents the actual system close enough so that it can be used as a substitute, and 2) develop a credible model so that it can be used with confidence in decision making.

I do not want to claim that statistics are completely useless. However, there are so many data related issues around the calculation of statistics that the use of a workflow simulation model in a process redesign project should always be looked upon with a dose of healthy skepticism.

2.1.3 Problems with workflow

"Work" consists of more than a functional transformation of work products by resources (Sachs 1995; Suchman 1987). One just has to think about the informal, circumstantial factors that influence work quality. Think about the effects of collaboration, types of hand-offs, geographical location of the people, the willingness of people, and sometimes organizations, to work or not work together ("engineering believes that they own the company"). All this is missing from a functional view of the work. In this section, I discuss some of the problems we have identified using workflow modeling and simulation in work redesign projects at the former NYNEX telephone company. I draw from our experiences using the SPARKS™ environment to model the current work process and the redesigned work process during two process redesign projects at the former NYNEX telephone company (the T1 re-design project (Corcoran 1992), and the BNA re-design project).

2.1.3.1 Data acquisition

It is very difficult to gather statistically significant and/or valid data for the development of a workflow model. This is true for the design of a future work process, but it is even true for the development of a workflow model of a *current* work process. Often, the work process organization does not keep historical data of the process. This means that the process analysts will have to gather the time and cost data for job-flow and resources. To do this correctly is not an easy task. When there is no data already available one should question why this is the case. Most of the time you will find that this is because it is not easy to gather this type of data. To do an extensive statistical data analysis of the work process might take longer than the life of the project itself. One approach is to do "spot observations" for jobs, tasks and resources. The data that one collects with this approach is, most likely, *not statistically valid*, meaning it is:

1. Not collected from the correct population,
2. Collected from incorrect or not representative observations,
3. Collected from an unreliable source.

2.1.3.2 Biases in simulation data

When analyzing simulation data, it is important to understand the types of biases that may be incorporated in the data from a simulation run (Banks et al. 1996). The two biases that are apparent are the "snap shot" bias and the "average vs. bucket" bias.

The *snapshot bias* occurs when, during the simulation, the data is examined at a "snapshot" in time. The bias occurs when comparing data between two measurement points in the workflow model, such as looking

at the difference in average elapsed time. The data is biased, because at any given time the data measured at the latter point in the flow does not include the data from jobs that are, at that moment, in between the first and the latter point, and therefore have already past the first measured point, but not the second. In other words, the data measured at the second point is "lighter" than the data measured at the first point. This bias increases when either the elapsed time between the points increases, or the number of jobs measured decreases. Therefore, to gain truly accurate difference measures, the simulation should be run for a large number of jobs.

The *average vs. bucket bias* occurs when the averaged data does not represent the actual distribution at a given point. An excellent example of this is shown in the T.1 model for the average elapsed time at the completion of the "order-negotiation" process. The average elapsed time was approximately five hours. However, a look at the distribution of elapsed times showed 1378 of 1679 orders took under 30 minutes, and 301 orders took 24 hours or longer, with absolutely no orders completed between 30 minutes and 24 hours! This type of bias can occur with any average measure (cost, time, etc.) and is highly dependent on the actual work practice that is modeled. This bias can also occur when mixing jobs of different types.

2.1.3.3 Resource issues

The workload in a workflow simulation is dependent on the resources available to do the work. For a workflow simulator, resources are all the same. The only use of a resource is to balance the workload; i.e. the more resources that can be chosen from for a particular job in a task, the more jobs can be done in parallel. The type of resource has no influence on the time it takes to finish a task—some tools, such as SPARKS™ try to model personal, social and cognitive factors of resources as numerical measures. However, these measures are very subjective and unscientific, and can do more harm than good (see 2.1.3.3.3). The use of types of resources in a model is therefore mostly for the gathering of cumulative statistics at the group level. For example, the cost of one type of resource is higher than others.

2.1.3.3.1 Motivation and culture

We all know intuitively that experience levels of individuals, as well as culture and work ethics, have impact on the work process. In addition, we know intuitively that some groups are more adaptable to change than others. So, what happens to the performance of groups when their work is changed? More importantly, can we design a new work process with these kinds of factors in mind? These are important issues in the redesign of a work process. Workflow models do not include these types of factors. The business community starts to realize that the *intellectual capital* of a company consists of the people, and their work practices (Stewart 1997) (Nonaka and Takeuchi 1995). Modeling and simulating the work of an organization without taking the capabilities (cognitive, social, and cultural) of the individuals into account leads to work process designs that most likely fail to produce the expected results after implementation. In short, the change from the current to the future situation impacts the work at a deeper level that cannot be predicted by a model of the future work system.

2.1.3.3.2 Resource approximations

One dilemma in modeling the resources in a workflow model is the dilemma of how accurate to model the resources of an organization. The tasks modeled in a taskflow model are usually just part of the total work of an organization. We referred to the work *not* modeled in a workflow model as "off-task" behaviors. For example, the taskflow of the T1 model only represented a part of all the jobs that were being worked on by the different organizations in the model. Therefore, if the resources would be modeled very accurately (i.e. the number of people working, etc), there would always be enough resources to handle the job-load in the simulation model. So, how many resources is a realistic measure? The modeler for the T1 model "compressed" the resource model to an approximate number of resources needed to work "full time" on T1 provisioning jobs. Again, these kinds of approximations make the output of a workflow simulation less valid.

2.1.3.3.3 Modeling social, cultural and cognitive factors

The way people work is very much defined by factors other than time and money. For one thing, people build relationships in the workplace that influence the way they work. (Suchman 1987) (Wenger 1997). The point is that work is not just the flow of jobs through the process, or the flow of information between people and systems. Work is about people, their habits, their norms and relationships, their physical environment, et cetera. A new design of a work process that is going to be a success when implemented has to consider these factors. Efficiency in the workflow does not necessarily come by removing some redundant tasks.

Sure, it might help, but maybe the tasks are there for a good reason, and taking them out might actually hurt the process.

Some modeling tools try to model social, cultural and cognitive factors using quantitative measures. For example, in SPARKS™ we can model the “diligence” of a resource by specifying in seconds how long the resource will work on a task *after* the time model specifies that it stops working for the day. This way, a resource that just started a task that takes thirty minutes, at five minutes to five, and who will normally stop working at five o’clock and has a diligence factor of 30 minutes, will finish the task before stopping. Although this may sound like a good way to model the flexibility of people, it is not a very realistic way of modeling flexibility. It means, for instance, that if the task would take 31 minutes the resource would not be diligent enough and stop. How does someone know exactly, to the minute, how long a task will take? This is not realistic and not how people do work, even if they have to “punch the clock.”

2.1.3.3.4 Multi-tasking: interrupt and resume

Work consists of interruptions, switching to and from different activities; picking up a phone while checking the status of a job, answering someone’s question while entering data in the order system, et cetera. It is hardly ever that we are engaged in just one task at a time. People are masters in “juggling” many tasks at once. However, representing the multi-tasking of individual resources in a workflow model is not possible. Resources cannot be controlled at an individual level. The workflow simulator schedules the resources. Which resource does what and when depends on the number of resources available and the workload (the number of jobs flowing through the simulation), and not on a model of the cognitive behavior of the resource.

2.1.3.4 The Turf Coordinator problem

The TC role was briefly mentioned in the introduction chapter, and is described in more detail in Sachs (Sachs 1995). The problem of representing the work associated with this new role in a workflow model was one of the driving forces to start our research on a new modeling paradigm for simulating work practice. In this section, we describe the problem in more detail.

In the radical new design of the T1 process, the design team came up with a new concept called a “turf.” The city of Manhattan was divided in a number of geographical turfs. Each turf has their own field force that covers that specific geographical area and the central offices in that area. In the redesign model, the T1 jobs are assigned to a specific turf. Consequently, the field force is now dedicated to a turf. In the model, this meant that jobs, as well as the tasks and resources had to be duplicated, making the model more complex and more difficult to maintain¹⁰. The next big change in the design was the introduction of the TC role. For each turf, there is a TC. A TC has end-to-end responsibility for a job. The work associated with this role is not what a business manager would call “value added” work. The tasks of a TC are not directly related to the flow of jobs through the work process. His or her tasks do not transform the job, but are mostly related to the activity of coordinating conversations between the field and the central office. Therefore, most of the tasks of the TC have no place in a workflow model. This is best explained with the example of coordinating the testing of a T.1 circuit.

2.1.3.4.1 Coordinating a T.1 circuit test

One of the identified problems in the old (current) T1 work process was the coordination of testing circuits between the central office and the field (the customer premise). Before the concept of a turf and the TC role, nobody in the process was specifically assigned to a job. The Trouble Ticketing System (TTS) controlled the whole work process. The TTS was designed to eliminate what management felt were “off-task” conversations between workers. The TTS was setup with the assumption that *any* worker can perform a specific task for a job, and that the installation, testing and completion of a job did not have to be done by one team of workers. It was believed that it is more efficient to break the work up and have people work on one specific task in a job. This way, orders could be handled as they come up, and people wouldn’t be caught in time-consuming troubleshooting. The process worked by sending and receiving “tickets.” Each ticket would be dispatched by a central dispatch system. Every time someone had performed a task from a ticket, the ticket was sent back to the central dispatch system, which would then send it to another person,

¹⁰ In SPARKS™ each task can only be worked on by one resource, selected from one group. Therefore, modeling seven turfs entailed creating seven identical process flows. Just as in computer programming, duplication of “code” creates a maintenance overhead.

leaving the first person to pick up a new ticket. In practice, this meant that when a worker encountered a problem (like not hearing a dial-tone) (s)he would send a trouble-ticket into TTS, and assume that someone else would pick up this trouble and fix it. The person who sent the ticket in the first place would then have time to pick up the next ticket and start working on a new job. From the worker's perspective, the electronic dispatching got in the way of being able to solve problems collaboratively. Unlike problem-solving conversations in which two people can discuss a problem and explore the possibilities to solve it, the TTS transforms a problem-solving conversation into a number of sequential tickets picked up by an array of workers, who do not speak to one another.

The translation of each turn of talk into a single ticket reduced an effective network of co-workers who could troubleshoot together into something like a relay-race, handing-off pieces of work to the next runner, creating an aggregate of dissociated workers. It changed a troubleshooting conversation into a series of solitary commands disembodied from context. Not only was the conversation—the story line of the problem, if you will—lost, so was the work community. (No one knew who else was working on the job.) (Sachs 1995)

Therefore, whenever the technician, installing the T1 line in the field (at the customer premise), had to run a test to verify the complete circuit from end to end, (s)he had to send a trouble-ticket to the central office to request the test. The tester who would pick-up the ticket would setup the circuit so that the test could be run. There was no direct contact between the tester, and the field technician. When the test would go wrong, (s)he would have to go through the whole process again, while the second time the ticket might be picked up by a different tester.

In the new design, the TC is an experienced technician whose role is to coordinate the telephone communications between, up to five, field technicians installing and testing a circuit. The problems that can occur when installing and testing a circuit are unpredictable and of a wide range. The TC needs to know whom to contact, and where people are geographically located in order to coordinate getting the right people involved. The TC, field technicians, and turf managers meet regularly to identify recurring problems and solve them.

2.1.3.4.2 Modeling the turf coordinator

The difficulty with modeling the TC in a workflow model has to do with the fact that the work activities of a TC are not necessarily identifiable with specific jobs flowing through the process. True, when a TC coordinates communication between several people in the field due to a problem that occurred, (s)he is working on a job, and the task can be seen as such. However, *when this happens* is unpredictable and cannot be pre-specified in a task sequence. Furthermore, representing a three or more person phone conference is not easy, if not impossible, in a workflow model. Representing the coordination of such a conference call is so situated that it is even hard to think about pre-specifying the steps that are involved. As mentioned before, in a workflow model we cannot specify a specific resource working on a specific job. Therefore, the only way such a conference call would be possible is if there are enough resources available at that moment in the simulation to work on that specific job. The chances that the right resource, from the right location, performs the correct task in the task sequence, is next to null. Modeling a phone conference call as a pre-specified sequence of tasks is impossible, since it is unpredictable how such collaboration might play out over time.

All this resulted in the fact that the TC's work hardly showed up in the SPARKS™ model of the radical new design. Because of that, it was hard for the design team to justify a full-time person playing the role of a TC. The resource statistics of the model did not justify a full-time person. Nonetheless, the design team knew that the importance of coordinating the technicians was one of the major factors in the time and cost savings of the new design. Although management was very impressed with the time and cost savings of the model, the team had a hard time convincing management of the need for a TC.

This experience led us to start working on a modeling paradigm and modeling tool that would allow the representation of coordination activities, such as the TC. Work needs to be represented based on the activities that individuals engage in, and the flexibility of people's decisions to change the sequence of these activities. Collaboration cannot be pre-specified, but is emergent. The next section discusses the main

difference between the emergence of behavior versus the pre-specification of the sequence of tasks, possible error situation, and the collaboration between individuals in a work process.

2.1.3.5 Emergent behavior vs. static taskflows¹¹

As I have indicated, workflow models typically describe only idealized tasks in transformation of a work product. In the following illustration (Figure 2-7), for example, an engineer receives an order form from a representative, assigns a circuit loop using a computer tool; later the representative enters more data about the order. Figure 2-7 presents an excerpt of a SPARKS™ model for Business Network Architecture (BNA) order processing¹².

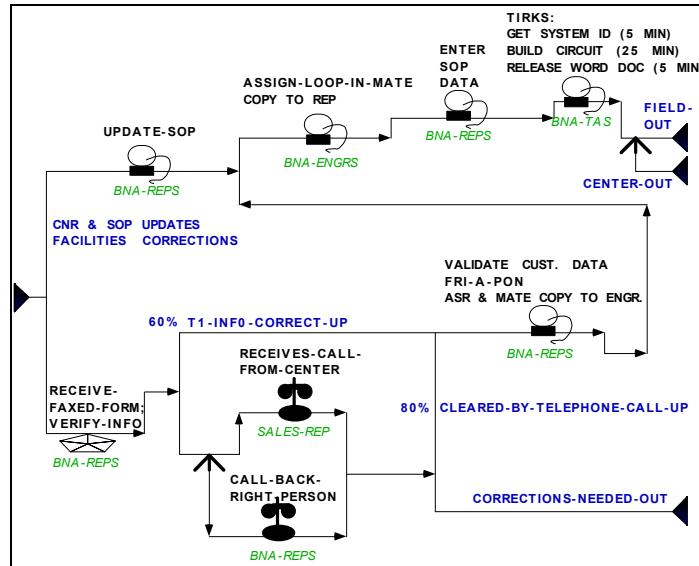


Figure 2-7. Order processing in the business network architecture (BNA) organization, showing flow of orders from left to right and conditional branching (indicated by arrows hitting a vertical bar). Top section shows updates by representatives (BNA-reps) and engineers for customer-not-ready (CNR) and other revisions to orders. Lower section shows standard process for handling faxed order from sales center, followed by correcting 40% with missing or invalid information by calling the sales representative. After validating customer data (center right), orders are handled by circuit allocation process (top). Other acronyms (e.g., SOP) are internal databases.

A critique of this diagram from the perspective of situated action (Suchman 1987) would inquire why order processing occurs this way and how it might be improved. Perhaps surprisingly, the figure leaves out what a problem-solving (cognitive task) model would typically focus upon. For example, what information does the engineer read from the order form and what deductions are required in order to assign the circuit? This particular model leaves out *how orders are planned and assigned*, *multi-tasking* (the fact that a rep or engineer works on several jobs at once before completing them) and *how people interrupt and resume their work*. A cognitive model of the same business process might consider some of these factors, but would leave out how people come to be synchronized in a phone conversation, how an engineer might help a representative do his job, and broader considerations of how a representative actually spends her day. In particular, because interpreting and executing orders can be problematic in unexpected ways, people need to improvise in ways that work system designers might not have anticipated:

Information flow charts show "information" moving in little blocks or triangles moving along arrows to encounter specific transformations and directions along the diagram. In reality, it seems, all along the arrows as well as at the nodes, that there are people helping this block to be what it needs to be—to name it, to put it under the heading where it will be seen as a recognizable variant, deciding whether to leave it in or take it out, whom to convey it to. (Wynn 1991)

¹¹ This section is adapted from Clancey, W. J., Sachs, P., Sierhuis, M., and van Hoof, R. (1998). "Brahms: Simulating practice for work systems design." *International Journal on Human-Computer Studies*, 49:831-865.

¹² The BNA project was another business process redesign project within the, by then, Bell Atlantic company.

Wynn's complaint might be viewed as an issue of modeling granularity—she is asking for more details. But her broader issue is how people think about work and how they solve problems cannot be reduced to information processing tasks and reasoning. Additional concepts are required.

To analyze the example more precisely, consider what the various branches and joins mean:

- Proportional mix of different kinds of orders, customers, or services (e.g., the first branch indicates how the order comes into the organization, as an update/correction or as an initial faxed order).
- Hand-off to the next (possibly dependent) step in a functional sequence (especially clear in the modeler's use of a step notation in the top portion of Figure 2-7).
- Condition of job being processed (e.g., incorrect information, indicated by a branch showing 60% correct and 40% requiring troubleshooting).
- Events that occur during troubleshooting (e.g., receive-a-call or keep-calling-back).

Although this abstraction is useful, notice that everything in this diagram was specified and connected by the modeler. The model essentially leaves out the logistics, *how these conditions come to be detected and resolved*, such that work and information actually flows. What is wanted is a model that includes aspects of reasoning found in an information-processing model, plus aspects of geography, agent movement, and physical changes to the environment found in a multiagent simulation (Tokoro 1996). The designed flow of Figure 2-7 assumes that people are always on the spot, picking up faxes and handing them over to others, reviewing the status of database entries on-line, responding to phone calls, et cetera. In reality such behavior is an emergent property of the situation; people come together and their work is constrained by the environment. A designed work and information flow diagram leaves out the accomplishment of *synchronization* and the effect of *juxtaposition* of materials, such as the following:

- Parallel-dependent processing (e.g., in practice, people start a time-consuming step in processing order before approval/clearance for doing the work (Dourish 1996))
- Cognitive interpretation, social knowledge, and variability (e.g., how do people know that information is not correct? How are intra-group variations in how the work is done a resource for handling difficult situations, such as the unavailability of a computer system?)
- Interactional logistics and daily activities (e.g., the steps marked "Receives call" and "Call-back-right person" in Figure 2-7 omit the activity of "making the call" in the first place and when it occurs during the day. Is a pager and cellular phone used or voice mail at a desk phone?)
- Informal help and "keeping an eye" on the work (e.g., stepping outside defined roles, especially being concerned about the end result even after doing one's own step in a process).

By ignoring the movement and transformation of information through human action, especially conversation, a designed workflow not only fails to explain how flows actually can happen at all, but leaves out the emergent effects of *serendipity*, such as stumbling on one order while looking for another or bumping into someone in the hall and learning about a new organizational priority.

2.1.4 Discussion

In this section, I discussed workflow modeling and simulation as a tool for business process re-design projects. Although the purpose of a workflow model is to represent the work of people in a work process, I have shown the limitations of this paradigm to model and simulate work as it really happens. I discussed the problems surrounding the validity of the statistics generated by a workflow simulation. Furthermore, we discussed many of the other limitations in representing how people actually work. A workflow model focuses on the sequential movement of jobs through process steps; therefore, the paradigm is very limited in representing the work from the point of view of a worker. People are seen as statistical resources. The representation of the flexible behavior of humans is not possible, and makes workflow models not a realistic

representation of how people work. This problem was emphasized by the example of trying to model the TC role in the T1 re-design project at NYNEX. Although the work activities of a TC were central to the work process, a workflow model was not able to show the “off-task” coordination work of the TC. Therefore, the model was a bad representation of the new work design. A simulation of work practice should emphasize:

- What people actually do, not just official job functions;
- What people are doing every minute of the day, where they are, and what they are perceiving, not just working on one task at a time;
- The collaboration between two or more agents, such as face-to-face conversations, telephone calls, etc, not just communication as a stochastic event;
- That people have personal identity, and are not interchangeable resources.

I conclude this review of the workflow model-based paradigm showing in Table 2-1 its limitations to represent people's collaboration, “off-task” behaviors, multi-tasking, interrupted and resumed activities, informal interaction, knowledge and geography.

In the next sections, I describe several agent-based modeling approaches for modeling individual intelligent agent behavior. If we want to model the work of a group of individuals, it is obvious that an agent-based approach is warranted.

Table 2-1. Workflow modeling limitations

	Collaboration	Off-task behaviors	Multi-tasking	Interrupt and resume	Informal interactions	Cognitive behavior	Geography (location)
Task Model	Tasks <i>cannot</i> represent collaboration between people	Tasks that do <i>not</i> directly "touch" jobs flowing <i>cannot</i> be represented	Not supported	Tasks <i>cannot</i> be interrupted and resumed	Tasks <i>cannot</i> represent informal interaction between people	Not supported	Tasks are <i>not</i> associated with location
Input Model	N/A	Jobs <i>have to</i> "flow through" tasks to be worked on	N/A	N/A	N/A	N/A	Jobs are <i>not</i> represented as located
Resource Model	Resource interactions <i>cannot</i> be represented	Resources <i>cannot</i> be associated with tasks outside of the workflow	Resources <i>cannot</i> work on more than one task at the same time	Resources <i>cannot</i> be interrupted when working on a task	Resources <i>cannot</i> participate in informal interactions between other resources	Resources are numeric, <i>not</i> symbolic entities. Resources <i>do not</i> have cognitive capabilities	Resources are <i>not</i> located
Timing Model	Timing and coordination of tasks of different resources is <i>not</i> possible	N/A	N/A	When started, tasks take a fixed amount of time, and <i>cannot</i> be interrupted and resumed	N/A	Resources have <i>no</i> notion of time	There is only one timing model, and therefore we <i>cannot</i> represent multiple time zones based on location

2.2 COGNITIVE MODELING

Ever since modern cognitive psychology took form in the 1950s and 1960s, it has focused on aspects of understanding human cognition. In the early 1970s, it was Allen Newell who started to work on a unified theory of cognition that would address all aspects of cognition. Newell felt that the only way cognitive psychology would ever come to a unified theory, it needed to understand from the beginning how all the different pieces fit together (Newell 1973b). Newell, at the same time, introduced his answer to this dilemma. He described his first production system theory of human cognition. It was a single system that was able to perform a diverse set of tasks that occupied cognitive psychology. He described production systems as follows (Newell 1973a, pp. 463-464):

A production system is a scheme for specifying an information processing system. It consists of a set of productions, each production consisting of a condition and an action. It has also a collection of data structures: expressions that encode information upon which the production system works—on which the actions operate and on which the conditions can be determined to be true or false.

A production system, starting with an initially given set of data structures, operates as follows. That production whose condition is true of the current data (assume there is only one) is executed, that is, the action is taken. The result is to modify the current data structures. This leads in the next instant to another (possibly the same) production being executed, leading to still further modifications. So it goes, action after action being taken to carry out an entire program of processing, each evoked by its conditions becoming true of the momentarily current collection of data structures. The entire process halts either when no condition is true (hence nothing is evoked) or when an action containing a stop operation occurs.

Much remains to be specified in the above scheme to yield a definite information processing system. What happens (a likely occurrence) if more than one production is satisfied at once? What is the actual scheme for encoding information? What sort of collection of data structures constitutes the current state of knowledge on which the system works? What sort of tests are expressible in the condition of productions? What sort of primitive operations are performable on the data and what collections of these are expressible in the actions of productions? What sort of additional memories are available and how are they accessed and written into? How is the production system itself modified from within, or is this possible? How much time (or effort) is taken by the various components of the system and how do they combine to yield a total time for an entire process.

Over the years, Newell explored a number of variations on his production system concept, concluding with his Soar theory of human cognition (Newell 1990). Right now, there are at least four current and active production system theories: Soar (Newell 1990), ACT-R (Anderson 1993), 3CAPS (Just and Carpenter 1992), EPIC (Meyer and Kieras 1997). All these computational cognitive modeling systems are developed as implementations of a theory of cognition. As such, domain specific models of problem-solving tasks that are developed in these systems are seen as theoretically valid models of how humans perform problem solving. In this chapter, I briefly describe the Soar and ACT-R systems, in order to give a brief overview of the field of cognitive modeling. The reason for not describing all four systems, mentioned above is, a) because Soar and ACT-R are the best known and mostly used production systems for cognitive modeling, and b) to limit the space used to describe the nature of cognitive modeling.

2.2.1 Soar

Soar is a general cognitive architecture for developing systems that exhibit problem-solving behavior. Researchers all over the world, both from the fields of artificial intelligence and cognitive science, are using Soar for a variety of tasks. It has been in use since 1983.

Soar attempts to approximate rational behavior, using the following guiding design principles (Laird et al. 1999):

- The number of distinct architectural mechanisms should be minimized. Soar has a single framework for all tasks and subtasks (problem spaces), a single representation of permanent knowledge (productions), a single representation of temporary knowledge (objects with attributes and values), a single mechanism for generating goals (automatic subgoaling), and a single learning mechanism (chunking).
- All decisions are made through the application of relevant knowledge at run-time. In Soar, every decision is based on the current interpretation of sensory data, the contents of working memory created by prior problem solving, and any relevant knowledge retrieved from permanent memory.

Soar is based on the general hypothesis that *all* goal-oriented behavior can be viewed as the selection and application of operators to a state. A *state* represents the current problem-solving situation in memory, at a specific moment in the problem-solving process. The application of an *operator* changes the current state to a new state, which means that it is changing the representation of the state in memory. A *goal* (or subgoal) is seen as the desired outcome of an operator. Trying to reach its goals, Soar continually applies operators selected to achieve a goal. When an operator succeeds or fails, Soar selects the next operator for a subgoal of the current goal, or for a new goal. Reaching a goal can be seen as having the system reach a goal-state, i.e. the desired representation of objects, attributes, and values.

Soar has two types of memories for storing the different representational objects of a problem-solving process,

- *Working memory* contains descriptions of the current situation—the current state—including data from sensors, results of intermediate production firings, active goals and operators. Working memory is organized into objects. Objects are described in terms of attributes. The current state is the total of objects in working memory with their current attribute-values.
- *Long-term (or production) memory* contains *productions* that define how to react to the objects and their attributes in working memory. The productions in long-term memory can be thought of as the Soar program.

A Soar program contains the knowledge that is relevant in a particular problem-solving task (or a set of tasks), including the knowledge about when to select and apply operators to transform the states of working memory in order to achieve its goals.

2.2.1.1 Problem solving in Soar

Soar's long-term knowledge is organized around the functions of selecting and applying operators. These inherent Soar functions are performed using five distinct types of knowledge, *operator proposal*, *operator comparison*, *operator selection*, and *operator application*. Soar also has generic knowledge about making monotonic inferences about the state (i.e. *state elaboration*). Inferences that result in state changes in working memory have an indirect effect on the operator selection and application functions, because new state descriptions can cue new operators.

The knowledge used in the selection and application of operators is represented in terms of *production rules* encoded in the Soar program. Production rules are declarative "if-then" statements. The if-part of the rule is called the *condition* or *precondition*, and the then-part is called the *action* or *consequence*. The execution of production rules is referred to as *rule firing*. A rule *matches* when its conditions are met based on the current state in working memory. At that moment, the rule fires and its actions are executed. Executing actions means changing working memory (see Figure 2-8).

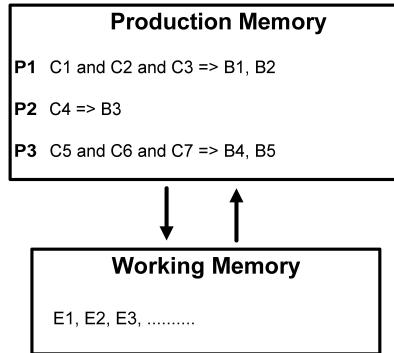


Figure 2-8. The Soar production system (borrowed from (Newell 1990))

Thus, by continuously testing conditions of the rules in long-term memory, matching the rules, based on the current state, the current-state of the system changes continuously. This cycle continues until there is a kind of equilibrium state between long-term and working memory. This means that the current state does not result in firing any of the rules in long-term memory. When this happens, and a specific goal-state has not yet been reached (i.e. the operator has not been fully applied), Soar is unable to make further progress and reaches an *impasse*. There are three types of impasses possible:

1. No operator can be selected, because none is proposed.
2. No single operator can be selected, because there are multiple operators possible and there is not enough knowledge to distinguish between them.
3. An operator is selected, but there is no additional knowledge to apply the operator.

When Soar is in an impasse, the architecture creates a *substate* (i.e. a sub search-space) in which a new operator selection-apply cycle can be started, with as the subgoal to resolve the impasse.

2.2.1.2 Learning in Soar

Learning is an important part of a theory of cognition. There are many aspects of how learning occurs in human cognition that are not well understood today. However, Soar has a primitive notion of learning called *chunking*. Chunking is a form of learning from experience. It is a way to transform specific problem-solving scenarios into productions stored into long-term memory for future use.

A chunk is a newly created production rule. The conditions of the chunk are parameterized working memory elements (WME) of a state that lead (through some chain of production firings) to a specific impasse resolution. The action of the chunk is the WME that actually resolved the impasse. In other words, chaining backwards over the specific WME's that were used to resolve a specific impasse creates the chunk. The first WME in this backward-chaining process is the action of the chunk being created. By parameterizing (replacing objects by variables) the chunk is made more generically applicable as a production rule in long-term memory.

2.2.2 Act-R

ACT-R is the result of long cognitive science research at Carnegie Mellon University, started in the mid-seventies with the introduction of the ACT production system (Anderson 1976). Like Soar, ACT-R is a theory of cognition trying to deal with the empirical knowledge of human cognition that has evolved in cognitive science. ACT-R consists of a theory of the nature of human knowledge, a theory of how this knowledge is deployed, and a theory of how this knowledge is acquired (Anderson 1976). ACT-R is also, like Soar, a computer system that implements the ACT-R theory. ACT-R can be used to develop computer simulations of a wide range of cognitive phenomena in memory, problem solving and skill acquisition.

The ACT-R theory assumes that there are two types of knowledge, declarative and procedural. *Declarative knowledge* is knowledge that we are aware of and can usually communicate to others. It is sometimes

referred to as *factual knowledge* or *axioms*. Examples include “George W. Bush is the 43rd president-elect of the United States,” and “one plus two is three.” *Procedural knowledge* is knowledge that we are not conscious of, and is applied in our problem solving behavior using our declarative knowledge.

Declarative knowledge is represented as *chunks*. Chunks in ACT-R are different from the notion of chunks in Soar. In ACT-R, chunks are WME that represent the factual statements in working memory. For example, the fact $1 + 2 = 3$ is represented as the following chunk (Figure 2-9):

Fact1+2
isa ADDITION-FACT
addend1 One
addend2 Two
sum Three

Figure 2-9. Representation of an ACT-R chunk

Procedural knowledge in ACT-R, just as in Soar, is represented using *production rules*. A production rule is a condition-action pair. The condition specifies what must be true for the rule to apply, and the action specifies a set of things to do if the rule applies. Conditions in a production rule test for the state of the current goal and chunks in declarative memory, whereas the actions change the goal state.

From this brief description of the ACT-R architecture, we can infer that there are many similarities with the Soar architecture; ACT-R’s declarative and procedural memory is synonymous with Soar’s short-term and long-term memory, where its declarative and procedural knowledge is similar to Soar’s objects in working memory and production rules in its long-term memory. The third comparison that holds, although should not be seen as a one-to-one comparison, is the similarity between *goals* in ACT-R, and *operators* in Soar. Goals in ACT-R are held in a separate memory structure, namely the *goal-stack*. In ACT-R, goal structures provide a higher-level organization to control the execution of production rules (relatively similar to operators in Soar).

2.2.2.1 Problem solving in ACT-R

Problem solving is organized through the *current goal*, which represents the focus of attention at each step in the problem solving procedure. ACT-R is always trying to achieve the goal that is at the top of the goal-stack. The current goal is pushed onto the stack, and is the next goal to be achieved. When it is done achieving a goal it pops the goal off the stack, which means that it will start achieving the next goal on the stack (see Figure 2-10). Thus, the goal-stack encodes the hierarchy of intentions that guide the problem-solving behavior.

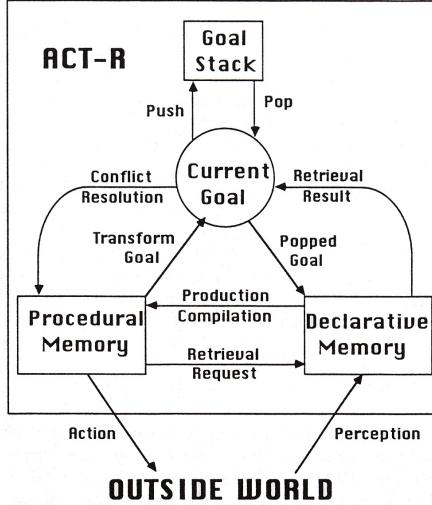


Figure 2-10. The ACT-R production system (borrowed from (Anderson and Lebiere 1998))

To select a production rule to achieve the current goal, there is a *conflict resolution* process that chooses one production from among the productions that match the current goal. The selected production is executed and can result in a transformation of the current goal, possibly resulting in pushing subgoals on the goal stack, or popping the current goal of the stack. It can also result in retrieval requests to declarative memory, which can be returned to the current goal satisfying it, thus causing popping the current-goal. Soar's type of chunking (i.e. production learning) is performed through a process called *production compilation*. Last, execution of a production can result in *actions* to be taken in the outside world, while *perception* of facts in the outside world can independently create chunks in the declarative memory. This last piece is the subject of the next section.

2.2.2.2 The total cognitive system

Newell describes a larger system that includes *perception* and *motor* behavior as the total cognitive system, shown in Figure 2-11 (Newell 1990, pp. 194-195). Here is where we identify a significant difference between the ACT-R and Soar systems implemented in software. Although, both theories describe the need for a total cognitive system, only ACT-R has implemented such a total system.

ACT-R has a number of extensions that have been created by different researchers. First, there is a *visual interface* that incorporates ideas on visual attention and perception. This extension implements in software the capability of accessing information from a computer screen and dealing with a keyboard and mouse, in a similar way human subjects do in psychological experiments. It parses “screens” as humans do and enters key-presses and mouse gestures. The data record that is created using this interface is indistinguishable from the data record that human subjects create. The visual interface is extended more generally to also deal with *audition*, *speech*, and other *hand gestures*.

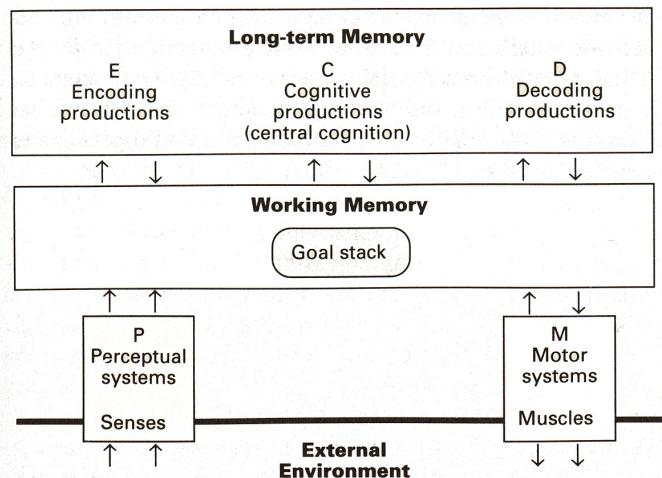


Figure 2-11. The total cognitive system (borrowed from (Newell 1990))

2.2.2.3 Human performance models

The grounding of ACT-R in real experimental data has created the ability for developing models whose correspondence to actual cognition can be validated with a subject performing similar experiments in the real world. This, in some sense, has created a level of reality in cognitive modeling. ACT-R makes predictions about every aspect of the empirical phenomena that are being simulated. For example, in simulating memory experiments the ACT-R model predicts not only the correctness of response choices and response latency, but also the steps involved in the problem solving process.

The use of this is mostly relevant in the field of human factors, where the focus of research is on understanding the impact of a particular interface design on human performance.

2.2.3 Discussion

The home page of the ACT group¹³ at Carnegie Mellon University states: "The architecture takes the form of a computer simulation that is capable of performing and learning from the *same tasks* worked on by human subjects in our laboratories." This says that ACT-R's main focus is to simulate psychological laboratory experiments. The reason for this is the following. Using simulation models of psychological experiments that can be done in a laboratory setting with human subjects, the simulation models can be validated by comparing the simulation output data with data from these real-life laboratory experiments.

Thus, the sole purpose of implementing the ACT-R theory into the ACT-R computer simulation system is to verify and validate the theory. In other words, the ACT group's method for researching unified theories of cognition is through *modeling and simulation*. I make this point, because of two reasons; 1) I apply a similar research method for developing a theory of work practice, and 2) it helps me to show the limitations of ACT-R (and Soar, for that matter) in applying it for a different research problem, namely the development of a theory of work practice.

¹³ <http://act.psy.cmu.edu/ACT/act/actr.html>

Table 2-2. Limitations of cognitive modeling and simulation

	Collaboration	Off-task behaviors	Multi-tasking	Interrupt and resume	Informal interactions	Cognitive behavior	Geography (location)
SOAR	No There is <i>only one</i> actor	No Tasks are performed through goal-directed reasoning	No Only through complex goal switching	No Goals <i>cannot</i> be interrupted and resumed	No There is <i>only one</i> actor	Yes	No
ACT-R	No There is <i>only one</i> actor	Yes The actor can react to perceptions from the outside world	No Only through complex goal switching	No Goals <i>cannot</i> be interrupted and resumed	No There is <i>only one</i> actor	Yes	No

The limitations of applying cognitive modeling architectures to work practice modeling and simulation are given in Table 2-2. However, the main reason for being skeptical about using cognitive modeling environments, like Soar and ACT-R, for modeling work practice has to do with the level at which cognition is modeled. As described in this chapter, using Soar and ACT-R as examples, cognitive models are represented at a level that allows the cognitive science researcher to model cognitive cycles in the human brain. These cycles have a length of no more than 50-100 msec. Even though I have not yet defined what needs to be included in a model of work practice, it seems clear that work practice needs to be shown at a higher-level of human activity than the low-level cognitive cycles of a person. One of the reasons for believing this is because of the scale factor. If we want to model a work practice in an organization of, for example, tens of people, it seems obvious that modeling each person in this organization at the level of their cognitive cycles would not be useful, even if it would be possible. The objective of a work practice model must lie at the level of showing what the total system behavior is. It seems obvious that this should be done by modeling each person's activities in the physical world, based on some relatively high-level representation of the reasoning behavior of each individual, and the impact this has on the individual's action in the physical world.

The notion of multiple actors or agents being involved in problem solving, and the research into organizations of multiple actors is the topic of the next section, in which I will discuss the research field of distributed artificial intelligence.

2.3 DISTRIBUTED ARTIFICIAL INTELLIGENCE

I am interested in modeling and theorizing about activities of people. People are inherently social actors and we, therefore, must be concerned with the social dimension of action and knowledge. Gasser states that classical artificial intelligence (AI) research is largely *a-social*, meaning that the unit of analysis is a computational process with a single locus of control and knowledge (Gasser 1991). Because of this, it has been inadequate with dealing with social human behavior. Gasser investigates how contemporary DAI deals (and should deal) with the *social* conception of knowledge, action, and interaction. In (Gasser 1991) he makes an argument that distributed artificial intelligence (DAI) is fundamental in the research on how agents coordinate their actions, use knowledge about beliefs, and reason about the beliefs and actions of other agents. Here, I draw a similarity between the research problems in the DAI literature and the problems in simulating work practice.

The notion of “social” in DAI is in my opinion too limited. The term social in DAI is meant as “more than one.” That is, DAI does not concentrate on problem-solving behavior as sitting in the head of a single agent, but investigates problem-solving behavior as a distributed multiagent process. As such, the word “social” means that there is communication and coordination between multiple agents in a problem-solving task. Detail can be found in (Bond and Gasser 1988; Gasser and Hill 1990). However, the definition of “social” is broader, and relates more closely to the way the group as a whole acts and interacts in the environment. The notion of social conception of knowledge and action is not a new idea. Social psychologist George Mead stated (Mead 1934):

We are not, in psychology, building up the behavior of the social group in terms of behavior of the separate individuals composing it; rather we are starting with a given social whole of complex group activity, into which we analyze (as elements) the behavior of each of the separate individuals composing it. We attempt, that is, to explain the conduct of the individual in terms of organized conduct of the social group, rather than to account for the organized conduct of the social group in terms of the conduct of the separate individuals belonging to it. For social psychology, the whole [society] is prior to the part [the individual], not the part to the whole; and the part is explained in terms of the whole, not the whole in terms of the parts.

The traditional techniques and methods of AI do not include any fundamental social elements. The focus is on the individual as the object of knowledge, truth and knowing. Gasser provides a great example of this limitation in AI research. The example is centered on the concept of *commitment*, and provides an excellent description of how an individual’s commitment is not just based on his or her individual *relativized persistent goal* (Cohen and Levesque 1990). In contrast, an individual’s commitment is based on an agent’s overall participation in many settings (activities) simultaneously (Gerson 1976), exemplified by Gasser’s example (Gasser 1991):

For example, imagine that a Los Angeles industrialist takes off in an airplane from Narita airport, bound for California, after formulating preliminary business deals in Tokyo and telephoning her associates in Los Angeles. While flying, she is participating in many settings simultaneously: the activity in the plane, the ongoing business negotiations in Tokyo and in Los Angeles (where people are planning for her arrival and making business judgments while considering her views, even in her absence). Her simultaneous involvement in interlocking courses of action in all of these situations provides the commitment to her arrival in California. Both she and others balance and trade off her involvement in joint courses of action in many different situations. Moreover, whether she makes a choice or not, she is committed to landing in LA because the plane is not in her control. Her commitments in any of these settings *amount to* the interaction of many activities of many agents in many other settings. Since this multi-setting participation occurs *simultaneously* in many places, it can’t be located simply to where she physically ‘is’. In other words, the notion of commitment is distributed because the agent of commitment—‘she’—is a distributed entity.

Although this example focuses on the notion of commitment, it is an excellent example of *collaboration* between multiple people in a work system. This example shows the importance of individual participation, knowledge, and location in the system as a whole. It shows that resources are not interchangeable, and that the work practice cannot be understood without a close investigation of the collaboration between the individual agents and the context in which this collaboration takes place. As Gasser states (Gasser 1991):

[...] since continued participation is distributed and simultaneous, it isn't based on localized, individual choices and goals.

The next few sections discuss a number of agent-based systems from the field of DAI. None of these systems provide a complete solution for dealing with simulating work practice, however they have formed a basis for our Brahms multiagent system.

2.3.1 TacAir-Soar

In (Tambe et al. 1995) an intelligent agent simulation environment is described for simulating battlefield scenarios and the knowledge intensive reasoning of independent pilot-agents. This environment is called TacAir-Soar, a kind of virtual world that can be populated with not only humans, but also with intelligent automated agents (AI systems). Such synthetic environments provide a new laboratory in which intelligent agents can be studied. Intelligent agents can be substituted for humans, such that a large number of entities can be used to populate the virtual world. A benefit of such an environment is that artificial agents can simplify and speed-up experimentation by providing more control of behavior, repeatability of scenarios, and an increased rate of simulation, faster than real-time simulation.

The goal of TacAir-Soar is the production of behavior that is close enough to that of humans, and to force the other entities to interact the same way as they would interact with humans. Although ambitious, they do not have to deal with low-level perception and robot control. There is also no verbal interaction between opposing entities, and cooperating agents restrict their communication to the details of the current mission. TacAir-Soar was to provide synthetic pilots (IFORS) for all the missions in STOW-97 (Simulated Theater of War), a large-scale simulation of a tactical exercise that took place in 1997, including fighters, troop transports, reconnaissance aircrafts, and helicopters. The set of requirements for the automated pilots include:

- Goal-driven and knowledge-intensive behavior;
- Conformance to human reaction and limitations;
- Performance of multiple simultaneous tasks;
- Episodic memory.

Pilot agents in TacAir-Soar are created as individual knowledge-based systems within the Soar integrated architecture (Laird et al. 1987) (Newell 1990). TacAir-Soar represents a generic automated pilot. Specializing it with specific vehicle parameters provided to them during the briefing process creates specific automated pilots. These pilot agents then participate in battlefield simulations by flying simulating aircrafts provided by ModSAF (Calder et al. 1993), a distributed simulator that has been developed commercially for the military.

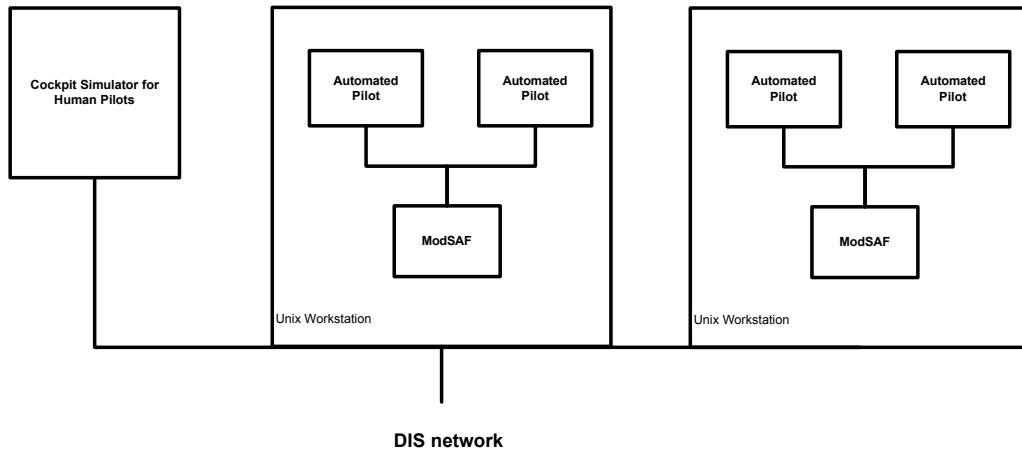


Figure 2-12. Human and Automated Pilots interact with the DIS environment using Distributed Simulators

TacAir-Soar has been constructed from Soar through the addition of perceptual motor interfaces (in the form of C-code) that allow pilots to fly ModSAF planes in the DIS (Distributed Interactive Simulation) environment (Thorpe et al. 1989). Each automated pilot is implemented as a Soar agent, interfacing with the simulated plane it is flying. To do this, they implemented a cockpit abstraction on top of ModSAF that allows TacAir-Soar to focus on behaving like a pilot, while ModSAF simulates planes, sensors, and weapons (see Figure 2-12).

2.3.1.1 Goal-driven task behavior

Each automated pilot agent has specific knowledge about tactical air-combat, and is implemented as a Soar knowledge base, having independent reasoning behavior. Goal-oriented and knowledge intensive behaviors are addressed by the manner in which Soar uses its architecture and knowledge in the process of dynamically expanding and contracting a goal hierarchy (Laird et al. 1987). Task switching also arises from Soar's decision-making abilities, but then specifically applied to the selection and switching of tasks. Tasks (also called goals) are represented as operators in Soar, and are the main foci of its decision-making. Task decomposition arises from Soar's ability to automatically generate a new task context when a decision is problematic. Task decomposition is achieved by combining task context generation with rules that generate preferences about which subtasks are appropriate for parent tasks. Real-time interaction with the DIS environment arises from the combination of Soar's incorporation of perception and action within the inner loop of its decision making capabilities—thus allowing all decisions to be informed by the current situation (and interpretations of it, as generated by rule firings) and the use of ModSAF as the interface to the DIS environment.

2.3.1.2 Conformance to human reaction and limitations

Reactivity of the individual pilot agents is addressed through a combination of Soar's use of productions to enforce context sensitivity in the representation of the knowledge, and Soar's decision-making procedure. Soar can react to changes by suggesting new goal-operators be pursued at any level in the same goal hierarchy, generating preferences among suggested operators. The decision procedure determines what changes have to be made to the goal hierarchy. Reactivity is limited to changes from within the decision-making process, and not from external available cues.

For example, communication between agents is simulated by the transmission of radio messages via the ModSAF simulation substrate, through the DIS network. That is, a discrete event simulates the transmission of a radio message between agents. The content of the radio message is received by an agent through the ModSAF interface into the working memory of the agent. Agents can react to radio messages through the activation of rules in the current task context that can propose new operators to be evaluated. This limits the agent to reacting only to radio messages that are relevant in the current context.

As a result, one of the limitations of TacAir-Soar in its use for modeling work systems is its inability to react to general external cues, other than the ones built in through the ModSAF cockpit abstraction and the DIS network. People, on the other hand, constantly react to the state of their external environment, regardless of their goal context at that moment. For example, when the telephone rings, people react to it. Depending on the activity we are involved in at that moment, we either pick it up, or let our answering machine answer. If we're not in the same location as the telephone, we do not notice the telephone ringing. When we decide to pick up the phone, we will most likely engage in a total different task context. This poses a problem for the way the Soar decision-making process works.

2.3.1.3 Performance of multiple simultaneous tasks

The Soar architecture is inadequate in certain requirements. One requirement that poses an inherent problem in modeling social human behavior is the inability of the simultaneous performance of multiple tasks. People inherently work on multiple tasks at once. TacAir-Soar agents need to simultaneously execute maneuvers to destroy the opponent, survive opponents' weapon firings, as well as interpret opponents' actions. The limitation of Soar is that it cannot create multiple goal hierarchies to serve multiple high-level tasks. The approach taken in TacAir-Soar to handle this problem is to create operators for simultaneous goals as needed, and to add these operators at the bottom of the active goal hierarchy. Although this may work, with sufficient care taken, it is not a real solution to the modeling of realistic human behavior. Soar interprets "lower" goal operators as being dependent on the higher-level goals, although these lower-level goals are supposed to be interpreted as goals for independent tasks. Jones, et al worked on changing the architecture to allow "forests" of goal hierarchies (Jones et al. 1994). Covrigaru approached the problem by being able to flush the current goal hierarchy whenever a new one needs to be established (Covregaru 1992). However, these approaches mean a change to the Soar architecture that would allow the dynamic compilation of new goal hierarchies.

2.3.1.4 Episodic memory

Endel Tulving introduced the term *episodic memory* (Tulving 1969). The notion of episodic vs. semantic memory arises from the central learning tradition of American experimental psychology. Tulving made clear that learning of verbal material was tied to a specific episode, i.e. a specific time and context in which the learned material was memorized, and thus was associated with. Episodic memory in TacAir-Soar is used primarily to support explanation. It is also used to a limited extent during simulation, so that an operator's current actions are interpreted based on its actions in the past. The general constraints are that episodic memory should add minimal processing overhead, and it should not substantially increase working memory. Episodic memory is a basic characteristic of human cognition, something a unified theory of cognition ought to provide for (Newell 1990). This brings tough issues for the Soar architecture especially. The approach taken in TacAir-Soar is that the sequence of events is recorded in working memory so that it can be recalled accurately. The states in which events occur are stored by committing state changes to long-term memory. Long-term memory employs chunking, which allows agents to learn new productions from episodic memory.

This is a point where TacAir-Soar wins over Brahms. Currently, Brahms agents have no significant episodic memory. Brahms agents only have a list of "current-beliefs" which is changing constantly. A trace of past activities, and states is not memorized. Brahms agents have no long-term memory, nor the ability of chunking. This is a future research issue that is closely related to the issue of learning. Although learning is an important piece of modeling human behavior, it falls outside the scope of this thesis. I return to a short discussion of learning as a topic for future research in the conclusion chapter of this thesis. However, for now suffice it to say that the Brahms simulation engine does keep track of all changes to the system, and that these historical events are created so that the history of a simulation can be saved in a database, and investigated post-simulation. However, at this moment, Brahms agents cannot access these historical events.

2.3.2 Phoenix: simulating fire-fighting in Yellowstone National Park

Paul Cohen, et al (Cohen et al. 1989) developed an agent-based simulation environment for simulating how fires in Yellowstone National Park are fought. The research objective of the Phoenix project is to understand

how complex environments constrain the design of intelligent agents. In contrast to the objectives of this research, in which the goal is to understand how work processes are created in terms of the work practices of people, the aims of the Phoenix research are more of a technical AI nature:

- Real-time adaptive planning
- Approximate scheduling for coordination of multiple planning activities
- Knowledge representation of plans and measuring progress towards the goals

They describe the Phoenix environment as follows:

We began the Phoenix project by designing a real-time, spatially distributed, multi-agent, dynamic, ongoing, unpredictable environment. (Cohen et al. 1989)

Phoenix is a multi-layered system. Figure 2-13 shows the architectural layers.

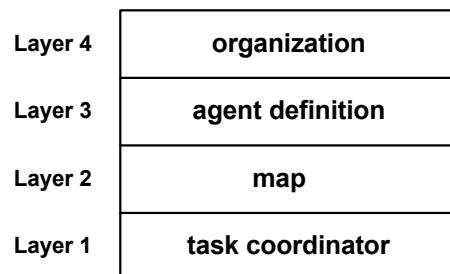


Figure 2-13. Phoenix layers

Layers 1 and 2 implement the environment (i.e. the forest and forest fires), in this case a part of Yellowstone National Park. Layer 3 is the agent design layer, which is the layer that creates the cognitive and reflexive behavior of the agents. Layer 4 is a model of the organization of multiple fire fighters.

2.3.2.1 Task coordinator layer

The task coordinator layer is responsible for creating the *illusion* of simultaneity among fires, and the agent's physical and internal actions. The task loops over all agents in each clock tick, changing the environment and the agent's actions accordingly. The clock grain size of the Phoenix simulator is 5 minutes. This means that the smallest action or change in the environment takes a minimum of 5 minutes of simulated time. You can increase the clock grain size to make the simulation more efficient. However, increasing the clock grain size will make agents become discordant with the environment, and with each other. When the clock grain-size becomes too large, it is possible for more than one action for an agent to have taken place in one simulation clock-tick. Communications between agents might have been missed, and changes in the environment might not have been recorded by the individual agents, leading to a situated-specific model that is not in sync with what actually happened in the simulation.

2.3.2.2 Map layer

The fire simulator resides in Phoenix's *map layer*. A Phoenix map is a composite of a two-dimensional data structure in which, for each map coordinate, information is stored about the environment. The environment is described as a set of symbolic features found at that specific coordinate on the map. These features include values for type of ground cover, elevation, features such as a road, a river, houses, et cetera., and the state of the fire at that coordinate (fire intensity).

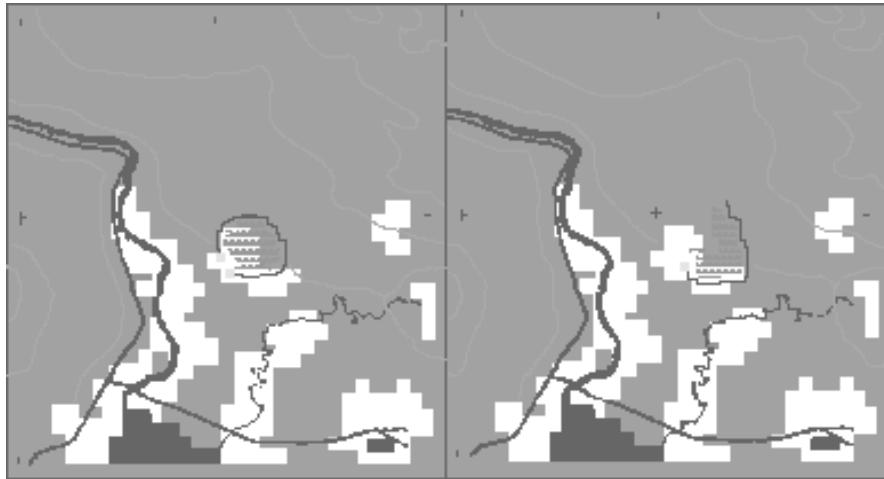


Figure 2-14. Screen display of Phoenix's situation-specific model (here gray and white indicate forest, which is seen hugging a river on the left side; the fire is near the middle (darker gray dots), nearly surrounded by the bulldozers in the view on the left). The left pane is the view of the real world, the right pane is the view of the fire boss.

The process of changing the map (i.e. changing the environment) during each simulation clock-tick is parameterized according to these defined concepts (Cohen et al. 1989):

Fires spread in irregular shapes and at variable rates, determined by ground cover, elevation, moisture content, wind speed, and direction, and natural boundaries. For example, fires spread more quickly in brush than in mature forest, are pushed in the direction of the wind and uphill, burn dry fuel more readily, and so on.... Fire-fighting objects are also accurately simulated; for example, bulldozers move at a maximum speed of 40 km/h in transit, 5 km/h traveling cross-country, and 0.5 km/h when cutting a fire-line. (p. 34-35)

The fire itself is implemented as a cellular automaton in which each cell at the boundary of the fire decides whether to spread to its neighboring cells. The simulator is generic in the sense that it can be used to simulate any type of environment involving maps and needs to simulate changes to the environment, based on the changes to the symbolic representation of the environment.

As is shown in Figure 2-14, the map layer also controls the movement of the agents, such as the bulldozers trying to surround the fire and cutting a fire-line, based on the situation-specific description of the terrain.

2.3.2.3 Agent Design Layer

Agents have two independent parallel mechanisms for generating actions: a reflexive component, and a cognitive component. The *reflexive component* generates actions for quick changes of direction on the order of seconds. The agent's sensors trigger reflexes. For example, when a bulldozer agent is about to drive into a fire, a reflex stops it and further reflexes handle the fine tuning of the movement of the agent to keep following the road without getting into the fire. Reflexes hardly cost any CPU time, but have no memory. They are merely a quick action (i.e. a reflex), based on triggers from the sensors.

The *cognitive component* generates and executes *lazy skeletal plans*, which are stored prescribed action sequences that, when executed, are instantiated with situation-specific data (Freed 1998). The cognitive component executes plans using a selection process that first decides which action to execute, next it finds out how much time is available to execute that action, and last it decides what execution method should be used for the action to execute based on the time available. An agent can execute several plans simultaneously; for example, when there are multiple fires to combat. The planning process goes as follows (Cohen et al. 1989):

Planning is accomplished by adding a selection action to the timeline to search for a plan to address some conditions. Executing the selection action places an appropriate plan action or primitive action

on the timeline. If this new entry is a plan action, then it expands into a plan when it is executed by putting its subactions onto the timeline with their temporal interrelationships. If it is a primitive action, execution instantiates the requisite variables, selects an execution method, and executes it. In general, a cognitive agent interleaves actions from the several plans it is working on. (p. 43)

Just as in TacAir-Soar, plans cannot be interrupted, because saving a state of the world and context switching is prohibited in the architecture. Therefore, to change an action the agent has to generate an error selection action “deal-with-error.”

The only way an agent can change its activities (i.e. re-plan) is to view a change as an error condition, and try to fix the current plan by expanding it into a kind of error recovery plan. Activities in Phoenix are sequential non-interruptible actions. However, the error-conditions in Phoenix are dynamically generated. This architecture allows for more flexibility in simulating the work activities of individual agents than workflow simulation models.

The reflexive and cognitive components interact via flags that are set by the reflexive component when reflexes execute. Since plans executed by the cognitive component can take several simulation hours to complete, the reflexive component takes over and changes the directional behavior of the agent. When the cognitive component notices the flag, it might react by changing its plan by calling the “deal-with-error” action. A Phoenix agent deals with two time scales requiring micro-actions, such as following a road, and cognitive processing, such as route planning. The combination of the reflexive and the cognitive components is designed to handle these time-scale mismatches.

2.3.2.4 Organization Layer

Agents in Phoenix are centrally organized in a hierarchical organization of fire-fighting agents. There is always one coordinating agent for each organization of fire fighters. This agent is called the *fire-boss*. The fire-boss is a purely cognitive agent that coordinates all fire-fighting agents’ activities, scheduling and communicating action directives. The fire-boss receives status reports from the fire-fighting agents, including fire sightings, position updates, and action completions. The fire-boss has a global view of the fire situation, while each fire-fighting agent has a limited individual view of the fire. Based on this global view and the updates, the fire-boss chooses global plans from its plan library. It communicates the actions in these plans to the agents. When an agent receives a plan, it selects a plan from its own plan library that implements the received action. Although the fire-fighting agents and the fire-boss communicate, there is no communication amongst the fire-fighting agents (i.e. there is no *cross-talking*).

The organization model in Phoenix is very limited, and is not conducive to collaboration amongst the fire fighters. The fire-boss orchestrates the work, and is a kind of meta-agent, whose sensors and detectors are the fire-fighting agents. It has the overall picture, the size of the fire, weather conditions, and action that already have been taken. None of this information is shared across the group of agents. Although this seems a good model to represent the TC in the T1 model (see 2.1.3.4)—it allows us to describe the coordination work of the TC—it does not allow for the agents in the central office to collaborate on a test with the technicians in the field.

This brings us to the next section in which we discuss the limitations of the Phoenix and TacAir-Soar systems in simulating the work practice of humans.

2.3.3 Discussion

Both TacAir Soar and Phoenix are multi-intelligent agent environments. The reason for choosing these two environments in my description of relevant previous work is that they are on the opposite ends of the spectrum, as far as simulating human behavior is concerned. TacAir Soar is based on the Soar architecture. Soar was developed as a theory of human cognition, stating that human cognitive processing is symbolic. Newell’s claim is that the Soar architecture is representative of how the human cognitive process works. As such, TacAir-Soar models pilots at a very fundamental cognitive level. The focus in TacAir Soar is on how the reasoning process of combat pilots can be simulated to the level of cognitive reaction times (i.e. human performance). The fact that there are multiple agents, simulating multiple pilots flying in a squadron, is to

allow for training of individual pilots. The focus is on the individual, and less on the collaboration between pilots. For example, it would be hard to simulate one pilot being attacked by an enemy aircraft, and another friendly pilot collaborating with this pilot to shake off the enemy aircraft.

Phoenix, on the other side of the spectrum, simulates a group of fire fighters who together fight a fire, coordinated by a fire-boss. At first, it seems that the Phoenix environment simulates collaboration. However, when we look closer we see that the fire-fighting agents in Phoenix do not even communicate together, a necessary prerequisite if we want to simulate collaboration. It is self evident that real fire fighters communicate together to setup strategies, change plans in times of despair, et cetera. A Phoenix agent does not represent how a fire fighter fights fires; instead it simulates how distributed lazy skeletal planning can be done. It would be very difficult to have a fire-fighting agent select the appropriate plan on the fly when suddenly, out of nowhere, a burning tree is falling down on a colleague ten feet away from where it is standing. Phoenix agents perform pre-specified plans that are assigned by a coordinating fire-boss agent. Because the overall view of the situation by the fire-boss, plans are being selected and agents directed. The agents are like robots acting out what the fire-boss instructs them to do, with local reactive behavior dependent on the environment they encounter, but independent of the fire-boss. In this sense, fire-fighting agents are fighting fires in groups without being aware that they are working in collaboration with other agents. An agent might be aware of some other agent, but it does not know what it is doing, and even more so, why it is doing what it is doing and that they are actually working together to fight the fire. In comparison with TacAir-Soar, Phoenix agents do not simulate the way human cognitive processes happen.

The goal of Phoenix is to design the right software-agent architecture for the needed agent behaviors and environmental characteristics to fight fires. Actually, the Phoenix architecture is a generic simulator for the *central* coordination of distributed agents in a natural environment. As such, the Phoenix environment contains all the components that are necessary to design an environment to simulate the work practices of individuals and groups. The problem is in the view the developers of Phoenix take in agent cognitive behavior. Phoenix, not surprisingly, uses a more classical AI planning approach, i.e. lazy skeletal planning.

With Table 2-3, I conclude this section on DAI by showing the strengths and limitations of the two reviewed systems, TacAir-Soar and Phoenix, in particular with regards to their ability to represent people's collaboration, "off-task" behaviors, multi-tasking, interrupted and resumed activities, informal interaction, knowledge and geography.

Table 2-3. Limitations of Distributed Artificial Intelligence

	Collaboration	Off-task behaviors	Multi-tasking	Interrupt and resume	Informal interactions	Cognitive behavior	Geography (location)
TacAir-Soar	No Communications of radio messages. Agents are <i>not</i> aware of each other's tasks	No Only high-level goal/tasks of the mission	No	No When contexts are switched tasks are stopped	No Only specific goal-directed behavior	Yes	Yes But, there is <i>no</i> separate geography model
Phoenix	No	No	Yes	Yes	No Firefighter agents for the most part are reactive agents. The fire boss agent is a deliberative	Varies	Yes

						agent	
--	--	--	--	--	--	-------	--

2.4 COMPUTATIONAL ORGANIZATION THEORY

Organizational issues were one of the first issues researched with the help of computational modeling. In the fifties, it was Herb Simon and his colleagues March and Cyert, at what was then Carnegie Tech, who started using computational modeling to create a behavioral theory of organizations (Simon 1955). Computer simulation is becoming a more accepted and indispensable method for organizational research and theory building. The need to understand organizational behavior is moving into the realm of detailed computational models of people, tasks, dynamic and adaptive ecological systems. Within organization theory models, organizations are often too complex to be analyzed by conventional techniques that lead to closed-form solutions. Computational organization theory (COT) researchers view organizations as a computational system of multiple “distributed” agents that collectively work on organizational tasks, use resources, have knowledge, skills, and communication capabilities. In (Carley and Prietula 1994b) and (Prietula et al. 1998), different organizational multiagent simulation models are presented as simplified descriptions of what happens in real organizations. Although simplified, these models are still sufficiently complex to simulate the dynamic behavior of organizations that allows for predictions and theory development. There are three basic reasons for using multiagent simulation as one of the main technologies in COT research (Carley and Prietula 1994b):

1. Many models contain non-linearities that cannot be eliminated. In modeling reality the non-linear behavior of individual agents and groups are central to the aspect that is being studied.
2. Differential equations do not deal with the differences of the discrete items in organizations, such as the people, tasks, and organization.
3. Agents in an organization act in parallel and adapt to the behavior of others. The behavior of a group is thus recursively defined. Controlling this fine order of agent interaction, and enabling agents to adapt is most effectively handled by simulation. Especially, if we are interested in investigating the behavior of large groups of agents, it is almost necessary to turn to simulation.

2.4.1 ACTS theory

Carley and Prietula, in (Carley and Prietula 1994a), describe an extended model of *Bounded Rationality* (Simon 1955). The model of bounded rationality states that agents in an organization may be rational in intent, but less than rational in execution because functional limits on cognition restricts their ability to achieve optimality in the pursuit of their goals (Simon 1976). The theory of bounded rationality was developed to replace the limited models of agents in theories of economics and organizations with a better approximation of the actual capabilities of people's decision making. The ACTS theory extends the model of bounded rationality and incorporates a general process theory of organizations. In the ACTS theory organizations are viewed as collections of intelligent agents who are cognitively restricted, task-oriented, and socially situated. The ACTS theory extends the model of bounded rationality in two ways:

1. It replaces the general principles of bounded rationality with a broader perspective of a cognitive agent.
2. It replaces the general notions of environmental constraints with specific environmental perspectives, a) the task, and b) the organizational social situation within which the task and the agents are situated—situatedness.

Within ACTS, organizations are build-up of individual intelligent agents whom together perform tasks, collaborate, and communicate in a social environment. The agent's knowledge, which is constantly changing, mediates the effect of the task and the social situation on the individual agent (micro-level) and organizational behavior (macro-level). Agents and tasks are situated in the organizational environment.

ACTS tries to explain such individual and organizational behavior and performance by supporting the development of a set of computational models:

- a cognitive model of an agent,
- a task model,
- a model of the social situation.

At the micro-level, the ACTS theory focuses on how a given organizational design affects the behavior and performance of individual intelligent agents whom communicate and reason within a social environment and situation. At the macro-level, the ACTS theory focuses on the behavior and performance of groups and organizations, given the fact that groups and organizations are comprised of intelligent agents whom are socially situated and task-oriented.

In ACTS theory the actions and decisions of intelligent agents are a function of the agent's cognitive architecture and knowledge (Newell 1990). The mechanisms by which an agent processes information, learns and makes decisions are a function of the cognitive architecture of the agent, the (social) position of the agent in the organization, and the 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 performance.

Many COT researchers use a multiagent version of the Soar system (Laird et al. 1987), because Soar is an implementation of a cognitive architecture that simulates the reasoning behavior of an individual. By integrating multiple copies of the Soar architecture in a distributed communication environment, a multiagent simulation environment for COT research can be created. In the next two sections, I briefly describe two of such environments: Plural-Soar, and Team-Soar.

2.4.2 Plural-Soar

Plural-Soar models a warehouse where multiple workers fill orders by retrieving items stored in stacks located in different places (Carley et al. 1992). In Plural-Soar, each Soar agent runs on a separate workstation, with the agents communicating over network connections. Plural-Soar consists of Soar production rules that define the agent's task knowledge. The task agents perform involves moving to a particular location (the order stack), and after possibly waiting in line, picking the next order from the stack, and then determining where the item from the order (in any of the known item stacks) may be in the warehouse, and last, moving to the item location to pick up the item to fill the order. Agents have a memory of the contents of the item stacks they have encountered. They can also broadcast requests for item locations to the other agents. This research focuses on the examination of how different combinations of cognitive constraints (such as communication and memory capability) combined with varying organizational structures (for example, different sizes) result in different organizational behaviors.

In subsequent work, adding elements (social knowledge) extended Plural-Soar in order to bring social elements of groups to the system (Carley et al. 1993). A social agent is defined by the decreasing information processing ability of the agent; omniscient, rational, bounded rational, cognitive, or emotional-cognitive, and by its knowledge of the social environment; non-social, multiple agents, interactive multiple agents, organizational structures, group goals, cultural history (Carley and Newell 1990). This model of a social agent was implemented in Plural-Soar in two ways. First, an agent has a social memory about the *reliability* of other agents. Secondly, the reliability of another agent depends on the correctness of previous given information by that agent. For instance, if inaccurate information was given by an agent, on the location of a specific item, the reliability of that agent is "down graded." After two unreliable pieces of information, an agent is determined "unreliable," and no more information from that agent is accepted. Thus, while in the first version of Plural-Soar information from other agents was accepted unconditionally, in the extended social agent version, communication from other agents is accepted or rejected on the basis of "social historical knowledge" of that agent.

In order to test the extremes of social behavior they then varied the social characteristics of agents: honesty, cooperation, and benevolence. In the study, they measured concepts such as cognitive effort, physical effort, communication efforts, and wait time from five different organizations. Conclusions then were drawn about the differences in these measures from different types of agents in organizations.

2.4.3 Team-Soar

Team-Soar models the decision-making behavior of a team of four commanding officers (CO) from different units in a naval carrier group (Kang et al. 1998). The four CO's are modeled as interconnected individual Soar agents. The agents are distributed in a simple authoritarian hierarchical organizational structure, which means that there is one leader, the CO of the aircraft carrier, and three equal subordinates, a CO of the coastal air defense, a CO of an AWACS air reconnaissance plane, and a CO of an Aegis cruiser. Each agent can communicate with all other agents, however the leader controls the team-based problem-solving task. The objective of the Team-Soar model is to track an aircraft by radar, and evaluate and decide in a team effort a course of action. To make a judgment, a Team-Soar agent first interprets the raw data in terms of nine attributes; speed, altitude, size, angle, direction, corridor-status, radar-type, range, and IFF, and evaluates them on a scale from one to three. When an agent has made the evaluations, by applying its knowledge in terms of Soar production rules, it makes a judgment about which of the seven possible actions to recommend (i.e. communicate) to the leader. Recommendations range in degree from ignore (a value of zero) to defend (a value of six), with intermediate states of review, monitor, warn, ready, and lock-in. When the leader has received the recommendations from all three subordinates, including its own, it makes a team decision, based on a team decision scheme, such as majority win or average win.

Team-Soar uses two types of communication strategies; one-to-one communication and broadcasting. One-to-one communication is used when one agent sends a message to another agent. This happens when the subordinate agents communicate their decision to the team leader agent. Broadcasting is used to send a message to all agents simultaneously. There are four different types of information that can be communicated: raw data, evaluations, judgments, and decisions. The raw data are the values of the nine attributes. An evaluation is an interpretation of the raw data. A judgment is a team member's recommendation on a decision. The decision is the team's final decision made by the team leader.

By varying the competence model for different agents in terms of domain expertise, meta-knowledge about the other agent's expertise, member judgment, agent type, cooperativeness, and activity, the team performance model can be varied and tested. Examples of two studies of team decision-making that have been done with Team-Soar are:

- To examine the relationship of a team decision scheme used and the amount of information available to teams with measures of team effectiveness.
- To explore the relationship of meta-knowledge (knowledge about the knowledge of the other agents) and the amount of communicated information with how long it takes for the team to reach a decision.

2.4.4 Discussion

COT research studies theories of organizational behavior by creating *simplified* models of real-life organizational systems. The objective of such models is not to create a representation of how people really perform the work or task; instead the objective is to create a controlled experiment to test a theory. Ironically, the researchers that are using a distributed multiagent version of Soar (such as Plural-Soar, and Team-Soar) argue that they use Soar because it claims to be a computational architecture for human cognition. A multiagent Soar architecture provides an implementation of the ACTS theory of distributed human problem solving, which allows the experimentation of subsequent theories on human distributed problem solving applied to human organizations.

When we look at the problem domains that are being studied, we can see that these studies involve either a very simplified version of a real-life organization (e.g. order fulfillment), or an organization, like the military, that works according to very stringent and pre-defined procedures. In contrast, the objective of the research

described in this thesis is to create a computational architecture to study the way people work in *real-life* organizations. The focus of the research is on the work practices that exist in a workplace, and not so much on the cognitive problem-solving behavior of individual agents, or on the study of optimal organizational structures given certain constraints. The computational models that we are after are models that, at the micro-level, can describe a person's daily activities and interactions with his or her environment and others, and at the macro-level show an organizational behavior that can be interpreted as the work practice existing in that organization. Our models are not meant as an experiment of organizational theory, but instead, as a laboratory to study work practices of people in real-life organizations.

Just as in the previous two chapters, I conclude here as well with a table showing the limitation of COT (see Table 2-4).

Table 2-4. Limitations of Computational Organization Theory

	Collaboration	Off-task behaviors	Multi-tasking	Interrupt and resume	Informal interactions	Cognitive behavior	Geography (location)
Plural-Soar	Yes Represented as the communication between agents to help in picking items from the warehouse	No Only high-level goal/tasks of picking orders	No	No When contexts are switched tasks are quit	No Agents <i>only</i> interact with other agents about task related issues	Yes	Yes But, no geographical reasoning
Team-Soar	Yes But, <i>only</i> represented as communication of specific attributes using the hierarchical organization of the CO's	No finding values for the eight variables. Agents do <i>not</i> decide themselves what to work on	No	No	No	Yes	No Only values of the variables related to the location of the incoming missile

2.5 CONCLUSION

I conclude my description of related work with some general observations about the different modeling and simulation tools used in the different research fields. My objective here is to take this back to the research topic of this thesis and relate these tools to the requirements for modeling work practice. To do this, I generalize the data from Table 2-1, Table 2-2, Table 2-3, and Table 2-4 and give you an overview of the analysis.

Collaboration

The tools and models described, either do *not* represent the collaboration between people, or have a *limited* representation. Some of the agent-based models allow for some indirect representation of collaboration through hard-wired communication channels and/or communication message content. But, only the workflow modeling tool (Sparks) has an explicit form of showing two tasks (using a spawn) being performed in parallel. However, due to the actual semantics, this type of parallelism turns out to not always perform tasks in parallel.

In modeling how tasks are performed in real-life organizations, being able to represent collaboration between people is a necessity. However, what is meant with collaboration is not immediately clear. What is clear is that this is a topic that has not been well defined yet in any of the research fields mentioned here. In the theory chapter (chapter 3) I will return to the question of what is meant with collaboration.

Off-task behaviors

None of the tools and models that were described represent off-task behaviors. The workflow paradigm has a representational limitation in not being able to allow for representing off-task behavior. This is because of the constraint on representing only those tasks where work-products are being touched. The other tools and models all take a goal-driven approach, and therefore do not allow for tasks outside of the domain goal/task hierarchies. ACT-R is the only tool that would allow for the representation of off-task behaviors, by using its ability to allow for reactive agent-behavior through input from the outside world. In general, the field of cognitive science does not focus on the influence of off-task behavior on the problem-solving capability of humans.

The one observation that can be made from this is that in these research fields there is no explicit focus on how people behave in real-life tasks being performed in real-life organizations. If this would be the case, we would see a lot more interest in representing the influence of off-task behaviors in the performance of tasks. If we want to model how people really work we do need to include the effect of off-task behavior in the model, since it is very obvious that people are constantly interrupted by the need to perform tasks that are not part of the explicit work. For example, just think about the influence of getting a phone call, while performing a task, or the scheduling of group meetings in organizations, and how this impacts the “rhythm” of our work.

Multi-tasking

By allowing interruption and resuming of tasks we can approximate multi-tasking. Only the Phoenix system (chapter 2.3.2) allows for such interleaving of multiple tasks. Only Sparks allows for the execution of actual parallel tasks. However, the question is what the meaning is of performing tasks in parallel. In Sparks the purpose is for showing percentage of resources being associated with parallel tasks. If we want to represent one resource performing two tasks in parallel—driving a car, while being on the phone—we can use a spawn of the flow. However, in Sparks it is not possible to associate specific resources with a task. Resources are picked from a resource pool. Only if there is one resource in the pool can we be sure that the parallel tasks are actually performed by the same resource in parallel.

Although people can do multiple things in parallel, and we do need to be able to show this, the actual way people perform most parallel tasks is by switching between them in very short time intervals. A lazy skeletal planning approach, in which the commitment of what task to work on next is delayed as long as possible is one approach to allow for this form of multi-tasking. Task-priorities is a way to decide what task to work on

next. Most parallel activities can actually be seen as hierarchical. With this I mean that showing someone doing two things at once can be represented hierarchically, in the sense that the person is in activity $A_{1,1}$, while also being in activity A_1 . The on-phone-while-driving example could be thought of as such a hierarchical multi-tasking event.

Interrupt and resume

As pointed out above, only the Phoenix system allows for interruptions and resuming of tasks through a lazy skeletal planning approach. This is an essential part of work practice. We, humans, are constantly interrupted in what we are doing. When an interruption happens we do not stop a task we are working on to start another unrelated task and restart the original task when we come back to it. We interrupt tasks to come back to them where we left off when possible or wanted. This type of reactive, and unplanned interrupt and resume behavior is natural, and is part of the reason why we humans are not brittle in our task performances. Therefore, if we want to model work practice, the ability to represent interrupted and resumed activities is a must.

Informal interaction

In none of the tools and models are there representations of informal interaction between agents. Every tool and model described represents only the formal organization and tasks. Resources and agents only interact when the task being modeled asks for it. In real-life organizations there is an abundance of informal interactions between colleagues. For example, having lunch at work with a group of people is an informal activity (i.e. no formal work-task is being performed). However, during lunch there could be a lot of work related communication going on. Therefore, in modeling work practice it is essential to allow for the representation of informal tasks and communications and informal group behavior.

Cognitive behavior

There is a sharp distinction between models and tools that do or do not include cognitive behavior. The interesting observation to make is that there seems to be an either-or approach to this. I mean, either the models are totally reliant on deep cognitive problem-solving behavior, to the point that every cognitive-cycle is being represented, or the models are at a level where the cognitive ability of the individual agent is not represented at all.

The question in representing work practice is, what level of cognitive behavior representation is important. It seems that the low-level problem-solving behavior of Soar and ACT-R are not necessary relevant in showing the relationship between people's activities in a work process. On the other hand, it seems important to represent each agent in the process, and the agent's knowledge of when and how to perform tasks.

Geography

There is a range in the representation of geography in the models and tools described. The range is from *no* representation (in Sparks and Soar) to a *simple* abstraction (in Phoenix). None of the tools and/or models have a very detailed explicit representation of locations and spaces. The only system that has a separate geography model (i.e. the map-layer) is the Phoenix system (see chapter 2.3.2.2). In the other models and/or tools in which an agent's location is somehow represented, it is done through an indirect representation of the agent *knowing* about location. However, there is no explicit objective representation of location and space.

People's environment impacts their work. In modeling work practice, it is important that we have an explicit representation of the location of people and their artifacts. In ACT-R there is an explicit representation of the outside world, but this representation is domain specific and is *not* a part of the ACT-R modeling language. If we want a language for work practice modeling we need to have, at minimum, the capability of representing the outside world inside the model.

I end with a comparison between the tools and models in the four research fields. Table 2-5 lists the domain dependency, technology, environment, communication-, problem-solving and group interaction model for

each human behavior tool and model described. One last interesting observation is that in workflow and cognitive modeling research there is a tendency to develop generic modeling tools, while the DAI and COT research fields have a tendency to use the tools from the other two fields for developing their models. In doing so, the tools are being applied in ways they were not specifically designed for. This leads to interesting extensions and changes. What this shows is the benefit of a multi-disciplinary approach to science, as well as some of its shortcomings by the mere fact that we can never include or re-use *all* of the theories from other academic fields.

Table 2-5. Human behavior model comparison

		Domain Dependency	Technology Used	Environment Model	Communication Model	Problem-Solving Model	Group Interaction Model
WFM	Sparks	General	Monte-Carlo discrete event simulation	None	None	None	Fixed (using spawns-tasks to show interaction between resources)
CM	Soar	General	Production System	None	None	Soar theory of cognition	None
	ACT-R	General	Production System	Fixed (using P/M interfaces)	None	ACT-R theory of cognition	None
DAI	TacAir-Soar	Dependent	Multiple Soar production systems	Fixed (representation of cockpit model using ModSAF)	Fixed agent content messages (through DIS network)	Soar theory of cognition	Fixed (agents interface through simulated cockpit in ModSAF)
	Phoenix	Dependent	Reactive planning (i.e. lazy skeletal planning)	Cellular automaton representation layer with low-level reactive behavior to environment	Fixed hierarchical agent content messages	None	No interaction between agents at the same level
COT	Plural-Soar	Dependent	Multiple Soar production systems	Fixed representation of the stack locations	Fixed communications of item locations	Soar theory of cognition, combined w/ ACTS theory	Varied based on social knowledge about other agents
	Team-Soar	Totally dependent	Multiple Soar production systems	Fixed attributes representing radar information for aircrafts	Fixed hierarchical agent content messages	Soar theory of cognition, combined w/ ACTS theory	Fixed attribute-level interaction with radar for tracking aircrafts

3. THEORY OF MODELING WORK PRACTICE

This chapter describes the theory of modeling work practice. I start out, in the first section, with given credence to practice as a valid concept of human behavior, separate from cognitive problem-solving behavior. I discuss a number of views on practice from the research literature. This will give the reader some background for the next section, in which I define the elements of work practice at an epistemological level. These elements are what becomes the driving force in finding a modeling language to represent work practice. In the section before the conclusions, I describe a model-based approach to work practice modeling and the operationalization of a model into a computational form (Sierhuis and Clancey 1997).

3.1 HISTORY OF PRACTICE

In this thesis I take a strong stance by fully adhering to practice as a valid form of knowledge that drives the behavior and actions of people. I, therefore, am of the opinion that we can objectify this knowledge in a knowledge-level representation of practice, much in the same way as AI researchers have created an epistemology of problem-solving knowledge. Ironically, the principle of rationality¹⁴ in AI comes from the Technical Rationality model, and it is this model that I denounce as the only model for defining knowledge (Newell 1982) (Newell and Simon 1972) (Simon 1976). It is this view that has the field of AI ignore the value of practical knowledge.

In this section, I attempt to show that practice is a valid level of knowledge that can be represented, not independent, but complementary to the problem-solving knowledge of humans in organizations.

3.1.1 Practical knowledge as knowing-in-action

Donald Schön offers an approach to an epistemology of practice, based on close examination of what practitioners actually do (Schön 1982). When people talk about *practice*, they often mean the practice of professions that have great social importance, such as medical doctors, lawyers, engineers, architects, et cetera. We even go as far as calling the business of a medical doctor his “practice.” When people talk about the practice of such professionals, they mean the exercise of *professional activity*. People believe that the schools in which these professionals have been taught give them a level of practical knowledge and experience that can be applied to solve daily problems. This view of practice is embedded within the model of Technical Rationality.

According to the model of Technical Rationality, professional activity consists of the application of scientific theory and techniques in problem solving. The knowledge base of a profession is thought to have four essential properties: it is specialized, firmly bounded, scientific, and standardized. This view of professional knowledge forces people, still today, to view practical knowledge—what is known in practice—as the application of professional knowledge, while *practice* is viewed as minor knowledge. Practice is said to be the application of scientific theory. It is said that applied science “rests on the foundation of basic science, and the more basic and general the knowledge, the higher the status of its producer.” (Schön 1982)

Why is the application of scientific theory and techniques to problems in practice the dominant view of professional knowledge? Why do we not put practical knowledge at the same level as professional knowledge? Paraphrasing Schön, the answer lies in the history of Western ideas about knowledge over the last three hundred years. Technical Rationality is the heritage of Positivism¹⁵, and the Positivist’s epistemology of practice. In the history of Positivism, practice is an anomaly. Practical knowledge exists, but cannot be seen as a descriptive knowledge of the world, and therefore is not seen as knowledge whatsoever. By viewing practical knowledge as the knowledge of the relationship of means to an end, the question “How ought I to act?” became a scientific one and the best means could be selected by the use of

¹⁴ Principle of rationality: If an agent has knowledge that one of its actions will lead to one of its goals, then the agent will select that action.

¹⁵ Positivism is the philosophical doctrine that developed in the nineteenth century. It was a social movement aimed at applying the achievements of science and technology to the well being of mankind.

scientific-based techniques. From the perspective of Technical Rationality, professional practice is, therefore, a process of problem solving.

With this focus on problem solving, the problem of *setting* and *situation*—in AI this is referred to as *context*—is ignored. Nevertheless, problems do not present themselves to the practitioner as givens. As Schön writes (Schön 1982):

[Problems] must be constructed from the materials of problematic situations which are puzzling, troubling, and uncertain. In order to convert a problematic situation to a problem, a practitioner must do a certain kind of work. [...] It is this [...] that professionals are coming increasingly to see as central to their practice.

Thus, the model of Technical Rationality leaves out the context of work. The practical knowledge used in performing the work constraint by the context, in which it occurs, is not seen as knowledge. The definition of knowledge in the model of Technical Rationality is incomplete in the fact that it does not view practice as a real category of competence. As Schön says it profoundly (Schön 1982):

If the model of Technical Rationality is incomplete, in that it fails to account for practical competence in “divergent” situations, so much the worse for the model. Let us search, instead, for an epistemology of practice implicit in artistic, intuitive processes which some practitioners do bring to situations of uncertainty, instability, uniqueness, and value conflict.

If we are able to put the Technical Rationality model aside, we come to the realization that practical knowledge is a kind of knowing inherent in intelligent action. Common sense admits that the category of *know-how* is in the action. Meaning that the know-how of workers is revealed in the way they act in problematic situations. It is this know-how that constitutes the practice.

3.1.2 Hermeneutics and work practice

Here I touch upon Heidegger and Gadamer’s philosophy of being and understanding, as it relates to work practice. The main source has been the groundbreaking work of Winograd, and Flores (Winograd and Flores 1986). I do not claim to have a full and complete understanding of either Heidegger’s or Gadamer’s philosophy (Heidegger 1962) (Gadamer 1976), and want to stress that I mainly touch upon their work as it relates to my ideas of what constitutes work practice. Most, if not all, of the credit has to be given to Winograd and Flores, since they explained the importance of hermeneutics¹⁶ for artificial intelligence, and more broadly for system design. It is their thinking that made us, who initially worked on Brahms, realize that if we want to understand the way people work we need to understand how people interact with and interpret the world. Therefore, we need to go beyond a description of individual cognition to a more holistic and social view of cognition as it relates to the way people work.

As Winograd and Flores explain, it was Heidegger and Gadamer who placed the hermeneutic idea of interpretation as the foundation of human cognition. Just as we can ask how *interpretation* plays a role in understanding text, we can ask how it plays a role in understanding the world as a whole. Winograd and Flores put forward four assumptions that, simply put, explain the way humans interpret the world (Winograd and Flores 1986, p. 30-31). Here I relate this, more narrowly, to the way people work, and I postulate the following four worldviews:

1. We are the inhabitants of a ‘real world’ made up of objects bearing properties. Our actions take place in the world.

¹⁶ The science and methodology of *interpretation* of texts, particularly mythical and sacred texts, such as the bible.

This means that the way people work is constrained by the location in which this work takes place. Therefore, if we want to model work practice we need to model the “real world,” its locations and the objects it is made up of.

2. There are ‘objective facts’ about that world that do not depend on the interpretation (or even presence) of any person.

This means that we cannot model a world by just modeling the individual interpretation of that world. We need to separate the different individual interpretations from the “objective facts.” Here is where we get confronted with *solipsism*¹⁷, i.e. the modeler of the “objective facts” is also an individual in the world, and hence also interprets the facts of the world according to his or her subjectivity. However, it is important to make a distinction between modeling the interpretation of an individual in a world, and the interpretation of facts in the world. Both are subjective, but both are necessary if we want to take a holistic view of the way people work. However, we should never forget that this means that our model of work practice is *our* interpretation, and not reality.

3. Perception is a process by which facts about the world are (sometimes inaccurately) registered in our thoughts and feelings.

This seems a trivial point after having made the point that every interpretation is a subjective one. However, the important issue that needs to be emphasized is that people make *inaccurate* interpretations of what they perceive, and that they will *act* according to (inaccurate) interpretations. It is therefore important to not only model the facts about the world, but also each individual’s perception of those facts, since it is their perceptions that make people act independently from each other.

4. Thoughts and intentions about action can somehow cause physical (hence real-world) motion of our bodies.

This means that if we want to model work practice, we need to model physical motion of individuals. We can satisfy this assumption by simply modeling the causal relation between thoughts about action and physical motion, and we do not need to model how this happens in the human body (i.e. the neurophysiology).

These four worldviews are my starting point for talking about work practice as a knowledge-level concept. By defining what this level is about, we will be able to represent our practical knowledge in a computational model in a similar way as we are able to model our problem-solving knowledge at a knowledge-level (Newell 1982).

3.1.3 Understanding context

A broad range of work in psychology and anthropology has shown that to fully understand how people work we need to study context in order to understand the relation between individuals, artifacts and social groups (Leont'ev 1978) (Vygotsky 1978) (Suchman 1987) (Lave and Wenger 1991) (Rogoff and Lave 1984). This chapter describes three approaches to study context—situated action models, activity theory and distributed cognition—that have been fundamental in the development of my theory for modeling work practice. All these three approaches use the notion of *activity* as the central point in the way they analyze the context in looking at human behavior.

3.1.3.1 Situated action models

Situated action models emphasize the emergence of activities within the situation. The focus is therefore on *situated action* or, what I call practice, as opposed to problem solving, which means that it is an inquiry into the everyday activity of persons acting in a particular setting. The analysis of situated action is a moment-by-moment analysis of the interaction between people, and between people and the artifacts used in a particular situation (Suchman 1987). Lave identifies the basic unit of analysis for situated action as the

¹⁷ The theory or view that the self is the only reality (definition in the American Heritage Dictionary, 2nd college edition). Kant called it “a scandal of philosophy and of human reason in general” that no philosopher had been able to provide a sound argument against solipsism.

activity of people as it relates to the setting in which this activity takes place and is constructed at the same time; “The setting both is generated out of [the] activity and at the same time generates the activity” (Lave et al. 1984). A *setting* is the relation between acting people and the arena in which they act, almost like a theatrical play. The *arena* is the physical place, i.e. the geographical space, as well as the institution with its social, political and economical background, like the stage within the theatre.

An important aspect of the focus on the activity of persons acting within an arena is that it forces the analyst to pay attention to the flux of the ongoing activity, the minute-by-minute understanding of a real activity in a real setting (Nardi 1996). One of the interesting notions coming out of situated action studies, put forward by Suchman, is that plans are not the mechanism to action, but that plans are resources for action; a “retrospective reconstruction” of situated action (Suchman 1987). In that sense, I postulate that goals are generated within the activity, as an individual’s rationalization of what the intention of the activity is; they are not the conditions of when activities are to take place.

3.1.3.2 Activity theory

Activity theory goes back to the 1920s, and developmental psychology work done in the former Soviet Union. The main developers of activity theory are Vygotsky and Leont’ev (Vygotsky 1978). In activity theory the unit of analysis is an activity. An *activity* is composed of a subject, the object, its actions and operations. A *subject* is the person or group of persons that is engaged in the activity. This makes the analysis of activities focus our attention on one or more people.

An *object* is the objective of the activity as it is held by the subject(s) and motivates them in the engagement. *Actions* are processes that must be undertaken to fulfill the object. Subjects are conscious about the actions to take to accomplish the object of an activity. Actions are more or less synonymous with tasks in cognitive science. The notion of an activity can span multiple actors being engaged together in coordinated actions. The actors engaged together might actually have different, even conflicting objects (Kuutti 1996). This is an important concept for the understanding of what collaboration between individuals is about.

Operations are routinized and unconscious actions. For example, when learning to drive a car with a standard gear, the shifting of the gear is at first a conscious action with an explicit goal. Later on, when we are well versed in driving with a stick shift, shifting gears becomes operational and is not a specific goal-driven process anymore. The difference of actions and operations reminds me strongly of the difference between explicit and tacit knowledge (Polanyi 1983). The important take away point from this is that it seems that activities are decomposed into actions, when the activity is not yet “automatic,” while an activity that is already operationalized is not decomposed into lower-level actions, but can be seen as a primitive action.

Another key notion in activity theory is the notion of *mediation* by artifacts (Kuutti 1996). Artifacts include instruments, machines, etc, that mediate activity and are created or used by people to control their behavior. In this sense an activity constitutes the context itself. An activity creates a context through its enactment of actions and operations of the people engaged in the activity, and using artifacts to control their engagement. As such, we can see practice as the engagement in activities over a period of time.

3.1.3.3 Distributed cognition

Distributed cognition is a branch of cognitive science that studies the representation of knowledge both inside the heads of the people, as well as within the artifacts and systems they use. The cognitive system can be seen as an activity in activity theory. For example, Hutchins, in his study of the activity of “flying a plane,” describes the cognitive system as the total setting of the cockpit (Hutchins 1995). He takes the *cockpit system* as the unit of analysis and observes the many representations that are inside the cockpit system, yet outside the head of the pilots. By taking this social-technical systems approach he can describe the “cognitive” properties of the system, meaning giving an account of the system’s behavioral properties in terms of its internal representations, without saying anything about the processes that operated inside the heads of the individuals within the system.

Thus, distributed cognition moves the unit of analysis to the system as a whole, and analyzes the functioning of the system as a “functional unit,” instead of as a cognitive system. In doing this, the emphasis

is on understanding the coordination among the individuals and the artifacts in the system. However, this understanding is created by focusing on the available information in the system, as represented in the artifacts and the heads of the individual. There is less of a focus on the activity and situated-actions as a whole, but more on how the lack of information creates a breakdown in the execution of plans and tasks by the individuals in the system.

One of the limitations of this approach is the necessity of drawing a boundary on the system to be analyzed at the start of an analysis. As opposed to letting the analysis of the setting be the driver in setting the boundary of the system. For example, Hutchins, in his study of the cockpit system, does not take into account the interaction and coordination between the pilots in the cockpit and the other crew and the passengers in the airplane (Hutchins 1995). Neither does he consider the interaction with the control tower and their view of the cockpit system. However, the interesting part of distributed cognitive analysis for getting an understanding of the work practice of pilots is the focus on the “memory” of the system as driving the activities of the pilots. This emphasizes the importance of a total systems view in the understanding of practical knowledge.

3.1.4 Work practice

Many researchers in the social sciences use the word *practice* as if it is a well-defined concept that everyone knows. However, it is difficult to describe what a practice is. People notice when something is not a practice, and can often describe why. 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 the word practice is:

Definition 1 (practice) *The (collaborative) performance of collective situated activities of a group of people who collaborate and communicate, while performing these activities synchronously or asynchronously, by making use of knowledge previously gained through experience in performing similar activities.*

In short, practice is doing in action (Suchman 1987). Scientists have described how a practice develops, like Wenger, who defines the creation of a practice as follows (Wenger 1997):

Being alive as human beings means that we are constantly engaged in the pursuit of enterprises of all kinds, from ensuring our physical survival to seeking the most lofty pleasures. As we define these enterprises and engage in their pursuit together, we interact with each other and with the world and we tune our relations with each other and with the world accordingly. In other words, we learn. Over time, this collective learning results in practices, which reflect both the pursuit of our enterprises and the attendant social relations. These practices are thus the property of a kind of community created over time by the sustained pursuit of a shared enterprise.

Everybody knows what Wenger means when he says, “this collective learning results in practices”, but what is it that results? Can it be described? Can it be modeled? To do this we need to be able to describe practice at an epistemological level.

3.1.5 How modeling practice is like Aaron’s drawing

Can there be a model of practice? Is a description of practice equal to practice itself? This is similar to the question; is a description of knowledge equal to knowledge itself? This is a debate in AI that has been going on for many years. Clancey makes an argument that allows us to get away from the arguments for or against this issue (Clancey 1997a). Clancey’s way of describing “the representation problem” allows us to ask the question differently, namely;

How can we create an internal representation of work practice, such that the observer interprets the external presentation of a simulation of a model of this work practice, as a reasonable description of the actual work practice?

Clancey describes Aaron, a robot built by Harold Cohen that creates original drawings. The question asked is; is Aaron an artist, or is Aaron a mere mechanical apparatus that can create drawings in a prescribed mechanical fashion? As Clancey states it:

[...] Cohen's dilemma is to understand the relation between internal descriptions, which he formulates and builds into the program, and outside behaviors, which observers will abstract and interpret in Aaron's drawings. (Clancey 1997a, p. 15)

In a private conversation with Clancey, Cohen revealed that his goal is not to create a robot artist, but to create a minimum representational configuration of drawings that will, when put on paper, be interpreted as an artistic image (Clancey 1997a, p. 16):

In this way, the product (what observers perceive) and the mechanism (what is inside the robot) are distinct.

Similarly, in this thesis the goal is not to create a mechanism for developing work practice, but to develop a representational language and simulation program that produces a model of work practice that is interpreted as such.

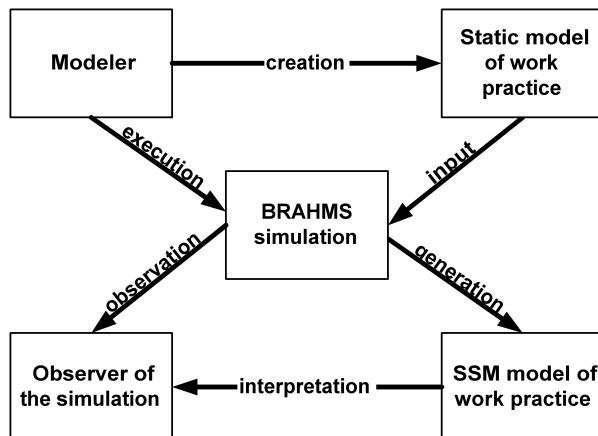


Figure 3-1. Relation of a model of work to a description of the work practice

The modeler in Figure 3-1 develops a model of the work using the representational power of the Brahms language. 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 simulation program, the model is simulated and a dynamic behavioral model of the work (i.e. a model of the practice) is generation. The observer of the simulation model can observe the model during and after the simulation, interpreting the work practice model.

In the next chapters I define what should be represented in a model of work practice.

3.2 ON THE EPISTEMOLOGICAL LEVEL OF 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. Not only am I 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, I 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 so, I focus the attention on work situations where multiple individuals from different professional backgrounds are collaborating. In contrast with Schön’s epistemological model of reflection-in-action in a specific profession, I focus the attention not on the application problem-solving knowledge of an individual, but on the collaboration of activities between individuals.

Work practices 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 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, I give definitions of the important elements: community of practice, activity, collaboration, communication, artifacts, and geographical environment.

3.2.1 Community of practice

People who are engaged in a work practice together belong to a *community* that has an identity (Wenger 1997). 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, et cetera. 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.

Definition 2a (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.*

I define a second type of community of practice (see definition 2b). The distinction between the first definition and the second is the type of people that belong to a community. The first definition (2a) includes individuals playing different roles and performing different activities. The second definition of community of practice 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, et cetera. 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.

Definition 2b (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 purpose of modeling, it is useful to make a distinction in the practice of a community in terms of different groups of people 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.

3.2.2 Activities

I now turn the attention to how the behavior of people can be represented as activities. In a knowledge-based system approach, the descriptive modeler's perspective of people's behavior is focused on a narrow description of what people do in terms of tasks and goals. Knowledge modelers start by choosing to model one task and the predefined goals that are to be pursued. With such a design approach, human activity appears to be a relation between goals, data, and decisions. For example in the PEES project, in modeling the "front-door sharing rule" in the Dutch social security law, I choose the interview with the client as the activity of the social security officer (van Dijck et al. 1987). I even focused more narrowly on the client's data specific for making a decision on how much his or her social security check was to be cut, ignoring the actual interview and the setting in which this takes place (Sierhuis 1986). Nevertheless, the social security officer is in the activity of "interviewing the client," as well as at the same time in the broader activity of "working in the social security office." I ignored the context of clients coming and going, colleagues asking for information about cases, people looking for the right forms to be filled out, clients asking for help. In designing the expert system for the "front-door sharing rule," I left out the "work life" of the social security officer. I ignored meetings, discussions in the hallways, the search throughout the building for the correct stamp needed. When I analyzed the task of determining the amount of social security a client would receive, I ignored most of the *activity* of the people in a social security office.

3.2.2.1 Activities versus tasks and goals

Imagine yourself going through a day. There is one response function when you get yourself out of bed, one when you reach for your clothes, one when you face yourself in the mirror, another when you go to breakfast and talk to your spouse, another when you get in your car and drive to work. Each of those situations is radically different and each calls for a quite different function about how to respond to the environment. One involves beds, floors, and covers; another involves mirrors and faucets, another yet something entirely different [...] Describing behavior as multiple response functions implies some sort of decomposition *within* the organism [...] How then should we describe systems? How should we describe their response functions? (Newell 1990, p. 43-44, emphasis added)

These are questions Newell asks in order to describe the foundations of cognitive science. He is interested in describing the workings of the *individual* information processing system (IPS). In other words, the way the individual comes to behave a certain way, or as he says it, "the working of the response functions." This is the individual IPS view he developed with Herb Simon, focusing his theory on *individual problem solving* as the way to describe individual behavior (Newell and Simon 1972).

The theory of humans as an IPS defines problem solving in terms of pursuing pre-specified *goals* in order to accomplish pieces of work that need to be done (i.e. *tasks*). The specification of a goal is a way to make a stated problem actionable, i.e. solvable by means of well-defined decisions. Problem solving is the systematic search over the problem space describing how one can attain a goal. Such an approach is in contrast to a theory for describing *how people actually work* within the constraints of their environment, and how the environment determines their actions and the interactions with other people and artifacts in that environment. Describing the behavior in terms of what actually happens in the world does not lead to a description of the individual's problem-solving behavior. Rather, it leads to a description of the emergent *total system behavior* in terms of the individual interactions, responses to the other elements in the system (people and artifacts), as well as the emergent sequence of individual *activity* (i.e. the state of being active), something Newell calls "microepics."

As is evident in my attempt to make this subtle, but important distinction, the focus in modeling work practice is on the IPS being the *total system*, including the environment, its people, artifacts, places, and time (see chapter 3.1.3). The emphasis of behavior lays at a broader level, namely at a level of *interaction between* discrete entities in the system, each being an IPS in its own right, but influenced by the other elements (IPS's) in the system. Problem solving happens at the individual level, while conceptual construction of activity (i.e. practice) happens at the system level. By describing the individual activity and interactions of elements in the system we can understand the behavior of the total system, as a result of the problem-solving behavior at the individual level. 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.

In this view of system behavior, *activities* are socially constructed engagements situated in the real world, taking time, effort and application of knowledge. Activities have a well-defined beginning and end, but do not have goals in the sense of problem-solving models. Instead, the goals are conceptual constructs created and articulated within activities of individual IPS's. Viewing work as activities of individuals 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.

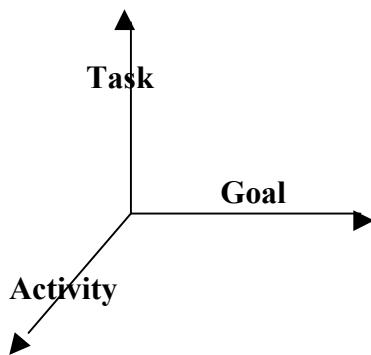


Figure 3-2. Dimensions of behavior

In this sense, as is shown in Figure 3-2, activities are orthogonal to tasks and goals. While engaged in an activity, people might articulate the task that they are working on, and the goal that they want to accomplish, but these are constructed within the activity. An example of an activity is pursuing a research career. A goal within this activity might be to get a research paper accepted for a conference. A task to reach that goal might be to gather all the relevant literature for the paper. The task and goal are created within the activity, but they are not determined by the activity (Clancey 1997b), meaning that they could similarly arise outside of that particular activity in another. 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."

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. Instead of viewing "social activity" as something that people do together, such as "socializing at a party" or "the social chat before the meeting," I take a social behaviorist's view. Describing human activities as social means that the tools and materials we use, and how we conceive of what we are doing, are culturally constructed. 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 in his hotel, as relaxation, while on a business trip. Engaging in the activity of "being on a business trip," there is an even larger social activity that is being engaged in, namely "working for the company," 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.

As represented in Figure 3-3, we can be in more than one activity at the same time. While performing one particular activity, we are also engaged in a larger, broader activity. For example, while in the broader activity of working on my dissertation, I am in the middle of the activity of writing the section on activities when my sister-in-law comes in the room to say good-bye. At that moment I suspend the activity of writing

the section, get up and go downstairs to say good-bye, which is the activity that I then engage in for a couple of minutes.

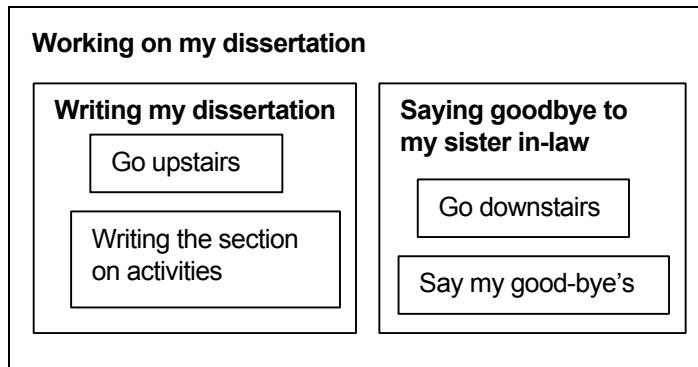


Figure 3-3. Activity subsumption

After my good-bye's I go back upstairs and continue my suspended activity of writing the section of my thesis. While saying my good-bye's I am still in the broader activity of working on my dissertation, otherwise there would be no reason for me to go back upstairs and continue writing. 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.

The idea is that humans can control their own behavior—not ‘from the inside’, on the basis of biological urges, but ‘from the outside’, using and creating artifacts. (Engeström 1991, p.12)

People choose which activity they engage in, but 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, et cetera. 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.

Definition 3 (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.*

3.2.3 Collaboration

One of the fundamental elements of work practice is the *collaboration* between individuals. An individual rarely works in isolation. Even if we would focus on the practice of one of the major professions, like a medical doctor, an architect or an engineer, we have to realize that they are acting in a context that includes more than just themselves. For instance, the doctor serves patients, and is paid for his services by an HMO. In the office there are physician assistants, nurses, secretaries, et cetera. They are all part of the picture; they collaborate with each other and with each patient that walks through the door. Even when there are no patients there are collaborative activities that take place, such as doing laboratory tests, entering results of tests into the patient's records, calling the pharmacy about prescriptions, and doctors mentoring the physician assistant. In short, the people in the doctor's practice collaborate (Wenger 1997).

Collaboration is a conceptual phenomenon that happens during the collection of activities being performed by the collaborators. Most individuals speak of having “a collaboration” when they feel that the activities engaged in with others is helpful to whatever the objective of the collaboration is. Mead calls it a *social act*, in his point of view of social behaviorism.

A social act may be defined as one in which the occasion or stimulus which sets free an impulse is found in the character or conduct of a living form that belongs to the proper environment of the living form whose impulse it is. I wish, however, to restrict the social act to the class of acts which involve the co-operation of more than one individual, and whose object as defined by the act, in the sense of Bergson, is a social object. I mean by a social object one that answers to all the parts of the complex act, though these parts are found in the conduct of different individuals. The objective of the acts is then found in the life-process of the group, not in those of the separate individuals alone. (Mead 1934, p. 7, footnote 7)

Collaboration can happen when two or more people work together at the same or at different times, being either in the same place or at different places.

The social act is not explained by building it up out of stimulus plus response; it must be taken as a dynamic whole—as something going on—no part of which can be considered or understood by itself—a complex organic process implied by each individual stimulus and response in it. (Mead 1934, p. 7)

In addition, collaboration can happen without people being conscious about it.

The mechanism of the social act can be traced out without introducing into it the conception of consciousness as a separable element within that act; hence, the social act, in its more elementary stages or forms, is possible without, or apart from, some form of consciousness. (Mead 1934, p. 18)

Especially for these forms of collaboration a work practice model could be useful in showing them, making the phenomenon visible and thus explicit. For example, my work practice has changed since I have moved from New York to California. My colleague in New York and I now use e-mail to discuss our research, whereas before we were mostly collaborating face to face, during our daily commute. Our form of collaboration has changed from same-time/same place to different time/different place. It is interesting to observe that we changed our communication tools as well (see paragraph 3.2.4). All this would be difficult to show in a workflow model, but in a model of work practice we include the different geographical places, as well as the different times we are each in our separate activities of reading and replying to our e-mails. The model would also show our new tool for communication (i.e. using e-mail), as well as the information (the stuff we are writing) we are communicating through our e-mails.

Collaboration is a conceptual creation, a state of *mental awareness* by the individuals collaborating. This mental awareness does not necessarily have to exist at the same time, in the same place, and in the same way for every individual in the collaboration. Such awareness is created at the moment we are in our individual activities, making us feel we are collaborating. Collaboration integrates the activities of the individuals in the group, thus establishing a community of practice.

Definition 4a (collaboration) A collaboration is a collection of activities of two or more individuals, all of them with the mental awareness (being conscious) of working together, either at the same time or at a different time, and either being in the same place or in a different place.

However, this definition does not capture the fact that people can collaborate even when they are not aware that they are collaborating, i.e. the mental awareness does not exist for them (see the above quote of Mead). I call this *indirect collaboration*. For example, when telephone company sales representatives add

order information to the order databases, this information is used to provision the telephone circuit by the trunk assignor at a different time and in a different office. The two individuals are not aware that they are collaborating when they are in their respective activities. Nevertheless, they are *indirectly collaborating* by the communication of the order information through the order database. Indirect collaboration is sometimes an external observer's conception, however, most of the time the people who engage in such indirect collaboration know that this takes place. It is especially in the breakdown of such collaboration—as when the information in the database is incorrect—that they realize this indirect collaboration they are engaged in.

Definition 4b (indirect collaboration) *An indirect collaboration is a collection of activities of two or more individuals, whom together, without mental awareness (not conscious) of the collaboration, but satisfying their individual goals, using an indirect form of (i.e. asynchronous) communication, either at the same time or at a different time, and either being in the same place or in a different place.*

3.2.4 Communication

Having defined collaboration as a collection of activities, direct or indirect between people, I now turn to how people coordinate their collaboration. The short answer is, through *communication*. In order for two or more people to collaborate they need to communicate. In the Speech Act theory by Searle, the meaning and intent of speech acts are formalized (Searle 1969). Searle describes people's action in terms of sending and receiving speech acts triggering response actions. A *speech act* has at least four distinct types of acts that are all part of the act at the same time (Searle 1969, p. 24-25):

1. Uttering words is performing an *utterance act*.
2. Referring and predicating is performing a *propositional act*.
3. Stating, questioning, commanding, promising, et cetera. is performing an *illocutionary act*.
4. The consequence or effect on actions, thoughts, and beliefs of the hearers is the *perlocutionary act*.

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. 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 if a communication activity between two people actually happens, or does not happen. We need to include the *communication tools* used in the speech act, 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. For example, when I work at home I am not checking my office voice mail as often as I should. Without justifying this, it is simply not part of my work practice. Therefore, if someone is trying to contact me, by calling me at my office phone and leaving a voice mail, I might not respond to it for a couple of days. It is not an efficient way of getting a hold of me. Sending e-mail is a better way, since I am constantly checking my e-mail at home. This emphasizes 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, and how they are used in practice.

3.2.4.1 Content and Information transfer

Speech acts are abstractions of the content of a communication activity between speaker and hearer. For instance, directive speech acts attempt to get the hearer to do something. What is left out is how this collaboration actually takes place. The speaker is in a communication activity, communicating some question or command. The hearer, when receiving the communication, reacts to this communication—based on the illocutionary point—and will perform some activity that ends in a communication activity that communicates the hearer's response to the speaker—the perlocutionary act. In reality, this speech act is a collaboration between two people, and they are using a communication tool, such as a phone, e-mail, or even a face-to-face conversation. The time it takes for this collaboration to complete, and be successful, depends on when the hearer receives the initial communication, and is able to communicate his or her response back to the sender. If the phone rings and the hearer is not in the location of the phone, the communication will not succeed and the speech act will not be completed. If a voice mail is left, the hearer might check it latter on and, depending on the message, will either do what is being commanded or will first need to call the speaker back to ask for clarification. This sequence of activities, constrained by the communication tool used, is part of the collaboration between the speaker and hearer, and needs to be taken into account in a model of work practice.

Definition 5 (communication) *A communication is the activity (speech act) of directional transferring of information (in the form of beliefs), held by one individual called the sender, to one or more individuals called the receiver(s), using a specific communication tool (face-to-face, telephone, e-mail, fax, document, etc). After the transfer activity is complete, and successful, the receiver(s) will hold the same information (belief) as the sender of the information, and can now react to it.*

3.2.4.2 Communication tools and their impact on work practice

There are different tools for communication dependent on the location and time spans of the collaborating individuals, having a major impact on the work practice of the group. In one of our investigations, we found that two different groups of workers would use different communication tools for accomplishing the same task. The first group, a group of technicians and a manager, communicated “the assignment of the day’s jobs” in a morning coffee meeting. The technicians all come in to work around eight o’clock in the morning. The manager who comes in at seven, will have scheduled the jobs for the day, and will sit with the “force” to have a coffee meeting. During this social gathering, the manager would hand out the job assignments for the day. The second group, consisting of all the same level workers, with one having an acting role as a manager, does not engage in the assigning of the day’s jobs during a coffee meeting. Rather, the acting manager assigns the jobs through the job scheduling system. As the workers come in to work, they check their work assignment through the computer. This example shows the difference in the communication activities—consequently the communication tools used—in the practice of assigning jobs for the day. The social interaction and work practice in these two groups is different, which is clearly impacted by the mode (tools) of communication.

It is worthwhile to emphasize that this example shows that the type of communication tools used is an important element in the communication mode, and is one of the defining factors in work practice. Thus, it is important to model the communication tools and its uses in activities, as they define the mode of communication and have an impact on the work practice.

3.2.4.3 Communication effectiveness and efficiency

A communication activity can be seen as simply an information transfer that constrains future actions for the receiver of the information. Either the information is received or not, in which case there is a communication breakdown. A communication activity can be qualified in terms of its *efficiency* and its *effectiveness*. In a communication breakdown its effectiveness is zero. Receiving information means that the information was transferred from the sender to the receiver with an effectiveness of one. Thus, *effectiveness* of communication is a measurement about whether the information is received or not.

Efficiency is a measurement of how many intermediate communication-activities are needed to receive the information. For instance, when the sender uses a telephone as communication tool and the receiver is not

there to answer the phone, the sender can leave a voice mail. When the receiver listens to the voice mail, and the message simply gives the receiver the intended information, the efficiency of the original communication activity was two. This means it took two communication activities for the transfer of the information from the sender to the receiver. If, on the other hand, the voice mail message states for the receiver to call back the sender, and the receiver calls back after which the information transfer takes place, the efficiency of the original communication activity is three, meaning it took a total of three communication activities to transfer the original information.

Using these measurements we can measure the effectiveness and efficiency of a speech act between people, or between people and artifacts (such as a computer system or robot).

3.2.4.4 Same-location communication

When collaborators are in the same location there are a number of *communication modes* they can choose from. If collaborating with just one individual, *face-to-face* communication is used. In face-to-face communication, two individuals are communicating synchronously and instantaneously. For communication with more than one individual, at the same time, a *broadcasting* communication mode is used. The distinction between these two modes is that in a face-to-face communication other individuals around are ignored. In a face-to-face communication people act and react only to the person they are communicating with. When broadcasting, people open the collaborative activity to everyone in the same location, as if speaking to everyone at the same time. The social coffee meeting described above is an example where a broadcasting mode allows the individuals in the group to not only get the information about their own jobs for the day, but also hear what jobs are assigned to the others in the group. Such a communication interaction facilitates learning, because suddenly the job assignment task becomes a social interaction of the group. The individuals in the group can exchange additional information; such as telling a colleague, who was just assigned a job at a location, about the problems at the location.

At the same time, individuals, who are in a location with a collaborating group and are not part of that collaboration, can ignore a broadcasting communication. This means that we, as individuals, can selectively react to communication. People are in control of their own actions; this is part of the meaning of collaboration.

Definition 7a (same-location communication) *Same-location communication is a communication form where the sender and receiver(s) are in the same geographical location. There are two modes of same-location communication, face-to-face communication, and broadcast communication. Face-to-face communication consists of one sender and one receiver. A broadcast communication consists of one sender and multiple receivers*

3.2.4.5 Communication over distance

Communication over distance happens when the communicators are not in the same geographical location. This form of communication can happen in different modes, *same-time communication* (synchronous communication over distance), or *different-time communication* (asynchronous communication over distance). Depending on these two modes, different types of communication technology can be used.

One of the oldest forms of different-time communication over distance is using a messenger who plays the role of a communication device. A more efficient form is mailing or faxing a written document. Alternatively, the use of workflow systems, e-mail or voice mail is becoming increasingly standard. Of course, the telephone is one of the most frequently used forms of same-time communication over distance. As technology is becoming more advanced, different types of communication devices will allow us to collaborate over larger and larger distances, more and more synchronously. As these technologies are being used in the daily work activities, they become a part of the practice.

Definition 7b (communication over distance) *Communication over distance is a communication form where the sender and the receiver(s) are in different geographical locations. In communication over distance, there is a communication device used to communicate. The sender sends the beliefs to the device. The receiver(s) receives the beliefs from the device. There are two modes of communication over distance, same-time communication over distance and different-time communication over distance. In*

same-time communication over distance (direct communication over distance), the sender and receiver communicate instantly or with some short transmit delay, using a communication device. In different-time communication over distance (indirect communication over distance), there is a time span between the sender's communication with the communication device, and the receiver's communication with the communication device

3.2.4.6 Taxonomy of communication types in work practice

From the above description and definitions of communication a taxonomy is presented. The taxonomy also includes possible communication tools that can be used for each type of communication:

Communication

The directional transfer of information from sender to receiver

Synchronous Communication

Same-time communication between sender and receiver

Same-Place Communication

Same-time communication where sender and receiver are in the same location

Face-to-face

Broadcast

Communication over distance

Same-time communication where the sender and receiver are in different locations

Phone-call

Voice-loop

Asynchronous Communication

Different-time communication with a delay between sending and receiving

Same-Place Communication

Different-time communication where the sender and receiver are in the same location

Using Artifacts

Documents

Using Electronic forums

E-mail

Database (or electronic document)

Communication over distance

Different-time communication where the sender and receiver are in different locations

Using Artifacts

Fax

Mailed documents

Using Electronic forums

Voice-mail

E-mail

Database (or electronic document)

3.2.5 Artifacts

People live and act within a physical world. People use and create artifacts in almost all activities that 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 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 into 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.

Definition 8 (artifact) *An artifact is a physical object in the world.*

George Mead's social-behaviorist notion of *instances of the universal*, as well as Heidegger's notion of *break down* and *readiness-at-hand*, explains the role of physical objects—artifacts—in an activity. Mead, as well as Heidegger, uses the hammer and the activity of hammering as the 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 break down 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-to-hand* that is taken for granted in the activity, without the user's identification as an object. It is only in the break down, 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* the artifact is used and conceptually understood within the activity. Figure 3-4 shows this relationship.

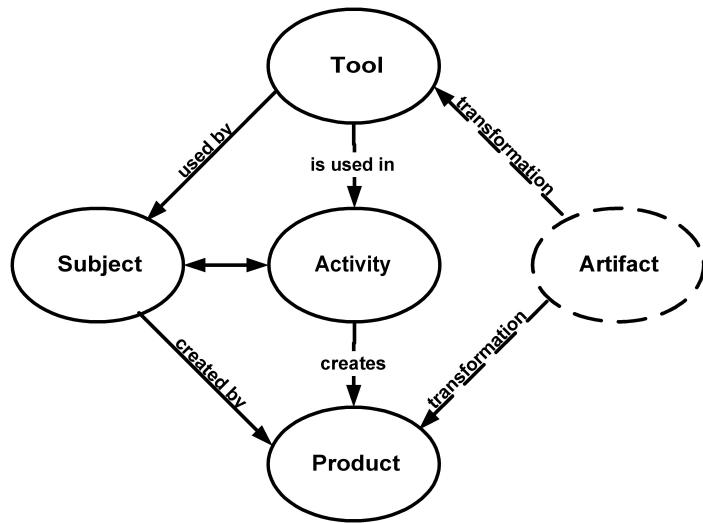


Figure 3-4. Mediated relationship of artifacts in activities

It is the use of the artifact in the activity—its role—that transforms the artifact into a *tool* 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 the object is necessary for the activity to be performed.

Definition 9 (tool) When an artifact is being used in an activity, it becomes a tool in the performance of the activity.

Definition 10 (product) When an artifact is created or changed in an activity, it becomes a product of the activity.

3.2.6 Geographical environment

Work is performed within a three-dimensional geographical environment. The restaurant we have dinner at, the office that we work in, and the moon crater the astronauts explore, are all examples of places, spaces, and environments which constrain the way we do 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 the practice, we need to include a model of the geographical environment in a model of work practice.

3.2.6.1 Important aspects of modeling the environment

Modeling geographical spaces is an intricate subject in and of it self. The question we need to ask is; how much and how detailed do we need to model the geographical environment if we want to show its importance to the work practice? The answer is that it depends on the work practice and the geographical space we are trying to model. If we are interested in office work, we need to model the office space in terms of where artifacts are located, such as where the offices of the people are, their telephones, fax machines, computer terminals, meeting rooms, et cetera. When we are modeling astronauts on an extravehicular activity on the Moon, we want to model the traverses, such as which craters they go to, how long it takes to go from one point of interest to another, which rocks are they looking at, and even which soil samples are they taking back with them. We are also interested in how they are traversing, and how long it takes to go from point A to point B. Are they walking or using a moon rover to travel. Are they aimlessly wandering around or are they following a pre-selected route? All these aspects are specified and constraint by the environment and the geographical space in which the work takes place. To give a concrete example of how the geography plays an important part in the way work happens, think about the things that might go wrong during a moon traverse, and how the environment constrains how long you can stay outside on the traverse. How much consumable oxygen do we have to get back to the spacecraft? This is a question that was constantly in the back of the minds of the people at mission control. It defined whether the next activity was to be done or was to be skipped. Dealing with the environmental constraints shapes the work practice.

Definition 11 (geography) *Geography is the description of the physical environment in which the people and artifacts are located when performing their activities.*

3.3 MODEL-BASED APPROACH

In this section I investigate how to operationalize a model of work practice. I use the term *operationalization* to refer to the implementation of a model of work practice that can be executed, i.e. a *computational model of work practice*. In this section of the thesis, I have described a framework for modeling work practice at an epistemological level. Here I investigate how we can implement such a model of work practice. This is the *operationalization problem* (Schreiber 1992). Generally, the term operationalization is used to denote the process of designing and implementing a system. In the context of computational modeling, the operationalization problem includes the ability to execute the model.

In the last decade, model-based development approaches have become the prevailing paradigm in knowledge-based system (KBS) development, as well as in more traditional system development. In KBS development, model-based refers to a development approach in which problem-solving expertise is described (represented) at the *knowledge-level* (Newell 1982) (Clancey 1985). One of the more well known model-based KBS design methodology is the CommonKADS methodology (Schreiber et al. 2000) (Schreiber et al. 1993). The CommonKADS methodology defines a number of design models that allow us to describe problem-solving behavior at Newell's knowledge-level. Much research has been done about how to operationalize KADS models of expertise (Angele et al. 1991) (van Harmelen and Balder 1992) (Karbach et al. 1991) (Linster and Musen 1992). In software engineering, model-based refers to a system design approach in which the system is described in terms of a number of well-defined design models (Yourdon 1989), using an object-oriented representation of the system that is being designed (Jacobson 1994). In this chapter I describe what is meant with a model-based approach for modeling work-practice.

We make observations from within our field of reality. A model is a description of that what we observe to exist in the real world. We create models all the time, mental or external, formal or informal. Mental models exist in our minds, and are our interpretation—description—of the world as we experience it. External models are models we create based on our mental models, and therefore, are manifestation of our mental models. In the context of this thesis, all external models are *system models* in the sense that they describe the world in components—*objects*—having properties, mirroring the properties of objects existing in the real world. When we create models that are not physically or geometrically identical with the world we are studying, we have to define system objects with properties that, for the purpose of our study, are similar to the real world objects. Secondly, the relations between the system objects have to be similar to the corresponding real world relations. In algebraic terms, the system objects and their relations have to be *isomorphic* with the real world objects and their relations in the real world.

3.3.1 Formal and informal system models

When we create non-physical, non-geometrical external models, we have a choice of creating these models with a formal representation or, as is often the case, with an informal representation. A *formal model* uses a description formalism that is predefined having a formal syntax and semantics. One of the benefits of a formal model is that the meaning of the model can be formally derived, and there can be no argument about this meaning. However, due to the formality of such models, creating and understanding formal models is often not a simple matter. On the other side of the spectrum, there are *informal models*. Informal models are models that do not have a well-defined meaning. Often the meaning of such models is in the eye of the beholder. Even though the meaning of informal models is not well defined, they can be useful in the understanding of a system. We all know the saying “a picture tells a thousand words.” This also holds for informal models. As such, I feel that the value of informal models is often similar to that of a picture of a scene. It gives context, an external description of reality that can be referred to and shared with others.

One of the benefits of creating external models is their use in analysis and design. External models can be used for explanation of relations and properties of a system that either already exists in the world or is to be developed; in which case the model is the only manifestation of the system.

3.3.2 Computational models

Despite some of the benefits, there are problems with informal models. Informal models cannot be used as a theoretical description of the real world. Therefore, we cannot use informal models to deduce new theorems—propositions about properties of the model. If we cannot do that, we cannot use the model to test hypothesis about properties of the world being modeled.

I distinguish two formal aspects of a system, namely a *structural* aspect of the system and a *behavioral* aspect of the system. Computational models are models that show the behavioral aspects of a system, by *simulating* the behavior of the system over time. This is in contrast with static models, which only show the structural aspects (i.e. the system elements and their relations at one moment in time). As the complexity of a system increases, understanding how the system changes over time—its behavior—becomes increasingly difficult. This is especially true for non-linear systems. A computational model allows us to observe the result of changes in the system as time moves forward.

A second problem with informal models is that they cannot be made computational, in the sense that they cannot be executed. Static models can only describe a system at a particular moment in time. They are a static representation of the interpretation of the modeler at the moment the world was interpreted. It depicts the model at a specific time slice. If a model is static it cannot be used to describe the changes over time of the world being modeled. In the case of modeling the work practice of a human activity system (Checkland and Scholes 1990) this is problematic. A static model could describe static properties of a system, but it fails to describe how dynamic properties change over time. Many elements of work practice contain *dynamic relations* between system objects, such as activities being performed by people, communications between people, changes in the environment, et cetera, et cetera. In other words, time is an important independent variable on which a lot of other variables depend. Therefore, if we want to model the work practice of a human activity system, we need to be able to create a dynamic model that can show how the system changes over time. In this case, we cannot use an informal description of work practice.

Figure 3-5 shows how our epistemology of work practice (described in this chapter), formalized in our Brahms modeling language and operationalized in the Brahms simulator (described in chapter 4), relates to a simulation of the work practice in a real world human activity system.

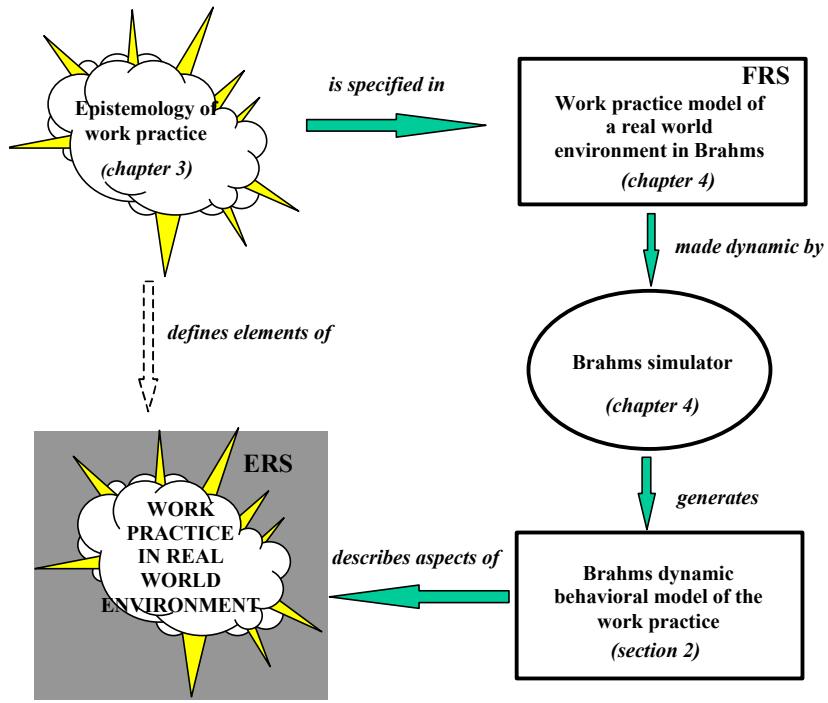


Figure 3-5. Describing real world work practice with computational modeling

3.3.2.1 The empirical relational system

The work practice in a real-world human activity system is an empirical relational system (ERS). It is empirical in the sense that it is the *source system* in which we can observe the objects and relations. The ERS refers to a group of people doing work in the real world, observed for the purpose of understanding the work practices of this group of people.

3.3.2.2 The epistemology of work practice

We observe the ERS by using the epistemological elements of work practice, described in chapter 3.2, as a sort of theoretical filter through which we view the empirical relations between the objects in the ERS. The elements of work practice we use in our filter are, again, community of practice, activities, collaboration, communication, artifacts, and geography.

3.3.2.3 The formal relational system

The elements of work practice, based on the epistemology, can be encoded into a computational Brahms model using the formal Brahms language (described in chapter 4). A Brahms model is a formal relational system with objects and relations isomorphic to real-world objects and relations in the ERS. The computational modeling language defines the formal relational system (FRS). In the FRS we describe the aspects of the work practice observed in the ERS. For each epistemological element observed in the ERS, there are formal Brahms language objects and relations that describe our observations.

3.3.2.4 The Brahms model simulator

From a computational Brahms model of the work practice a dynamic simulation model is generated, by executing the Brahms model using the Brahms simulation program (also described in chapter 4). This is the step in which the dynamic behavioral model of the work practice is generated.

3.3.2.5 The dynamic behavioral model

The behavioral model is a dynamic model in that it includes temporal activity-relations, and how they change over time.

The epistemological concepts of work practice define the theoretical basis of how to observe, capture and talk about work practice. The Brahms language operationalizes these epistemological concepts by defining a computational modeling language and a simulation program, allowing us to model and simulate a work practice from observations in the real world. Next, I describe how we develop a model of work practice.

3.3.3 Work practice models

This section describes the (sub)models of a work practice model, based on the epistemological work practice level described in section 3.2. I divide the work practice elements into related models that can be viewed independently. Dividing a model of work practice in this way helps the modeler with the decomposition of the domain, and makes the modeling effort easier.

3.3.3.1 Agent model

People are represented as *agents*. Just as people, agents do work. We can describe the work of people performing the same work, by describing the work of a *group*. Each member of the group is an *agent*, and is able to perform the work defined for the group. People can belong to multiple groups, and as such an agent can be a member of multiple groups. We represent the people in a community of practice as agents belonging to their respective groups. This way we can model any human activity system as communities of practice.

3.3.3.2 Activity model

The work that people do is described in terms of activities. *Activities* are defined at either the individual agent level, or the group level, in which case each member (agent) of the group can execute the activity. An activity represents the behavior of a person for a period of time. There are two types of activities, a *primitive activity* and a *composite activity*. A primitive activity is primitive, because it is not further decomposed, and takes some amount of time. The time element represents how long the agent is working *within* the activity. Thus, a primitive activity describes *what the agent is doing and how long the agent is doing that*. A composite activity is a higher-level activity. We can say, it is a more abstract representation of what the agent is doing. An activity can be decomposed in *sub-activities*, which can be primitive sub-activities, or again, composite sub-activities. Using primitive and composite activities we can describe what people are doing at any level of detail.

Work is the execution of activities under certain constraints. Agents' constraints for performing activities are matched against the beliefs they hold. We represent the constraints when agents can perform activities in an activity rule, called a *workframe*. A workframe defines the conditions under which the agent can execute the activity.

3.3.3.3 Communication model

We represent communication between people as an activity in which people engage when communicating with someone or something else. When communicating, people send or receive information. In our FRS, communication is represented as a type of primitive activity, called a *communication activity*. A communication activity is primitive, in that it takes a certain amount of time and is not decomposed into more primitive activities. In a communication activity we can specify what information the agent can communicate or receive, and with whom the agent is communicating when in the activity. Conditions in the workframes for communication activities specify under what circumstances an agent communicates.

3.3.3.4 Object model

People use and create artifacts in performing their activities. Artifacts are represented as *objects*. Types of artifacts, such as telephones, hammers, etc, are modeled as *classes*. New objects can be created as instances of a class. With these constructs we can model any type of artifact used within the work practice.

Some artifacts can perform activities, such as computer systems, telephones, microwaves, et cetera. Artifact behavior is represented similarly as the behavior in agents, meaning that the behavior in objects is also represented as activities and workframes.

3.3.3.5 Geography model

A human activity system is always located in some geographical space in which activities are performed. People and artifacts cannot be without location. Location also constrain when activities can be performed. For example, we cannot pick up the telephone if we're not located in the same geographical space as the telephone. We describe the location of where the agent and object's activities are performed in *geographical areas*.

Depending on the human activity system we can define types of areas with *area-definitions*, for example, buildings. A geographical area is an instance of an area-definition.

An agent and object performs its activities within only one geographical area. Moving from geographical area to area is represented as a *move activity*. A move activity is a primitive activity that takes time, and moves the agent from its current location to its new location. Using move activities we can formally describe the movements of people, during their activities.

3.3.4 Developing a model of work practice

Figure 3-6 describes an operational methodology for developing a formal computational model and a dynamic simulation model of a work practice, for an observable human activity system. A work practice is not simply the summation of the activities of all elements in the system, but it is the emergent behavior of the system as a whole, based on the interaction and collaboration between the elements in the system. Because a human activity system is about humans, we can observe *the way the humans are performing their activities*. In other words, we can observe the work practice of the system. The goal of the observation of the people in a human activity system is to create informal *static models* of the people, artifacts, the activities of those people and artifacts as they are being performed over time, as well as the geographical environment in which these activities take place.

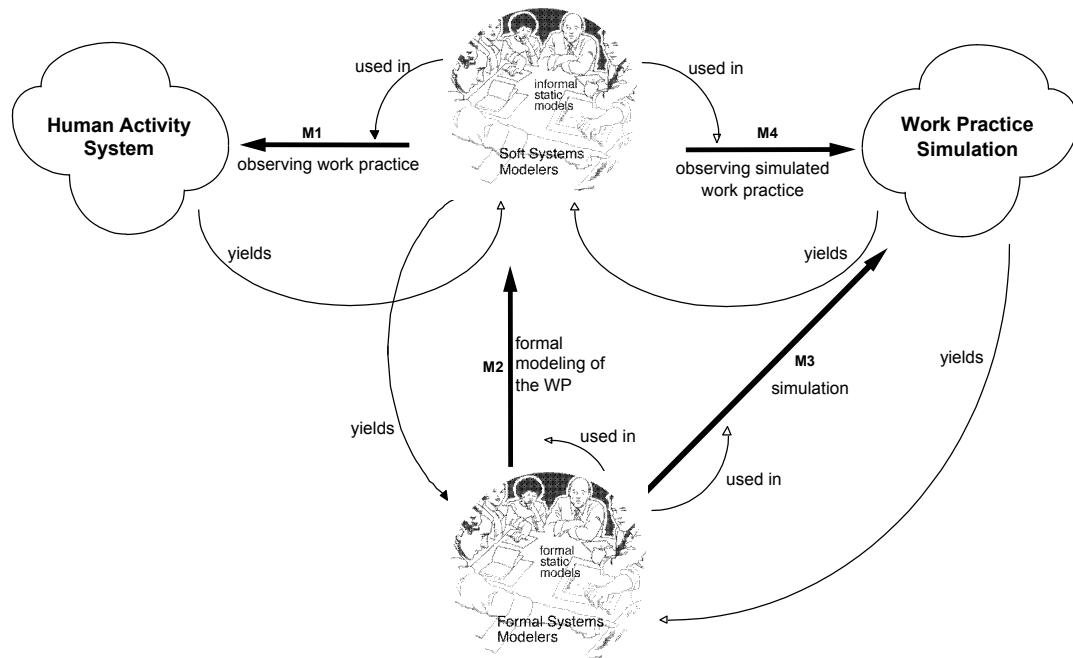


Figure 3-6. Modeling process

The empirical relational system is the human activity system being observed. The purpose of the methodology is to operationalize the modeling of the ERS, and create a Brahms model that can be executed by the Brahms simulator to create a simulation of the activities of agents and objects.

3.3.4.1 Method M1 – observing work practice

The purpose of method M1 is to observe the ERS and create an informal static description of an observation of the work practice of a human activity system. The goal of the observation is to create useful data to create static informal models, which will be used in M2 to develop formal models of work practice. There are different ways of observing a human activity system, and create data. I only mention two ways we can observe work practice in a human activity system, as examples. The first one is by analyzing video recordings of the actual work, and the second one is by using participant observation.

3.3.4.2 Method M2 – formal model of the work practice

The purpose of method M2 is to formalize the static informal models created during the application of M1, creating the FRS. In Brahms terms, this is where the Brahms model is developed. The formal system modelers need to be able to translate the informal models into formal models using a specialized kind of formal modeling knowledge. The formal modelers and the informal modelers do not necessarily have to be the same, and in fact, the skill set for these two types of modelers are very different. The informal modelers should be system analysts, knowledge engineers and anthropologists. The formal Brahms modelers should be people that understand the concept of agent-based modeling, and often have experience in developing rule-based systems.

3.3.4.3 Method M3 - simulation

The purpose of method M3 is to construct a simulation of the formal model, by running the simulator with the formal model as input and the work practice simulation as the output. The M3 method can be seen as the *model, compile, simulate, and debug cycle*.

3.3.4.4 Method M4 – observing the simulation

The purpose of method M4 is to observe and investigate the work practice simulation output, and compare it with the actual human activity system. It is during this cycle that the actual objective of the work practice simulation project is being accomplished. The result might be suggested changes to the formal model, in order to perform a what-if scenario. Thus, there is a modeling and simulation cycle between M1, M2, M3 and M4, which means that these methods have to be closely integrated if we want to make this cycle be as efficient as possible.

3.4 CONCLUSION

In this chapter, I discussed what I mean with “work practice.” In work practice modeling we focus on the collaborative activities of a community of individuals who collaborate together to accomplish a goal. I defined an epistemological framework for describing a work process at the *work-practice level*, using concepts such as collaboration, community of practice, communication, activity, and geography.

Having an informal *model of a work process at the work-practice level*¹⁸ could help us tremendously with our understanding of what is really happening within a work process. However, what has become clear from the framework is that practice is an emergent phenomenon that only shows its relationships and influences over time. Therefore, it is important not to leave out time. If we could simulate a model, we can observe how the work practice in an organization emerges. To allow for dynamics in a model, we need to make it computational. A model that is computational needs to be formal, so that it has a context-free grammar and a defined semantics. In the next chapter, I describe the formal Brahms language for modeling and simulating work practice.

¹⁸ From now on I will simply call this “modeling work practice.”

4. MODELING FORMALISM

In this chapter I describe the modeling formalism of Brahms¹⁹. Brahms models are written in an *Agent-Oriented Language* (AOL) that has a well-defined syntax and semantics. The Brahms language is a parsed language. A Brahms program is parsed by a LR(1) top-down parser. The parser generates an internal object representation for the run-time component. Using this language, a Brahms modeler can create *Brahms models*. The run-time component—the *simulation engine*—can execute a Brahms model; also referred to as a *simulation*.

Below is a paragraph from the abstract of the “official Brahms paper”, published in 1998 in the International Journal of Human Computer Studies (Clancey et al. 1998):

Brahms is a multiagent simulation tool for modeling the activities of groups in different locations and the physical environment consisting of objects and documents, including especially computer systems. A Brahms model of work practice reveals circumstantial, interactional influences on how work actually gets done, especially how people involve each other in their work. In particular, a model of practice reveals how people accomplish a collaboration through multiple and alternative means of communication, such as meetings, computer tools, and written documents. Choices of what and how to communicate are dependent upon social beliefs and behaviors—what people know about each other’s activities, intentions, and capabilities and their understanding of the norms of the group. As a result, Brahms models can help human-computer system designers to understand how tasks and information actually flow between people and machines, what work is required to synchronize individual contributions, and how tools hinder or help this process. In particular, workflow diagrams generated by Brahms are the emergent product of local interactions between agents and representational artifacts, not pre-ordained, end-to-end paths built in by a modeler. We developed Brahms as a tool to support the design of work by illuminating how formal flow descriptions relate to the social systems of work; we accomplish this by incorporating multiple views—relating people, information, systems, and geography—in one tool. Applications of Brahms could also include system requirements analysis, instruction, implementing software agents, and a workbench for relating cognitive and social theories of human behavior.

The Brahms language has its roots in other agent-based languages, such as AGENT-0 (Torrance 1991), and PLACA (Thomas 1993). The Brahms language is based on the formal logic of computational multiagent systems, as described by Wooldridge in his 1992 Ph.D. thesis (Wooldridge 1992). However, Wooldridge says that his theory was not intended as a model of human social systems. In this thesis I am describing a theory of human social systems. The theory focuses on meso human social systems—as a mid-level theory that links a micro-level mechanisms to macro-level phenomena, namely the physical and social to the individual cognitive (Carley and Prietula 1994a)—meaning that I describe a specific type of system, namely that of a *human activity system* (Checkland and Scholes 1990). As such we need to extend Wooldridge’s formal logic with provisions for modeling human-actors (social agents), including their activities, collaboration, their environment, and the fact that they are situated in the real world, acting and observing, reacting to and interacting with other agents, objects, and artifacts.

Brahms models may be thought of as statements in a new formal language developed for describing work practice. Appendix A shows the conventional notation and constructs used to express the syntax for the modeling language (BNF). The language is domain-general in the sense that it refers to no specific kind of social situation, workplace, or work practice; however, it does embody assumptions about how to describe social situations, workplaces and work practice.

The next chapters describe all the major parts of the Brahms language. Every major Brahms concept is described in a separate section:

1. Agents and Groups

¹⁹ Part of this chapter is taken from the Brahms US patent *Simulating Work Behavior*

2. Objects and Classes
3. Beliefs and Facts
4. Activities and Workframes
5. Geography
6. Simulation

The Brahms language is continuously evolving. Therefore, the description in this section is only a correct description for a short period of time. Just recently the Brahms development team released the new simulation engine, completely re-written in the Java language (the previous engine was written in a tool called G2²⁰). The new Java simulation engine resulted in an increased simulation speed of more than hundred times that of the old G2 engine.

For an always up-to-date description of the Brahms environment (language and engine), I refer to the Brahms web-site at URL <http://www.agentisolutions.com>.

4.1 AGENTS AND GROUPS

This section describes two important Brahms concepts for modeling individuals and groups of individuals. Agents and groups are central to Brahms. Defining the group/agent hierarchy for a model is one of central structures that need to be designed for any Brahms model.

4.1.1 Agents

The notion of an *agent* is central to the study of AI. In recent years a sub-field of AI has developed, called distributed artificial intelligence (DAI) (see Chapter 2.3). In DAI computer-based components appear as, more or less, independent agents (Gasser 1991). In the literature, there are two general usages of the term agent, a weak notion, and a strong notion (Wooldridge 1992).

4.1.1.1 Weak agency

Researchers concerned with weak agency focus on a shallow understanding of agents, meaning that they do not focus on human-like behavior. These researchers view agents as self-contained, concurrent software processes that encapsulate an internal state, and are able to communicate this internal state to other agents via a message passing protocol. In (Wooldridge and Jennings 1995), weak agency is described in the following general terms: A hardware or software-based computer system that employs the following properties:

- *autonomy*: agents operate independent from other agents and/or outside intervention, and have some control over their actions and internal state (Castelfranchi 1995);
- *social ability*: agents interact with other agents and/or users of the system, using some kind of communication language (Genesereth and Nilsson 1994);
- *reactivity*: agents have a way to perceive their environment, and can respond to changes in their environment. The environment may be the physical world, interaction with an end-user through some graphical user interface, the internet, and/or other agents;
- *pro-activeness*: agents do not just act in response to their environment. They are able to take initiative, through goal-directed behavior, in their actions.

Systems that fall into this camp are systems that behave as independent agents within a larger computational system, but do not possess human-like intelligent reasoning and behavior. Examples are web search-engine agents (also called “spiders”), software auction agents, web monitoring agents, et cetera.

²⁰ G2 is developed by Gensym Corporation.

4.1.1.2 Strong agency

Strong agency is used when the term *agent* includes the above properties, but also behaves more human-like. Researchers interested in strong agency belong mostly to the AI community. As is the norm in AI, an agent has cognitive behaviors, such as belief, desire and intention (BDI) systems (Shoham 1993). In other words, strong agency focuses on *intelligent agents*. In the ACTS theory, actions and decisions of intelligent agents are a function of the agent's cognitive architecture and knowledge (Carley and Prietula 1994a) (Newell 1990). The mechanisms by which an agent processes information, learns and makes decisions are a function of the cognitive architecture of the agent, the (social) position of the agent in the organization, and the 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.

Other attributes that are often discussed in the context of strong agency are, for example:

- *mobility*: agents “live” in an environment in which they can move around, be it in an electronic network (like the internet) or a closed environment, like a simulated world environment (Goodwin 1993).
- *bounded rationality*: it is assumed that agents act rational, but will act only in accordance with achieving their goals, and will not act in such a way as to prevent their goals from being achieved. In this sense agents act bounded to their ability to achieve their goals, and act according to their bounded cognitive ability. (Simon 1955) (Simon 1956).

In Brahms we use the notion of *strong agency*. The reason for this is obvious; Brahms agents model human behavior. In the next sections I describe how the Brahms modeling language implements, in some way, all of the attributes discussed in context of weak and strong agency, i.e. autonomy, social ability, reactivity, proactiveness, mobility, and bounded rationality.

People and artifacts (both physical and conceptual) are represented as “objects”, generally having properties, such as geographical location, which may change over time depending on their interactions. The term “agent” is generally used to refer specifically to an object that represents a person or, more inclusively, that represents an interactive system that has behavior interacting with the world that we want to represent as having the capabilities of awareness, reasoning and a mental state—intentionality.

An agent is a construct that generally represents a person within a workplace, or other setting being modeled. Agents have a name and a location. To specify what an agent does, the modeler defines activities and workframes for the agent. The key properties of agents are group membership, beliefs, workframes, thoughtframes, and location.

4.1.2 Groups

We could model the daily activities of actual individuals in an organization. For example, we can model the daily activities of agent “Maarten”. The activities will then be defined local to the agent, and will be agent-specific (i.e. not inherited by other agents).

Most Brahms models will not go into as much detail as to define the activities of individual agents, but rather describe the behavior of abstracted groups of agents in entities called *groups*. In describing the activities of groups, a specific member agent will inherit the activities of the group. In this way we can describe the daily activities of a group (i.e. non-individual specific).

A *group* can represent one or more agents, either as direct members or as members of subgroups. Typically, a modeler would associate descriptions of activities with groups, so that a group represents a collection of agents that perform similar work. A group may have only one member and roles may be highly differentiated. Depending on the purpose of the model, agents in a model may represent particular people, types of people, or pastiches. The modeler may define groups to represent anything, such as “service

technicians”, “people who like sushi”, or “people who wear spacesuits”. Each agent and group can be a member of any number of groups, providing that no cyclic membership results.

Groups and agents are the most central elements in a Brahms model. An agent represents an individual, whereas a group represents a group of individuals playing a particular role in an organization. The simulation engine schedules the constrained activities of individual agents, not for groups. However, being a member of a group, the group's constrained activities are scheduled as they pertain to the individual agent. A Brahms model is always about the activities of individual agents in a work process. Agents in Brahms are socially situated in the context of work, the organization, and its culture. However, the Brahms language is a multi-purpose AOL, which means that there is nothing inherently in the Brahms language that constrains the programmer to use agents for other purposes.

4.1.3 Elements of agents and groups

Brahms agents and groups²¹ have the following elements.

Name: The name of an agent is its unique identifier. Normally we give agents fictitious names to identify specific individuals in an organization without identifying them.

Display: The display name of an agent is a textual description of the agent's name. The display name can have spaces. It is not used as the unique identifier for the agent.

Group-membership: An agent can be a member of one or more groups. When an agent is a member of a group, the agent will inherit all elements from that group. An agent can be seen as an instance of a group in terms of object oriented practices. In case the same constructs are encountered in the inheritance path always the most specific construct will be used. For example, an activity defined for the agent has precedence over an activity with the same name defined in one of the groups of which the agent is a member.

Cost and time: The cost per unit (“Cost/unit”), and the unit time for which the cost is entered (“Unit (seconds)”). For example, if the cost attribute is 10.0 and the time attribute equals 3600 seconds it means that the cost of an agent is 10 BB's (Brahms Bucks) per minute. Using these attributes the simulation engine can calculate cost statistics of a work process, based on a calculation of the summation of an agent's activity time.

Location: An agent has an initial location within the geography (see chapter 4.5).

Attributes: Represent a property of an agent or object in the world. Attributes can have values. Currently only single-valued attributes are allowed. The value of an attribute is specified through facts and/or beliefs (see chapter 4.3).

Relations: Represent a relation between two concepts. A concept can be either an agent or an object. The first (left hand side) concept is always the concept for which the relation is defined; the second concept (right hand side) can be any other concept. Relations are specified through facts and/or beliefs (see chapter 4.3).

Initial-beliefs: A belief is a first-order predicate statement about the world (Konolige 1982) (Konolige 1986) (see chapter 4.3.3). Beliefs are always local to an agent, i.e. only the agent can access its beliefs, and no other agent can. This allows us to represent how a specific agent “views” the state of the world (Hintikka 1962). Agents act based on their beliefs. Beliefs are the “triggers” of agent's actions (see chapter 4.4.2). *Initial beliefs* define the initial state for an agent. Initial beliefs are turned into actual beliefs for the agent when the model is initialized at simulation start time.

Initial-facts: Facts represent the state of the world. A fact is a first-order predicate statement about the world (see chapter 4.3.4). Facts are, in contrast to beliefs, global. Any agent can detect a fact in the world and turn it into a belief and act on it. Initial facts define the initial state of the world. Initial facts are turned into facts in the world when the model is initialized for a simulation run. There is a fundamental difference between the

²¹ In the rest of this section I only refer to agents, however, this should be read as “agents and objects.”

“ownership” of a belief and a fact. A belief is “owned” by a specific agent during the execution of the model. No other entity in the model can access that belief without some interaction with the agent (direct or indirect). However, although initial-facts are defined with an agent or object, at execution time a fact is not “owned” by that agent or object. A fact is global, and can be acted on (in the case of objects) or detected (in the case of agents).

Activities: In this element the activities an agent can be engaged in are defined. Activities in Brahms take a certain amount of time, either derived or defined (see chapter 4.4.1). There are a number of types of activities that are defined for the Brahms language. Activities defined are executed by workframes.

Workframes: In this element the activity rules, called “workframes”, are defined. Workframes describe the constraints of executing activities. Workframes are situation-action rules (see chapter 4.4.2).

Thoughtframes: In this element the agent’s inference rules, called “thoughtframes”, are defined. Thoughtframes are inherently different from workframes, as they do not execute any activities and thus do not take any time during execution (see chapter 4.4.4).

For the syntax for agents see section 3 of appendix A.

For the syntax for groups see section 4 of appendix A.

4.2 OBJECTS AND CLASSES

Objects and classes are another important set of concepts in the Brahms language. Objects and classes can have similar behavior as agents and groups, except that objects should represent (non-) behavioral *inanimate* artifacts.

4.2.1 Objects

Brahms is different from other AOLs in that we make a definitional difference between animate—intentional—objects (which we refer to as agents) and inanimate—unintentional—objects (which we refer to as *objects*). In all other agent-languages there is only one type of object, namely an intentional agent. Shoham, Wooldridge, Castelfranchi all define their agent-languages as having intentional agents (Shoham 1993) (Wooldridge 1992) (Castelfranchi 1995). Dennett states that an intentional agent is an agent that “harbors beliefs and desires and other mental states that exhibit *intentionality* or aboutness, and whose actions can be explained (or predicted) on the basis of content of these states” (Dennett 1991) (Dennett 1987). In Brahms, our agents are intentional. However we also want to be able to describe artifacts in the real world as action-oriented systems, but unintentional at the same time. We describe such an artifact as an *object*. An example of an object in Brahms is a fax machine. If we want to describe the behavior of a fax machine, we could argue that we could describe a fax machine as an intentional agent. However, in the real world we would never ascribe intention to the actions of a fax machine. A fax machine mainly reacts to facts in the world; such as a person pushing the start button on the fax machine that makes the fax machine start faxing the document. Since in Brahms we are interested in describing the world with its animate and inanimate objects, we want the capability to make a difference between an intentional object (an agent), like a human, and an unintentional object (an object) like a fax machine.

An obvious question is whether it is useful to attribute beliefs and rationality, et cetera to inanimate objects. McCarthy, among others, has argued that there are occasions when the *intentional stance* is appropriate (McCarthy 1978):

To ascribe beliefs, free will, intentions, consciousness, abilities, or wants to a machine is legitimate when such an ascription expresses the same information about the machine that it expresses about a person. It is useful when the ascription helps us understand the structure of the machine, its past or future behavior, or how to repair or improve it. It is perhaps never logically required even for humans, but expressing reasonably briefly what is actually known about the state of the machine in a particular situation may require mental qualities or qualities isomorphic to them. Theories of belief, knowledge

and wanting can be constructed for machines in a simpler setting than for humans, and later applied to humans. Ascription of mental qualities is most straightforward for machines of known structure such as thermostats and computer operating systems, but is most useful when applied to entities whose structure is incompletely known.

When this is the case, we might decide to represent a machine as an agent. For example, when we want to model human-robot collaboration, it might be useful to represent both the human and the robot as an agent.

An object, in Brahms, is a construct that generally represents an artifact. The key properties of objects are facts, workframes, and activities, which together represent the state and causal behaviors of objects. Some objects may have internal states, such as information in a computer, that are modeled as beliefs²². Other artifact states—such as the fact that a phone is off hook—are facts about the world.

4.2.2 Classes

Classes in Brahms represent an abstraction of one or more object instances. The concept of a *class* in Brahms is similar to the concept of a template or class in object-oriented programming (Rumbaugh et al. 1998). It defines the activities and workframes, initial-facts and initial-beliefs for instances of that class (objects). Brahms allows for multiple inheritance for objects. Classes are used to define inanimate artifacts, such as phones, faxes, computer systems, pieces of paper, et cetera.

4.2.3 Elements of objects and classes

A Brahms object has all of the elements that an agent has, plus two additional elements; *conceptual object membership* and *resource*. Furthermore, instead of having a group membership relation with groups, an object can have *class-inheritance* relationships with classes.

A Brahms object has the following extra elements:

Class-inheritance: An object can be an instance of one or more classes. In case constructs with the same name are encountered in the inheritance path, always the most specific construct will be used. For example, an activity defined for the object has precedence over a workframe with the same name defined in one of the classes of which the object is an instance.

Conceptual-object membership: An object can be part of one or more conceptual objects by defining the conceptual-object-membership for the object. This allows for later grouping of statistical results and workflow.

Resource: The resource attribute defines whether or not the object is considered to be a resource when used in an activity (resource attribute is set to true), or whether the object is considered something that “is worked on” (resource attribute is set to false). The resource attribute is used in relation with the touched-objects definition for activities (see chapter 4.4.7.1 about primitive activities).

For the syntax for classes see section 5 of appendix A.

For the syntax for objects see section 6 of appendix A.

4.3 BELIEFS AND FACTS

One of the attributes of strong agency is *bounded rationality*. Bounded rationality states that an agent acts according to their bounded cognitive ability. In Brahms an agent acts according to its beliefs and its ability to deduce new beliefs from its current beliefs.

²² Brahms represents information stored in an object as beliefs. I am fully aware of this seemingly awkward naming, however the reason for this is to keep a minimal language specification.

In this section we describe the intentional notions of Brahms agents, and how intentional notions fit into a logical framework that has been researched by many (Konolige 1982) (Konolige 1986) (Genesereth and Nilsson 1987) (Hintikka 1962). We then describe how in Brahms agent and object intentions are described.

The state of the world and that of agents in Brahms is stored in informational units called “facts” and “beliefs”. A fact is meant to represent some physical state of the world or an attribute of some object or agent. Facts are global: with the appropriate workframe any agent can detect a fact (see chapter 4.4.9). Objects, on the other hand, only react to facts.

A belief is knowing about some fact by a particular object, typically an agent. Beliefs are always local to an agent or object; that is, only the agent or object itself can examine or search (i.e. reason with) its own beliefs, and no other agent or object can do so. The current value of a belief held by an agent may differ from the value of the corresponding fact. Beliefs can be declared as initial beliefs at the class, object, group, or agent level. An agent can also create beliefs while performing an activity or an inference in a Brahms simulation. A belief can be thought of as an object-attribute-value triplet.

The modeler can add initial facts at the class, object, group, or agent level, or created by the agent or object during simulation. Facts have the same components as beliefs.

The representation of beliefs and reasoning implements a conventional first-order predicate logic on beliefs. The modeler has available the full range of representation of, and reasoning on beliefs, conventionally found in rule-based systems such as EMYCIN (Van Melle 1979). However, because the logic is first-order, agents are not modeled as having second-order beliefs (beliefs about other agents’ beliefs).

4.3.1 Intentional systems

The philosopher Daniel Dennett has defined the term *intentional systems* as entities whose behavior can be predicted by the method of attributing belief, desires, and rational acumen (Dennett 1987, p.49). Dennett described multiple “grades” of intentional systems; first-, second-, third-, fourth-order, et cetera (Dennett 1987, p.243).

A first-order intentional system has beliefs and desires (etc.) but no beliefs and desires about beliefs and desires.

(1) x believes that p

A second-order intentional system [...] has beliefs and desires [...] about beliefs and desires [...] both those of others and its own. For instance

(2) x believes y expects x to jump left

A third-order intentional system is one that is capable of such states as

(3) x wants y to believe that x believes he is alone

Dennett asks his reader how high human beings can go. “In principle, forever, no doubt [...].” The question is: How high do our Brahms agents need to go in order to allow for realistic human behavior being simulated by our agents? It is clear that our agents need to be at least first-order intentional systems; since, if we want our agents to act independent from other agents they need to be able to act on their own beliefs about states of the world. Now then, do our agents need to be higher-order intentional systems? In exploring these multi-order phenomena, Cargile explains that humans can keep track of about five or six orders (Cargile 1970).

To be able to model social knowledge of humans we believe that our agents need to be at least second-order intentional systems. Brahms agents should be able to act on their beliefs about their beliefs, and also be able to act on beliefs about beliefs of other agents. Second-order beliefs allow us to model, for instance,

that a worker does a certain activity, because (s)he knows that another worker, whom is supposed to do the activity, believes that the activity is not necessary for the situation at hand. This allows for a level of intentionality that is very common in human behavior. The ability to reason with second-order beliefs would allow us to model agents such as tutoring agents. Tutoring agents need the ability to create a model of the agent it is tutoring. Such a model would include many beliefs about the beliefs the agent being tutored has.

Although it is necessary to be able to model second-order beliefs for intentional agents, up to this point we have found no real need to model second-order beliefs in our models of work practice. Using first-order predicates, we are able to create a second-order type of beliefs about others. For example, if Jerry has a belief that Joe believes that John is not trustworthy, we can represent this with the first-order predicate *Believes-Is-Not-Trustworthy*. The belief that Jerry has could then be specified with the first-order belief *Believes-Is-Not-Trustworthy(Joe, John)*.

4.3.2 Logical framework for intentional notion

Let's suppose we want to use first-order logic to represent the following belief:

$$(1) \quad \text{Mary believes Jupiter is the president of Immortals}$$

If we would try to translate this into a well-formed formula (WFF) in first-order logic we might get something like:

$$(2) \quad \text{Believe}(\text{Mary}, \text{President}(\text{Jupiter}, \text{Immortals}))$$

This, unfortunately, is not a correct WFF. First, the second argument of the predicate *Believe* is another formula, and is therefore not a term as is syntactically needed. Secondly, there exists the problem of, what logicians call, *referential opacity*. The standard substitution rules of first-order logic do not hold this problem. For instance, in (2) the term *Jupiter* could be substituted for *Zeus*, since they both denote the same deity:

$$(3) \quad (\text{Jupiter} = \text{Zeus})$$

Following the standard rules of first-order logic, and (2) and (3) we could derive (4):

$$(4) \quad \text{Believe}(\text{Mary}, \text{President}(\text{Zeus}, \text{Immortals}))$$

However, intuitively we know that believing (2) and (4) is not the same, since it might be that Mary does not know (3). In short, substituting equivalents into an opaque context does not preserve the same meaning.

The second problem with first-order logic is that it is *truth functional*. This means that the truth-value of a proposition is solely dependent on the values of its sub-expressions; for instance, the value of the proposition $p \vee q$ is solely dependent on the truth-value of p and q . It is thus said that the operators of classical logic are truth functional. However, intentional operators, such as *Believe* are not truth functional. To understand this, think of the fact that a person might believe a proposition that is *not true*. For instance, John might believe the following proposition:

$$(5) \quad \text{President}(\text{Bill Clinton}, \text{Immortals})$$

John his belief is thus not dependent on the truth-value of this sentence, since this sentence is obviously false. Because of these two fundamental logical problems, classical first-order logic is not enough to represent intentional notions, and we have to define another formalism. The best-known syntactic formalism that might be used is a *modal language* that contains *non-truth functional modal operators*.

However, a third problem needs to be addressed. This problem is a well-known semantic one, and is referred to as the *logical omniscience* problem. Logical omniscience implies that an agent is a perfect reasoner. This means that, an agent that reasons corresponding a logical theory, will deduce *all* beliefs that

can be logically deduced from its rules (this is referred to as *closed under logical consequence*). This is the problem with using Hintikka's *possible worlds model* for the logic of knowledge and belief (Hintikka 1962).

The most common approach for solving these three problems, and an alternative to the possible worlds model, is to use an *interpreted symbolic structure* approach. This approach, proposed by Konolige, is called Konolige's *deduction model of belief* (Konolige 1986). Konolige's approach is based on modeling resource-bounded believers. This fits with our need to model bounded rationality, and addresses the logical omniscience problem. In the deduction model of belief, an agent's beliefs are represented as logical propositions in a data structure associated with the agent. An agent then believes ϕ if ϕ is present in its belief data structure. Using an inference mechanism an agent will use the inference rules whenever possible to obtain *deductive closure* of its base beliefs. In other words; an agent has a set of beliefs represented as logical propositions within its database. Using an inference mechanism, it will always deduce new beliefs based on production rules that have been defined in the agent's knowledge base²³. We have used this approach in Brahms.

4.3.3 Beliefs

Agents and objects in Brahms have beliefs represented as first-order propositions. For instance, suppose agent A1 believes that he is writing his dissertation, and that it will be finished on time. A1 would then have the belief set:

$$\{ \text{BEL}(\text{Is-Writing}(A1, \text{Dissertation})), \text{BEL}(\text{Will-Finish-On-Time}(A1, \text{Dissertation})) \}$$

An agent will always start out with an *initial-belief* set that is defined at the agent's local-level, and the groups of which the agent is a member. Initial-beliefs are assigned in the initialization phase of a simulation. These initial-beliefs define the initial state for the agent. An agent without an initial state could be seen as initially "dumb", or an agent that has not experienced anything yet. As the simulation time moves forward agents will infer, detect and receive new beliefs, either based on their actions in the world or deducing new beliefs, using an inference rule.

To conclude, this model of *human* beliefs is arguably too simplistic to represent all intricacies of human beliefs. However, it should be noted that it is adequate for representing an agent's beliefs for the purpose of AI systems, and I assume therefore that it is also adequate for modeling work practice.

For the syntax of beliefs, see section 11 in appendix A.

4.3.4 Facts

Facts, in Brahms, are factual states of the world. They represent, what we call, a "birds-eye view" of the world. Facts are global to the world, meaning that they can be "seen" by every agent and object in the world.

Konolige (Konolige 1982) defines the *knowledge* of an agent. This definition goes as follows:

$$(6) \quad \forall a, f \text{ KNOW}(a, f) = \text{BEL}(a, f) \wedge \text{TRUE}(f), \text{ where } a \text{ is an agent and } f \text{ is a wff.}$$

This means that knowledge of an agent is a belief that *actually holds in the world*. In Brahms we do not distinguish between beliefs and knowledge of an agent. By definition all beliefs an agent has constitutes its knowledge. However, different from Konolige's first-order formalization of knowledge, in Brahms we want to be able to represent facts that actually hold in the world, but are not believed by an agent, and vice versa. This means that we can represent the facts in the world separate from the knowledge of agents. For instance, although the fact is that the color of my car is red, I believe that the color of my car is green, because I might be colorblind. In representing the context of the agent as facts in the world, we are able to have multiple agents react on the same facts in different ways, dependent on their beliefs about these facts.

²³ The point should be made that the agent only applies rules when they are available to be applied, i.e. when they are a) in the activity scope, and b) their preconditions match the agent's beliefs. Only then the agent deduces the new beliefs. This is a function of the subsumption-like architecture in agent's activity-trees. This architecture is discussed in more detail in 4.4.7.2.

Konolige defines a common fact, CF, as a fact that is known by all agents. This is different from facts in Brahms. In Brahms, it is not necessary that any agent has any knowledge about a fact. Specifically, facts are independent from the knowledge of agents (see Figure 4-1).

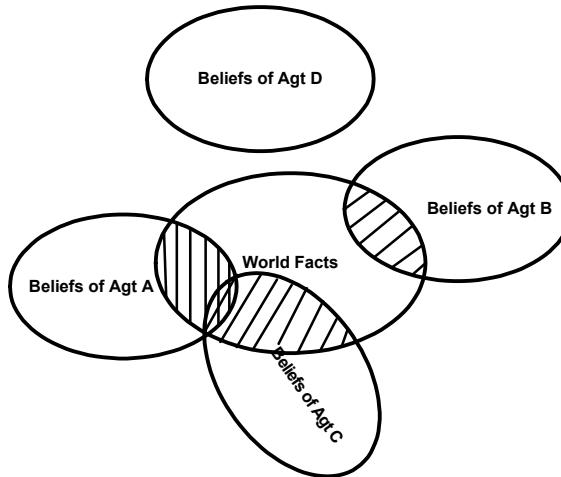


Figure 4-1. Beliefs and facts Venn diagram

For the syntax for facts, see section 12 of appendix A.

4.4 ACTIVITIES AND WORKFRAMES

The notion of *activities* is not strictly new. Shoham proposes that *action* is an intuitive concept, and has importance in common sense reasoning (Shoham 1989). For example, given the initial state in which the light is off, the action of “flipping the switch” creates a new state in which the light is on. This notion of action defines a timing-relation between two states of the world. This same notion captures our intuition that agents make choices about their actions, and intuitively we know that there is a connection between action and choice making.

Another property of action has to do with the level of knowledge the observer has about the observed action. This is closely related to Dennett’s argument that “free-will is in the mind of the beholder” (Dennett 1984). We do not ascribe free will to an agent if we can always predict the behavior of that agent. Shoham suggests that the same can be said for action. If we have detail knowledge, as an observer, about the actions taken by an agent, we might describe this observation as a fixed function performed by an agentless process. Shoham gives the example of our reasoning about a light switch and a dog (Shoham 1989). We understand the workings of a light switch completely, and would view it as described by a fixed transitional function, and not as an agent who receives our request and decides to act upon it by transferring current, et cetera. On the other hand, if we consider a dog, we are much more ignorant about its behavior. The dog’s behavior is too complex to view it as a transitional function, and some would ascertain that the dog does not have such a function, but has free will. In these situations, we make very much use of the abstract notion of action. In the case of modeling human beings and their work activities, it is clear that we are dealing with the latter situation. Human action-behavior over time is too complex to describe as a transitional function, and we are left to describe it in terms of decision-based actions.

4.4.1 Activities versus tasks

The issue discussed in this section is the difference between *task* and *activity*. Most of us who come out of the field of knowledge engineering (KE) and AI have been taught to describe the reasoning-behavior of humans in terms of *goal-satisfaction* and *tasks* (Newell 1990) (Schreiber et al. 1993). To understand the difference we start with two definitions:

1. A *task* is an abstraction of a reasoning process that accomplishes a predefined goal

This means that describing the reasoning behavior in terms of goals, sub-goals, and branch decision-points of the rational agent, describes an agent performing a task. In that case we are not concerned with time, just with the causal relationships that exist between the goals and the decision points.

2. An *activity* is an abstraction of real-life actions that help accomplish a task.

A model of an agent's activities describes what the agent actually does over time (i.e. its behavior) based on decision-points that are described based on the causal relationship between the decision to perform an action and the past and present state of its beliefs.

The premise in DAI is that activities²⁴ themselves do not consume any time. That is, activities are instantaneous. In describing people's real-life activities this presents a problem, since from our experiences we know that each activity in the real world takes time, no matter how short. A person is always within an activity. One cannot be *not* within an activity. The main difference between the two concepts of task and activity is the notion of time. Activities are associated with intervals on a time line. Activities by definition take time, because they represent a person's action in the real world. In the real world, we cannot ignore time, and thus, if we want to describe what people actually do we need a *temporal logic* that allows us to model this.

Wooldridge describes activities as being of only two instantaneous types, communicative and cognitive, that do not take time, although they are related in a temporal fashion on a time line (Wooldridge 1992). In Brahms we change this view of an instantaneous activity, and we represent activities to take time.

The following key points can be made about activities:

1. Reasoning is seen as an activity taking time, and not just an inference or deduction. Thus logical inferences happen *within* an activity.
2. Activities might not involve 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.
3. Modeling behavior involves more than logical inferencing, namely the representation of chronological activities that agents do.

Activities are constrained on their activation by preconditions that are associated with the workframe it is part of. For example, activities may have preferential start times, as expressed in the preconditions, which may refer to the time in hours, minutes, seconds, day of the year, and/or day of the week. Brahms recognizes a variety of temporal constraints, such as *always after* a particular time, *always before* leaving, and *usually* in the morning. Thus, an activity may be interrupted by a scheduled activity, such as going to lunch at noon. Time may change the priorities of behaviors and different people might do the same activities at different times.

4.4.2 Workframes

An agent cannot always apply all its available activities, given the agent's cognitive state and the location or place it is in. Each activity is therefore associated with a *conditional statement* or *constraint*, representing a condition/activity pair, most of the time referred to as a *rule* (Wooldridge 1992) (Shoham 1993) (Konolige 1982). If the conditions of a rule are believed, then the associated activities are performed. In Brahms, such rules are called *workframes*. Workframes are situated-action rules. Workframes are derived from rules in expert systems, but they are different in that they execute activities, and thus take time (see Figure 4-2).

A workframe defines an activity (or activities) that an agent or object may perform. Workframes have conditions, called *preconditions*, that constrain when to carry out the activity. A workframe precondition tests a belief held by the agent executing the workframe (see chapter 4.4.5). A workframe can also contain a detectable. *Detectables* describe circumstances (in the form of *fact-conditions* about the world) an agent

²⁴ In the DAI literature an *activity* is called an *action*. I prefer the word *activity*, but it can be used interchangeably.

might observe while executing the workframe. Detectables could, for instance, create an impasse to completing the activity (see chapter 4.4.9).

GROUPS are composed of
AGENTS having
BELIEFS and doing
ACTIVITIES executed by
WORKFRAMES defined by
PRECONDITIONS, matching agent's beliefs
PRIMITIVE ACTIVITIES
COMPOSITE ACTIVITIES, decomposing the activity
DETECTABLES, including INTERRUPT, IMPASSES
CONSEQUENCES, creating new beliefs and/or facts

Figure 4-2. Taxonomy for groups, agents, beliefs, activities, and workframes

Having two or more agents with different workframes, performing the same activity, can represent individual differences. Individual differences can be modeled by giving different agents the same workframes but different beliefs about the world.

A workframe is a larger unit than the simple *precondition-activity-consequence* design might suggest, because a workframe may model relationships involving location, object resources such as tools and documents, required information, other agents the agent is working with, and the state of previous or ongoing work. Active workframes may establish a context of activities for the agent and thereby model the agent's intentions, e.g., calling person X to give or get information, or going to the fax machine to look for document Y. In this way, behavior may be modeled as continuous across time, and not merely reactive.

Workframes can also be associated with objects. In this case workframes satisfy their preconditions with *facts* rather than with beliefs. Workframes for objects are inherited from object classes as workframes for agents are inherited from groups.

For the syntax for workframes, see section 13 of appendix A.

4.4.3 Consequences

Consequences are statements that are inside a workframe's body. They can be situated before or after activities. *Consequences* are facts or beliefs, or both, that may be asserted when a workframe is executed. They exist so a modeler may model the results of the activities in a workframe. A consequence is formally like a condition and defines the fact or belief that will be created or changed, when executed. The property *fact-certainty* is the probability that the fact will be changed or created; the default value is 100%. The property *belief-certainty* is the probability (with also a default value of 100%) that the belief will be changed or created, conditional on the fact being true. That is, if the fact-certainty and the belief-certainty are each 50%, then 1 in 2 times the fact will be created and 1 in 4 times the belief will be created. If the fact-certainty is zero, then no fact will be created but the belief-certainty determines how often a belief is created.

For the syntax for consequences, see section 22 of appendix A.

4.4.4 Thoughtframes

Thoughtframes define deductions, mostly referred to as *production rules*. Thoughtframes are similar to workframes, but are taken to be *inferences* an agent (or object) makes without doing any activities. Thoughtframes have the same type of preconditions and consequences as workframes. Thoughtframes have no activities, consume no time, and cannot be interrupted. Once the preconditions of a thought frame match the beliefs of the agent or object, its consequences are immediately executed, similar to forward-chaining rules (Charniak and McDermott 1986). An important point is that the preconditions in thoughtframes for objects *always* only match with the beliefs of the object. Another important point is that the

consequences in a thoughtframe can only create new beliefs for the agent or object, and *cannot* create new facts in the world.

For the syntax for thoughtframes, see section 14 of appendix A.

4.4.5 Preconditions

A precondition is a conditional statement that guards a frame (i.e. workframe or thoughtframe). All preconditions must evaluate to true before the frame will become available to be performed by an agent or object. Preconditions of workframes in objects are satisfied when *facts* match, whether or not the object has beliefs about those facts. In other words, objects do not react based on their beliefs, but on the facts in the world.

Brahms has three truth-values for beliefs and facts: *true*, *false*, and *unknown*. If a particular fact-value or belief-value is not present, then it is said to be unknown rather than being false. This allows agents to have preconditions of the form “If I do not know the due-date of order-3 then . . .”. In the *deduction model of belief*, Konolige describes beliefs as having the following meaning; Suppose d_i is the deduction structure of agent i , then *belief* is given the following meaning (Konolige 1986):

$$(7) \quad \phi \in \text{close}(d_i) \Leftrightarrow i \text{ believes } \phi$$

In Brahms this is represented as: **known-value(ϕ) or known(ϕ)**

$$(8) \quad \phi \notin \text{close}(d_i) \Leftrightarrow i \text{ does not believe } \phi$$

In Brahms this is represented as: **not(ϕ)**

$$(9) \quad \neg\phi \in \text{close}(d_i) \Leftrightarrow i \text{ believes } \neg\phi$$

In Brahms this is represented as: **knownvalue(ϕ is false)**

$$(10) \quad \neg\phi \notin \text{close}(d_i) \Leftrightarrow i \text{ does not believe } \neg\phi$$

In Brahms this is represented as: **not(ϕ is false)**

It should be noted that it might be possible that an agent’s belief system satisfies both the conditions (8) and (10). In other words the belief set of agent $A1$ may satisfy the following conditions at some point:

$$(11) \quad \phi \notin \text{close}(d_i) \wedge \neg\phi \notin \text{close}(d_i) \Leftrightarrow i \text{ does not believe } \phi \text{ and } i \text{ does not believe } \neg\phi$$

As Konolige proposes, this is equivalent to *having no opinion* about belief ϕ .

In Brahms this is represented as: **unknown(ϕ)**, which is equivalent to **not(ϕ) \wedge not(ϕ is false)**

In Brahms, we can model conditions (7) through (11) with one of three modifiers: *known*, *known-value*, or *unknown* as *preconditions*. The semantic meaning of the modifiers is described in section 21 of appendix A.

For the syntax for preconditions, see also section 21 of appendix A.

4.4.6 Variables

Variables in a frame make the frame a template for activities (workframe) or reasoning (thoughtframe) that agents and objects may perform. Variables may have quantifiers, as will be described below. The scope of a variable is bound to the frame it is declared in. Variables are inherited from the workframe in which they are declared, by any composite activities and workframes within.

Brahms supports three quantifiers for variables: *foreach*, *forone*, and *collectall*. Variables can be used in preconditions, consequences, detectables, and as parameters for activities. The quantifier affects the way a variable is bound to a specific instance of the defined type (group or class) of the variable.

4.4.6.1 For-each

A *for-each* variable is bound to only one instance, but for each instance that can be bound to the variable, a separate *workframe instantiation* is created. Consider, for example, a precondition and workframe indicating:

```
workframe Do-Work {
    variables:
        foreach(Order) order;
    when (knownval(order is-assigned-to Allen))
    do {
        work-on(order);
    }
}
```

If three Orders are assigned to agent Allen and agent Allen has beliefs for all three of the orders matching the precondition, Brahms creates three workframe instantiations (wfi's) for agent Allen, and in each wfi the for-each variable is bound to one of the three orders. This means that Allen works on all three the orders, *one order at a time*. The *order* in which Allen works on the three orders is underlined.

4.4.6.2 Collect-all

A *collect-all* variable can be bound to more then one instance. The variable is bound to all matching belief-instances, and *only one wfi is created*. Consider the previous example with a different variable declaration:

```
variables:
    collectall(Order) order;
```

In this situation the simulation engine creates *one wfi* and binds the *collectall* variable to a *list* of all three orders. This means that Allen *works on all three orders at the same time*, cutting the actual activity duration in three.

4.4.6.3 For-one

A *for-one* variable can be *bound to only one* belief-instance, and *only one wfi* is created. A for-one variable binds to the first belief-instance found and ignores other possible matches. As far as the modeler is concerned, the selection is random, meaning in the case of multiple matches it is undefined which order is selected. In the previous example workframe, the variable declaration would look like:

```
variables:
    forone(Order) order;
```

In this situation, one wfi gets created, and only one of the three orders gets bound. This means that Allen randomly *works on just one of the orders*, making the activity for one order take the same time as for all three orders together.

4.4.6.4 Unassigned variables

A variable may be declared as *assigned* or *unassigned*. An unassigned variable is unbound (that is, it does not get a value) when a frame instantiation is created; an unassigned variable gets a value through a communicated belief or object creation activity, which binds the variable to a newly created object.

For the syntax for variables see section 28 of appendix A.

4.4.7 Primitive and composite activities

The activities in a workframe are one or more primitive activities, one or more composite activities or both. A composite activity includes one or more workframes, any of which may trigger other composite activities, each with its own workframes (Figure 4-3). Other than a few predefined atomic activities that have semantics, activities are differentiated solely by the modeler's description and use of them (see sections 4.4.7.1 and 4.4.7.2 on primitive activities and composite activities).

4.4.7.1 Primitive activities

Primitive activities take time, which may be specified by the modeler as a definite quantity or a random quantity within a range. However, because workframes can be interrupted and never resumed, when an activity will finish cannot be predicted from its start time. Primitive activities are atomic behaviors that are not decomposed (see Figure 4-3). Whether something is modeled as a primitive activity is a decision made by the modeler. A primitive activity also has a *priority* that is used for determining the priority of workframes.

For the syntax for primitive activities see section 16 of appendix A.

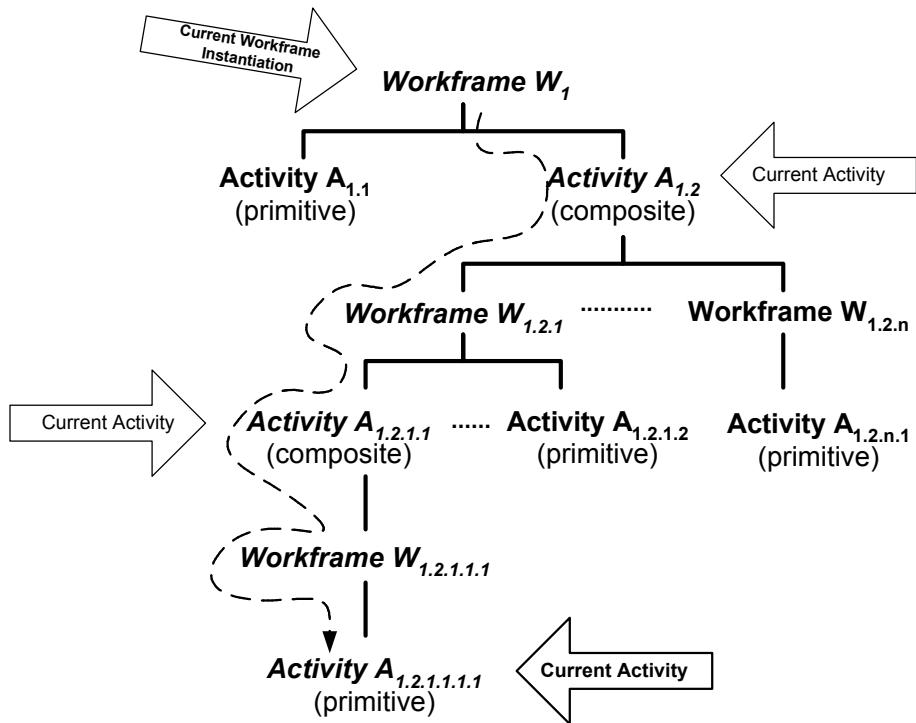


Figure 4-3. Workframe-Activity hierarchy

4.4.7.2 Composite activities

A composite activity expresses an activity that may require several workframes to be accomplished. Since activities are called within the do-part of a workframe, each is performed at a certain time within the workframe. The body of a workframe has a top-down, left-to-right execution sequence. Preference or relative priority of workframes can be modeled by grouping them into ordered composite activities. The workframes within a composite activity, however, can be performed in any order, depending on when their preconditions are satisfied. In this way, workframes can explicitly control executions of composite activities and, execution of workframes depends not on their order, but on the satisfiability of their preconditions and the priorities of their activities (see Figure 4-3).

A composite activity can terminate in the following four ways. First, a composite activity terminates whenever the workframe in which it is executed terminates, due to a workframe detectable of type complete or abort. Second, a composite activity terminates whenever a detectable of type complete or abort is detected within the composite activity. Third, a composite activity terminates immediately whenever an end condition declared within the composite activity is activated. And fourth, a composite activity terminates when the modeler has defined it to be ended “when there is no more work available” and no more workframes in the composite activity are available or being worked on. During the execution of a composite activity, the engine continuously checks whether the agent has received a belief that matches any end-conditions.

For the syntax for composite activities, see section 15 of appendix A.

4.4.8 Built-in primitive activities with semantics

As mentioned before, primitive activities are atomic actions, and a small number of primitive activities are defined to have built-in semantics that is implemented in the Brahms engine. These predefined primitive activities exist to communicate beliefs, create runtime objects, and travel to a location.

4.4.8.1 Create-object activity

Primitive *create-object* activities allow the modeler to create new objects at runtime or to make copies of existing objects dynamically. The modeler can specify when the actual creation or copying takes place during the performance of the activity, by setting the *when*-value to either *start* or *end*. Create-object activities can be used, for example, to model a fax machine creating a new instance of a fax, or a customer creating an order. In addition, in a create-object activity, an object can automatically be connected to a conceptual object or placed at a location.

For the syntax for create-object activities, see section 17 of appendix A.

4.4.8.2 Move activity

Primitive *move* activities trigger an agent or object to move to a location, if not yet located there. For this activity type the modeler defines the goal-location, such as the name of an *area* in the geography model, or a variable referring to the name of an area (see chapter 4.5).

In moving, an agent or object may act as a container for another agent or object that is carried along. For example, a car-object may carry an agent, and then move to a new location. To effect this, the modeler links the carrier and the carried with the built-in *contains* relation, before the move activity is executed. This is done with a consequence that asserts the relation, and then negates the relation with another consequence when the trip is completed, and the carrier “drops” the carried object or agent in the new location.

When a primitive move activity is executed, and the goal-location is different from the agent’s or object’s current location, the agent or object will start moving to the goal location. The simulation engine finds a path between the locations and gets or computes the distance. It is possible, however, to define the duration of the activity and thus avoid the need to define a geography model with travel paths. The engine calculates the duration of the trip and uses it to set the duration of the primitive move activity. When the agent or object reaches the new location, a new fact in the world and belief for the agent are created stating that it is there. The agents currently at the new location detect the agent or object and will therefore also get a belief about its location. A newly arrived agent will also detect the other agents and objects in the new location. The agent or object then continues with the workframe.

Brahms can handle interruptions that cause the location of an agent to change. Work that has to be done at a specific location may be interrupted and the agent may then move to another location to do work of a higher priority. When the higher-priority work is completed, before the agent resumes the interrupted work, the agent returns to the location where the agent has to do the work.

For the syntax for move activities, see section 18 of appendix A.

4.4.8.3 Communication activity

The predefined primitive *communication* activity transfers beliefs to/from agents or objects. An agent can give (send) and request (receive) beliefs. One can think of the agent-to-agent and agent-from-agent communication primitives as modeling a simple conversation. Agent A can *ask* agent B to tell him anything B knows about subjects X (From B), and likewise, can *tell* agent B anything that A knows about subjects Y (To B). In either case, beliefs must be specified in so-called transfer-definitions. In the first case, it specifies what beliefs will be transferred from the “From” agent or object. In the second case, it transfers the beliefs to the “To” agent or object. Only the agent or object’s beliefs that match the specified beliefs in the transfer-definition are transferred.

A belief specified in a communication activity is deemed to match another belief under the same conditions that a workframe *known-type* precondition is deemed to match a belief. The specified beliefs are transmitted only if they are actually held by the agent or object. In other words, an agent or object has to have the belief before it can communicate (i.e. tell) the belief to another agent or object. The transmitted beliefs overwrite any beliefs the recipient might have about the same object-attribute or object-relation.

Beliefs transferred to or from an object, model information stored in or on the object. For example, a modeler can use a communication activity to model the reading of information from, or the writing of information to, for example, a fax, paper, bulletin-board, or a computer system. If transmitted beliefs contain variables that remain unbound in the recipient-initiator’s workframe, then those variables are bound from matching beliefs supplied by the sender-responder.

For the syntax for communication activities, see section 19 of appendix A.

4.4.8.4 Broadcast activity

Primitive *broadcast* activities work like communication activities. Here, however, the acting agent is broadcasting the matching beliefs to *all* other agents in the same location as the acting agent. One can think of the broadcast activity as modeling an agent shouting information to other agents in the same location.

When an agent broadcasts, the agent transmits beliefs to all other agents in the *same* geographical area (location) if the agent has a location, or to all other agents if the agent has no location. If an object broadcasts, the object most likely transmits a belief about itself (e.g., a phone ringing), which will be received by the agents in the same location if the object has a location, or by all agents if the object does not have a location.

For the syntax for broadcast activities, see section 20 of appendix A.

4.4.9 Detectables

A *detectable* is a mechanism by which, whenever a particular fact occurs in the world, an agent or object may notice it. The *noticing of the fact* may cause the agent or object to stop or to finish the workframe. A detectable is defined in a workframe (a continue-, abort-, complete-, or impasse detectable), or in a composite activity (an end-activity detectable)

Two things occur in a detectable. First, the agent or object detects the fact and the fact becomes a belief of the agent or object. Second, *only* in the case of an agent, the beliefs of the agent are matched with the *condition* used in the detectable, and if there is a match the then-part of the detectable is executed, which may *abort* or *interrupt* the workframe. For objects, there is only the second step, in which facts in the world are matched with the detectable condition. If there is a match the then-part is executed. For agents, step one and two are independent: Whether or not the fact is present in the world, the condition in the second step is tested against the belief of the agent. For example, if “the color of the telephone-1 is blue” is a fact, and a workframe contains the following detectable condition, “the color of the telephone-1 is red”, in the first step an agent will obtain the belief “the color of telephone-1 is blue”. In the second step, “red” would be compared with “blue” and the condition will *fail*, so the then-part of the detectable would *not* be executed.

The *action* or *then-part* of a detectable defines the *detectable type* and is one of five keywords: *continue*, *abort*, *complete*, *impasse*, and *end-activity*. *Continue* is the default: the agent or object detects conditions, but the workframe proceeds unaffected. With *abort*, a condition causes the agent or object to stop executing the workframe. With *complete*, a condition allows the agent or object to only perform the remaining consequences of the workframe, without doing the rest of the workframe's activities. With *impasse*, the condition prevents the continuation of the workframe *until* the condition is removed. In this case, the workframe goes into the *interrupted-with-impasse* state (see chapter 4.6.2.2). *End-activity* is only meaningful when the detectable is in a composite activity: It causes the activity to be terminated immediately, based on matching the beliefs of the agent or object to the detectable condition. This allows an agent or object to abort working on composite-activities.

It is worth emphasizing that the detectable mechanism is operative for all workframes on the execution path of the agent or object's workframe-activity hierarchy (see Figure 4-3). For example, if there would be a detectable in workframe W1 in Figure 4-3, then this detectable is operative during the performance of *any* of the workframes and activities underneath it in the hierarchy. This even holds for detectables in workframes and activities that are in an interrupted or impasse state, so that a "detect whenever" detectable can detect a fact at any time.

Detectables can also be used to model *impasses*. A common example of an impasse is the case of inaccurate or missing information. Workframes may be written to handle impasses. For example, if a supervisor wants to call a technician but does not know the technician's telephone number, the current workframe can be impassed—with an *impasse* detectable—and another workframe may lead the supervisor to look up the number.

With a detectable, an agent may notice *passive observables*, as when someone shouts, a fax machine beeps, or an agent is present vying for attention. Passive observables fall into two general classes: *sounds* and *visual states*. Objects that cause a sound—fax or phone—create the fact that represent the sound, which can then be detected. Sounds may persist over many simulation clock-ticks. Propagation into the surrounding space will recur as long as the object is making a sound. Propagation may be affected by geography.

For the syntax for detectables see section 23 of appendix A.

4.5 GEOGRAPHY

We live in a three-dimensional geographical space. Social interaction, collaboration, and people's activities are very much dependent on the geographical environment in which it takes place. According to Latané's Theory of Social Impact, (Latané 1981), the distance between two individuals determines how much impact they exert on each other. Spatial relations provide a major constraint on work practice (Nowak and Latané 1994). Work is done in a 3-dimensional space, and it is therefore that we need to include a representation of the spatial relations between agents and artifacts in the environment.

In most DAI systems and agent theories spatial relation is left out. In the ACTS theory (Carley and Prietula 1994a) there is no social axiom that states that an agent has a location in space, nor that actions are performed within a geographical environment. In DAI the focus is on coordination. DAI is concerned with optimizing communication and with task- and resource allocation among intelligent cooperative agents, and less concerned with the spatial relationships within the environment. Both Brooks and Clancey describe new models of intelligence based on the fact that agents are *situated* in the world (Brooks 1991) (Clancey 1997a).

In Brahms, we deal with the emergent behavior of agents in the real world. In Brahms models, agents and objects are situated in a *description* of the physical world. We therefore need to be able to represent this physical world independent of the reasoning capability of agents about space. In the next sections, I describe the Brahms language capabilities of representing a geographical space. The geography design is

described in more detail in (Steenvoorden 1995)²⁵. The design of the geography model is much inspired by the computer game SimCity²⁶.

The current Brahms system includes only a partial implementation of this design, due to time constraints and resource limitations. However, I include a description of the design for completeness. As we continue to develop Brahms, we will run into modeling situations where a more complete implementation is needed.

4.5.1 Geographical objects

Geographical primitives allow the modeler to model locations in the physical world, as well as the relationships between locations. The scope of the geography is limited to an area. Connectivity between areas is established and displayed through connectivity relationships, rather than graphically.

Graphically, an area has an area map-grid. Level-1 objects are those that can be placed in the area map-grid. A level-2 object is one that can be placed on or in a level-1 object. A level-3 object is one that can be placed on or in a level-2 object, and so on. A level can be looked upon as a view in a geographical space. When a user zooms in on a level-3 object, the user sees the level-4 objects that are placed on or in the level-3 object.

An area map is a 3-dimensional grid (X, Y, Z). The grid has a scaling factor, so distance and travel time can be indicated. Travel time depends on distance and means of transportation and may also be affected by the elevation angles of road, train tracks, and other paths.

Geographical objects generally fall in one or more of the following groups: areas, transportation-related elements, general elements, and artifacts. An area is a super-class for buildings and pseudo-buildings. An area can have zero or more instances of “exit-entrance”. An exit-entrance leads to a path-part, such as part of a street, railroad track, or river.

4.5.1.1 Level-1 objects

For example, *buildings* are level-1 objects. They are composed of floors and do not have grids. By default a building has one floor. Buildings are placed alongside paths, such as roads, waterways, and train tracks. The exit or entrance of a building has a relation with one of the path-parts that are positioned alongside the building. A *pseudo-building* is a level-1 object, but does not have floors or rooms. These could be parks, parking lots, et cetera.

4.5.1.2 Level-2 objects

Floors in buildings are level-2 objects that are composed of rooms and have a 2-dimensional grid. A floor has a minimum of one default floor-room.

4.5.1.3 Level-3 objects

Rooms on floors are level-3 objects. A room has a 3-dimensional grid. To allow modeling a window that does not reach from the ceiling to the ground, or an artifact like a white board hanging on a wall, the Z-coordinate of a room has a minimum value of 2.

4.5.1.4 Level-4 and 5 objects

An artifact inside a room is a level-4 or level-5 object that can be placed anywhere, subject to restrictions related to the nature of the artifact. If an artifact has a grid, another artifact can be placed on top of it.

²⁵ Edgar Steenvoorden, Maarten Sierhuis, Ron van Hoof, Dave Torok and Bill Clancey designed the geography.

²⁶ SimCity is developed by Maxis.

4.5.1.5 Transportation

Transportation-related elements can roughly be divided into the following categories: vehicles, transportation-related buildings, and transportation paths. A transportation path is a level-1 object that is a route or surface on which agents and objects can move. It has at least one connecting path-part and can be one of the following types: aerial route, waterway, rail, pedestrian path, road (for automotive vehicles), and don't-care. A modeler not interested in simulating transportation can use don't-care paths, which are direct connections.

Transportation-related buildings can be used as a space or connection to move from one transportation path to another. A transportation-related building connects two or more path-parts not necessarily belonging to the same (kind of) path, without a direct physical connection. Transportation-related buildings include airports, train stations, subway stations, harbors, and tram or bus stops.

Vehicles are level-2 objects. Vehicles are container artifacts that can contain agents and objects, but vehicles can have limited cargo capacity. Vehicles can be divided into the following groups: automotive, water, air, rail-based, and don't-care vehicles.

General geographical elements are level-1 objects that enable a modeler to create a geographical landscape that resembles an actual landscape. Such elements may include bodies of water, which can function as paths or as decorative obstacles; hills, which can influence the speed with which certain vehicles can travel; and obstacles such as trees and bushes.

4.5.1.6 Movement

Agents and objects can move through the entire geography. An agent or object has the functionality of a container artifact and can carry other objects. The number of artifacts an agent is able to carry depends on the artifacts' weight and the nature of the artifacts. An agent can also be placed inside an artifact, such as a car.

Whenever an agent or object enters a geographical location, such as a room, the simulation engine does the following: it generates a fact that the agent or object is in that location, it generates a belief for that specific agent that it is in that location, and it generates that same belief for all other agents in that location. The geography may be implemented to limit the generation of beliefs to other agents in the immediate environment of the specific agent.

When an agent or artifact moves from one location to another location the simulation engine calculates the shortest route between the two locations. When in route, an agent can detect other agents and objects that it passes. In this way an agent can stop on his way to a location and talk to another agent if necessary.

4.5.2 Implementation in Brahm

The implementation of the geography design is limited in the current Brahm language and simulation engine, however it is such that the modeler can represent any type of geography model, from cities to rooms, and from spacecarts to craters on the Moon.

An *areadefinition* is used for defining area instances used for representing geographical information in a model. Area definitions are similar to classes in their use. Examples of area definitions are "Building", and "City".

An *area* represents a geographical location and is used to create a geographical representation for use in the model. An example is the area 'NewYorkCity'. An area is an instance of an area definition. Areas can be decomposed into sub-areas. 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. Using this relationship areas can be defined as level-1, 2, and 3 objects (as described in the previous sections). In other words, areas can be *part-of* other areas.

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, over the path. The automatic generation of location facts and beliefs for agents and objects moving from one area to another is also implemented. However, moving agents and objects do not notice other agents or objects on their path, and cannot stop to communicate when moving. This limitation of the current implementation means that we cannot model agents communicating while moving.

For the syntax for areadefinition see section 24 of appendix A.

For the syntax for area see section 25 of appendix A.

For the syntax for path see section 26 of appendix A.

4.6 SIMULATION

In this section I present the model of execution for a Brahms model. The model of execution defines how a Brahms model is executed, and thus describes a simulation of a model of work practice. As a multiagent system, a Brahms model consists of a number of agents and objects that operate independently, but interact with each other. Wooldridge (Wooldridge 1992) describes two possible execution models for multiagent systems, *synchronous execution*, and *interleaved execution*. In both cases the execution of a multiagent system is defined by a *state* σ_t of the system at time t , and a state σ_{t+1} of the system at time $t+1$ caused by a *state-transition* τ_t at time t . Keeping track of the state changes of the system over time the *history* of an executing system can be considered a sequence of state and state-transitions.

4.6.1.1 Synchronous execution

In a synchronous execution system each agent and object²⁷ has an initial state defined as its *initial belief set*, closed under its workframes and thoughtframes. This amounts to an initial state of the system as a collection of initial belief sets for each agent.

Agents are able to change their state by performing a *move*. A move is defined as a tuple of actions. A *transition* is a collection of moves, specifically, one for each agent. In other words, a move is a state-transition for an individual agent, whereas a transition is the global state-transition for the whole system (i.e. all the agents and objects and facts). The operation of a synchronous execution system as defined by Wooldridge goes as follows (Wooldridge 1992, p.60):

A system has a defined initial state (call it σ_0). From this state, each agent picks a (legal) move, which combines with those of others to form a transition, τ_1 . As a result of this transition, a new state σ_1 results. The whole process then begins again, with agents choosing moves that form transition τ_2 , and so on. The result is a sequence of states.

Next, there is the concept of a *simulation run*, and of the *history of a simulation run*—referred to simply as the *history*. The collection of the states and transitions that caused the state is called a *world*. A simulation run is now defined as a *sequence of worlds*. To capture the history the system needs to capture more than the changes of the belief sets of the agents. It also needs to capture the reason for the changes in the belief sets. This is accomplished by capturing all agent and object activities, as well as the changes in the belief sets.

The operationalization of this within Brahms is as follows: Each agent has a set of beliefs that can be changed over time through consecutive *moves*. Every execution step (i.e. a simulation clock-tick) all the moves of all agents and objects create a *transition* in the total simulation model. Each moment of a transition is called a *world*, and also includes the state of all facts in the world. A simulation run is then a sequence of *worlds* that are created every clock-tick. Saving the creation of *worlds* over time, Brahms creates a *simulation history*. This history can be inspected after the simulation.

²⁷ In the rest of this chapter, wherever I say “agent” you should read this as “agent and object”, unless otherwise specified.

4.6.1.2 Interleaved execution

In a distributed multiagent system there is no global clock that synchronizes all the agents. This type of execution is used in what I call a *real-time multiagent* system. As Agha points out in (Agha 1986, p.9):

The concept of a unique global clock is not meaningful in the context of a distributed system of self-contained parallel agents.

In real-time multiagent systems it is not possible to meaningfully have all agents' transition to their next state at the same clock time. The biggest difference between a synchronous and an interleaved execution model is that in the interleaved model only one agent may act in a transition at any one time. This brings with it an extra difficulty in synchronizing the messages being sent and received between agents. This introduces the necessity to keep track of all those messages that have not yet been received. This is done through a *message queue*. Thus, an interleaved execution model has an added complexity of controlling the synchronization between agents and their states.

4.6.2 Multiagent simulation in Brahm

In this section I define how the execution model of Brahm works. In Brahm we use the *synchronous execution model*. The reason for this is simply the fact that we simulating and not running in real-time. In our simulation model our agents and objects need to be synchronized according to a unique global clock^{28,29}.

A *world* is the situation-specific model (SSM) of the simulation, at a specific moment in the execution of the system (Clancey 1992). A *state* is defined as the belief-set of an individual agent, at a specific moment in the execution of the system. And, a *situation-specific model* for an agent is defined by all the existing global facts, and all the agent's beliefs at the moment of inquiry, as well as the current-, available-, and interrupted workframe and thoughtframe instantiations, and the current activities.

A *state-transition* occurs when an agent, performing a work- or thoughtframe, executes a consequence that creates a new belief or fact. A state-transition can also occur when an agent receives a new belief as a result of a) a communication, b) the detection of a fact, c) a move to a new location. During the state-transition the simulation engine determines the effects of the transition on the agent's internal state, which can result in more transitions.

4.6.2.1 Frame execution

An order of testing and execution must be imposed in any simulation tool on conditions and operations that in principle apply or occur simultaneously. The following paragraphs describe the order in which the parts of a workframe are evaluated and executed in Brahm.

For each agent the preconditions are the first things checked in a frame (workframes and thoughtframes). They are checked in the order in which they are declared within the frame. When all of its preconditions match (i.e., are satisfied), a frame becomes available. When a frame becomes available *frame instantiations* are created for each set of variable bindings from the precondition matching. If a frame has multiple variables that can be bound, there will be a frame instantiation created for each valid combination of variable-bindings. Each frame instantiation is executed in sequence (i.e. one after another). There can only

²⁸ The issue of a distributed event-driven architecture for agents and objects is an implementation issue. Although it has impact on the efficiency and applicability of an AOL, it is not of concern in this thesis, and will therefore not be discussed. It should be noted that the old G2 implementation of the Brahm simulation engine is *not* distributed and is clock-based. The new Java Brahm simulation engine implementation is a discrete event-driven system. This re-implementation resulted in at least a 100x speed increase for a simulation run. This is due to the speed of Java in relation to G2, but more importantly because of the move from a clock-based approach to a discrete-event approach.

²⁹ The new Java engine also implements an interleaved execution mode. In this mode there is no central simulation clock, and each agent runs as fast as possible. This mode is used for implementing real-time agent systems with Brahm. However, this is not the topic of this thesis, and will therefore not be further discussed.

be one frame instantiation executed at a time (in one clock-tick). The order of the sequence is undetermined.

After the preconditions match and a *workframe*³⁰ is selected it will start to work (one frame instantiation for each set of valid variable-bindings). The working time will be specified in the workframe; or, if the workframe contains any composite activities, the working time will be the cumulative time of the executed composite activities. At any time during this working time, a variety of things may happen. Consequences may be asserted, facts may be detected, and communications may occur, depending on their ordering in the do-part—the body—of the workframe. If the body includes one or more move activities, the agent will go to the specified locations as the moves are executed.

Within a detectable, the modeler can specify when the agent or object can detect a fact. When a workframe contains a composite activity, the modeler *must* specify the time to be "whenever", because the engine cannot calculate the total working time for the frame in advance.

When multiple detectables are declared within a *workframe*, they are checked in the order in which they are declared. When two detectables are specified to be executed at the same time, and the first states that the frame should be interrupted and the second states that the frame should be aborted, the frame will be interrupted.

The body of a frame is ordered, and the simulation engine evaluates the body components in the order in which they appear from top-to-bottom and left-to-right. The body may include activities and composite activities for workframes, and consequences that will be asserted as beliefs and/or facts for workframes and thoughtframes.

4.6.2.2 Frame states and transitions

As described above, frames are stateless and serve as declarative definitions, whereas *frame instantiations* are dynamically created, associated with a particular agent or object, have state, and have a related context.

The possible *states* of a frame instantiation are set forth in Table 4-1.

Table 4-1. Frame instantiation states

not-available	No instantiation exists for a given (frame, agent or object, context) set. Either the preconditions of the frame have no matches, or previously active instantiations have all completed and been reset with no matches. This is more or less the start-state of every frame instantiation.
available	The preconditions of the frame have been satisfied for some context and agent or object, but the frame instantiation has not yet been started by the agent or object.
working	The agent or object is performing this frame instantiation for the current clock-tick.
interrupted	The workframe instantiation has already had at least one clock-tick worked on it, but the agent or object is performing some other workframe instantiation during the current clock-tick. Thoughtframe instantiations cannot be in this state.
	A detectable has caused the agent or object to have an

³⁰ The next parts are limited to workframes, because thoughtframes do not take time, and do not have activities and detectables in them.

interrupted-with-impasse	impasse with the workframe instantiation. The workframe instantiation cannot continue until the condition causing the impasse is resolved. Thoughtframe instantiations cannot be in this state.
done	The agent or object has completed all the activities in the frame instantiation. If the reset-when-done attribute of the associated frame is false, then the frame instantiation will exist in the done state. Otherwise, the preconditions will be evaluated and the frame instantiation will become either available or not-available (i.e., deleted).

Given these possible frame states, there are a number of different allowable state-transitions for frame instantiations. These are shown in Figure 4-4.

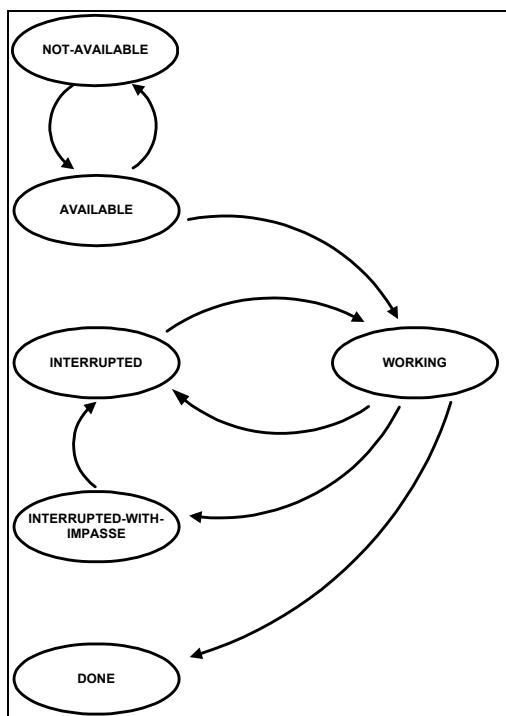


Figure 4-4. State-transition diagram for frame instantiations

The allowable state transitions, shown in Figure 4-4, are listed in Table 4-2, with their causes and implications.

Table 4-2. Frame state-transitions

not-available \Rightarrow available	When the preconditions of a frame are satisfied for a particular agent or object and context, then a frame instantiation is created and put in the state <i>available</i> . This frame instantiation can then be worked on by the agent or object.
available \Rightarrow not-available	If an available frame instantiation has not been started, and the preconditions (which were previously satisfied) become unsatisfied, then the frame instantiation is deleted, and thus becomes <i>not-available</i> .

available \Rightarrow working	If the frame instantiation becomes the current-work of an agent or object, then the frame instantiation has state <i>working</i> .
working \Rightarrow interrupted	Whenever a different workframe instantiation becomes the current-work of an agent or object, the (previous) working frame instantiation becomes <i>interrupted</i> . Note that the agent can choose from the union of the sets (available, working, interrupted) for the current-work for the next clock-tick. This resolution mechanism works on the basis of priorities of the workframe instantiations.
working \Rightarrow interrupted-with-impasse	This state change happens when the following conditions are all met: (1) an agent detects a fact, (2) the current working workframe instantiation contains a detectable that references that fact, (3) that detectable is satisfied, and (4) the type of the detectable is impasse. The agent cannot continue working on the workframe instantiation until the impasse is resolved, i.e. the detectable condition becomes false due to a change in the beliefs of the agent or facts for an object. The workframe instantiation is set to <i>interrupted-with-impasse state</i> .
working \Rightarrow done	When all activities of the frame instantiation are completed after the current clock-tick, then the frame instantiation becomes <i>done</i> , iff the reset-when-done attribute of the associated frame is false. Otherwise, the transitions working \Rightarrow available, depending on whether the preconditions are still being satisfied. However, this is accomplished by creating a new frame instantiation. When <i>done</i> , the frame instantiation is deleted.
interrupted \Rightarrow working	When the agent or object picks an interrupted workframe instantiation to become the current-work for the next clock-tick, the frame instantiation becomes <i>working</i> again.
interrupted-with-impasse \Rightarrow interrupted	When a belief of an agent or fact for an object causes the detectable, that caused an impasse, to be no longer satisfied, the impasse is removed—the belief causes the detectable-condition to no longer match the current beliefs of the agent or fact in case of an object. The frame instantiation can be worked on once again, so the state is changed from interrupted-with-impasse to <i>interrupted</i> , after which, in the next clock-tick the frame instantiation could transition to a working state.

4.6.3 Deciding what to work on next

To decide for each agent what to work on next, the simulation engine executes a number of steps. At each clock tick, the simulation engine determines which workframe should be selected to work on next. This selection is based on the *priorities* of available, current and interrupted workframe instantiations. A *current workframe instantiation* is selected in preference to interrupted or available workframe instantiations of equal priority, so that an agent tends to continue doing what it was doing.

The selected workframe is then executed, leading the agent to possibly detect things in the world (through detectables) and begin an activity. When a workframe instantiation is interrupted, it is reexamined on subsequent clock-ticks to see whether it should be considered for selection. When a composite activity is terminated, because its end-condition is satisfied, the workframe instantiations *below* it are also terminated. When an activity is interrupted, Brahms saves the workframe/activity-hierarchy so the context can be reestablished after an interruption.

The questions remain; 1) How does a workframe get selected to become instantiated, and 2) When multiple workframes are instantiated, how does the engine determine the priority of a workframe instantiation?

The answer to the first question is that at every clock-tick the simulation engine checks if any of the preconditions of the agent's frames are satisfied (i.e. match with beliefs in the agent's belief-set)³¹. When all preconditions in a frame match, the frame is instantiated and each frame instantiation is set to the *available state*. At that moment, the engine includes the frame instantiation in its decision to determine what frame instantiation to work on next.

The answer to the second question defines what workframe instantiation to work on next. Each workframe instantiation has a *priority*. The priority of each workframe instantiation is set based on the priorities of the *primitive activities* in the workframe. The priority of a workframe is the priority of its *highest priority* primitive activity.

Thus, all in all, the *emergent behavior* of agents during a simulation depends on two independent variables. First, it depends on when preconditions of frames match on the belief-set of the agent. Of course, the belief-set of an agent depends on many factors during a simulation, such as detection of facts, moving to locations, communication with others, as well as reasoning. This means that the behavior of an agent is first and foremost dependent on the behavior of other agents and objects in its environment, as well as the state of the environment itself. Secondly, the behavior of an agent depends on which frames are instantiated together at any moment in time. This is because each instantiated frame has a specific priority, and it will depend on the priority of the other frame instantiations whether a frame instantiation is picked as the next work to be done.

4.6.4 Multi-tasking agents

In a Brahms simulation, an agent may engage in multiple activities at any given time, but only one activity in one workframe is active at any one time. At each clock-tick the simulation engine determines which workframe should be selected, based on the priorities of available, current and interrupted work (see previous section). The state of an interrupted or impasssed workframe is saved, so that the agent will continue an interrupted workframe with the activity that it was performing at the moment it got interrupted.

An important consequence and benefit of this paradigm is that all of the workframes of a model are simultaneously competing and active, and the selection of a workframe to execute is made *without* reference to a stack or tree of workframe execution history. This *subsumption architecture* (Brooks 1991) is an important difference between Brahms and most other goal-oriented problem-solving systems, such as Soar (see section 2.2.1). In Soar, an agent has one goal-stack. Switching to another goal-tree means popping goals of the stack, and pushing new ones on it. Consequently, the agent cannot automatically return to the execution of a previous goal-tree that was popped on the goal-stack. This is a serious limitation of the Soar architecture in simulating multi-tasking using goal (or activity) context switching. In Brahms, activity-context switching is inherent in the subsumption architecture.

³¹ The speed at which this is done is heavily dependent on the implementation. In the old G2 engine this was done using a loop-structure, where for every agent and object, all preconditions for all frames are checked at every clock-tick. In the new Java engine, we have invented a multiagent version of a Rete-like algorithm. We call this a Reasoning State Network (RSN). This allows the engine to only check those preconditions in frames that can potentially match with beliefs that just changed in the agents' belief-set.

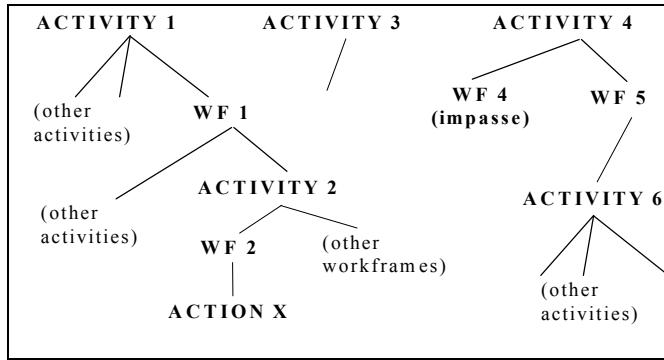


Figure 4-5. Multi-tasking in Brahms

An illustration of this is given in Figure 4-5. An agent (not shown) in a running model may have multiple competing general activities in process: Activities 1, 3, and 4. Activity1 has led the agent (through workframe WF1) to begin a subactivity, Activity 2, which has led (through workframe WF2) to a primitive activity Action X. When Activity 2 is complete, WF1 will lead the agent to do other activities. Meanwhile, other workframes are competing for attention in Activity 1. Activity 2 similarly has competing workframes. Priority or preference rankings led this agent to follow the path to Action X, but interruptions or reevaluations may occur at any time. Activity 3 has a workframe that is potentially active, but the agent is not doing anything with respect to this activity at this time. The agent is doing Activity 4, but reached an impasse in workframe WF4 and has begun an alternative approach (or step) in workframe WF5. This produced a subactivity, Activity 6, which has several potentially active workframes, all having less priority at this time than WF2.

The Brahms subsumption architecture allows two forms of multi-tasking. The first form is inherent in the activity-base paradigm; Brahms can simulate the reactive situated behavior of humans. An agent's context forces it to be *active* in only one low-level activity. However, at any moment an agent can change focus and start working on another competing activity, while queuing others. In Brahms having the simulation engine switch between current and interrupted work for each agent, simulates this type of multi-tasking behavior. The second form is subtler. People can be working concurrently on many high- and medium-level activities in a workframe-activity hierarchy. While an agent can only execute one primitive activity in the hierarchy at a time, the agent is concurrently within all the higher-level activities in the workframe-activity hierarchy. For example, the agent (not shown) in Figure 4-5 is concurrently within Activity 1, Activity 2, and primitive activity Action X. It should be noted that while a workframe, and its associated activities are interrupted or impassed, the agent is still considered to be in the activity. The activity duration time is still accumulating when interrupted or impassed. Therefore, the agent is conceptually within all current, interrupted and impassed activities.

This concludes the description of the Brahms language, as well as Part 1 of the thesis. In Part 2 of the thesis I will describe three case studies in which I show the use of the Brahms language and simulation engine. The more formal description of elements and features of the Brahms language and simulation engine in this chapter will be explained with examples from the case studies in Part 2.

PART 2

Applications

5. RESEARCH DESIGN

In this chapter, I describe my research design. A good research design is a step-wise plan to answer the research question. I first describe the research methodology and then describe ways of using computational modeling as an analysis and design method for systems. Then, I am able to introduce the cycles in my research process. Based on the research cycles, I define the case studies that are the foundation of each of the cycles, and are examples of the use of Brahms as a computational modeling and simulation environment for work practice.

5.1 SCIENTIFIC METHODOLOGY

There are many variations on the specifics of a scientific methodology, but they all share the common theme of scientific understanding of the world. In pursuing a scientific understanding of a problem, the outcome of the research must make sense and correspond with observations. While scientific methodology may vary across disciplines, most include the following characteristics (Zimmerman and Muraski 1995):

- Understanding of the problem
- Stating the problem
- Collecting data
- Analyzing data
- Interpreting data
- Drawing conclusions

5.1.1 Understanding the problem

In the introduction chapter I stated the problem I am addressing in this thesis. I restate the problem in the next section and operationalize it further. Understanding the problem, as Zimmerman describes, is key to being able to state the problem correctly, and is one of the most important components of research (Zimmerman and Muraski 1995). It may take years of disciplined study of a particular domain before an adequate understanding has been accomplished. It is obvious that the understanding of the problem will change as time goes on. A deeper understanding of the problem will most likely change the way the researcher states the problem. Research uses a cyclical process as its modus operandi for scientific understanding (Figure 5-1).

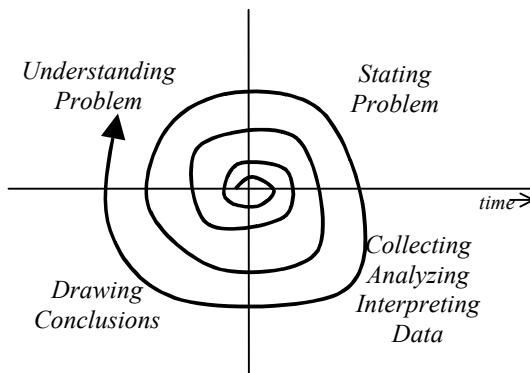


Figure 5-1. Research Process

5.1.2 Restating the problem

As stated in the introduction chapter, the main drive for starting this research was because of our experience in trying to model the work practice of an organization using a workflow-modeling paradigm. A workflow model is a particular view of the work. It is an abstracted functional view of how *conceptual* objects flow through a sequence of tasks in an organization. After a long effort we came to understand that, although useful, a workflow model typically omits people's collaboration, "off-task" behaviors, multi-tasking, interrupted and resumed activities, informal interaction, knowledge, and the environment, i.e. the geographical location and placements of people and artifacts. I am seeking to understand how to model and simulate work practice in such a way that we can include these aspects.

The deeper understanding I am interested in is the question of how work really gets done and how to make a formal representation of this, in the form of a computational model. The creation of a formal representation of a system results in deep understanding of the system. Making a computational model allows us to test and refine this understanding by comparing the results of a simulation with reality. In this sense, simulation is a research method for understanding the system.

In order to use simulation as the method for understanding the work practice of an organization, we need a scheme to represent work practice, and a computational paradigm to simulate a model developed using the representational scheme. The research problem is:

How can we model an organization's work process in such a way that we include people's work practice?

This leads to the main subsidiary question (restated from chapter 1):

1. How can we model an organization's work practice in such a way that we include people's collaboration, "off-task" behaviors, multi-tasking, interrupted and resumed activities, informal interaction, knowledge and geography?

One answer to this question is to develop a modeling language and simulation program in which we can represent the way an individual or group of individuals work—i.e. their practice. This leads to further related questions:

a. What is meant by the concepts in the question stated above?

Ethnographic fieldwork in the workplace (Sachs 1995) has shown that in looking at the way people work in practice we see a number of important aspects:

- (1) People collaborate with each other to accomplish what they have to do.
- (2) People often work on seemingly non-task related things, called "off-task" behaviors.
- (3) People often work on more than one task at the same time, called multi-tasking.
- (4) People are often interrupted in their activities, and will resume what they were working on, after the interruption is over.
- (5) People have many interactions with others that were not planned before hand, and/or not part of the task at hand, called informal interactions.
- (6) People use their domain knowledge, as well as their social knowledge about the organization and the culture to perform their daily work activities.

- (7) The environment is for most part a given. People are always situated in a three-dimensional space. Most of the time, people cannot change the work environment, and they can never ignore the constraints that the environment places on their activities.

Next, we need to operationalize these concepts. That is, to put them into a form in which they can be subject to testing by experiment (Preece 1994, p. 188). This leads to the following *operational questions*:

b. What interpretation should be placed on these aspects of work practice?

In other words: How can we model them?

c. How can these aspects be included in a computational modeling language?

In other words: What formal language can we create that makes it possible to simulate?

To reiterate what the presented work is, and maybe more importantly what it is not, the following note should be taken into account while reading the next sections; This research is about *finding a modeling and simulation paradigm*. It is not about researching a work practice of a specific organization. My hypothesis is that Brahms implements such a paradigm. Therefore, the research method used to test Brahms is not to use the Brahms simulation environment to get a deeper understanding of a particular work practice domain. Instead, the research goal is:

Test if Brahms is a suitable modeling language and simulation environment for modeling and simulating work practice.

However, to do this I need to show the use of Brahms in real world domains. It turns out, not surprisingly, that in the process one cannot avoid getting a deep understanding of the chosen domain.

5.1.3 Scientific method and data collection

There are a variety of research methods and data collection techniques. As a valid research method for testing qualitative modeling approaches, I test Brahms by performing a number of case studies. We can accept the hypothesis if it can be shown in a reasonable number of real world examples that we can simulate work practice with Brahms. Therefore, the research method I use is a *qualitative analysis of a number of empirical case studies showing different uses of Brahms*. The question “how many case studies is a reasonable number?” is the discussion in the next section.

5.2 USE OF COMPUTATIONAL MODELS IN SIMULATION

We can classify the use of computational models for solving problems in one of three ways (Table 5-1). *Descriptive models* are behavioral models of an existing system. The purpose of descriptive models is to describe the system in a way that makes us better understand the complexity of the system. Descriptive models are useful to analyze complex dynamic environments. *Predictive models* are models that predict the way an existing system behaves in the future. The purpose of predictive models is to be able to know beforehand how the system will behave in the future. Such models are useful in tasks in which we need to make decisions based on future data from a complex dynamic environment. *Prescriptive models* are models of future—not yet existing—systems. The purpose of prescriptive models is to prescribe what a future system should look like. Such models are useful to design complex dynamic environments.

Table 5-1. Use of Computational Models

Type of computational model	Use in problem domains
Descriptive model	Describe an existing system in order to understand it.
Predictive model	Predict the future of an existing system.
Prescriptive model	Prescribe a future system that does not exist yet.

Models help us understand systems. According to Klir there are four basic levels of knowledge about a system (Klir 1985). At each level we know some important things about the system we did not know at lower levels (Table 5-2). The lowest level, the source level, identifies what part of the real world system we want to model, and the means by which we are going to observe it. It identifies the variables to measure and how to observe them. The next level, the data level, is the database of observations in terms of measurements of the variables from the source system. At the third level, the generative level, we have a model that can generate the data from the previous level. This is the level of system knowledge most people refer to as a model the system. At the fourth and highest level, the structure level, we have a description of the total system by coupling all generative components from the lower level together into a generative system for simulation.

Table 5-2. Klir's levels of system knowledge

Level	Name	System Knowledge
3	Structure	Components (at lower levels) coupled together to form a generative system, i.e. a simulation
2	Generative	Means to generate data in a data system
1	Data	Data collected from source system
0	Source	What variables to measure and how to observe them

Zeigler, et al, (2000) define three basic ways to deal with system problems, based on Klir's levels of system knowledge; system analysis, system inference and system design. They allow us to move from one level of system knowledge to another. In *System analysis*, we try to understand the behavior of an existing or hypothetical system based on its known structure. *System Inference* is performed when we do not know the structure of the system before hand. In system inference we try to guess the structure from observations, allowing us to use this to predict future data. Finally, in *system design* we are investigating the alternative structures for a completely new system or the redesign of an existing system.

The important notion in Klir's levels of system knowledge is that in system analysis we are not generating new knowledge, as we move from a higher-level to a lower-level description of the system. In system analysis we are only making explicit what is implicit in the higher-level description. Klir does not consider this kind of subjective (modeler-dependent) understanding to be an increase in knowledge about the system. I disagree, and argue that making something explicit that was implicit before will lead to more insight and a deeper understanding, which is a form of new knowledge. Even though in Klir's sense system analysis might not generate new knowledge, interesting properties of the system will come to light of which we were not aware before the analysis.

In both system inference and system design we move from lower levels to higher levels of system knowledge. Therefore, in these activities we are creating new knowledge that did not exist before, according to Klir's definition.

In Table 5-3, I relate Zeigler's fundamental system problems first to the transitions in terms of Klir's levels, and secondly to the types of computational models that we are developing at the generative level. When we are in a system analysis activity we are developing a descriptive model of the system. The development of a descriptive computational model leads to an increased understanding of how the system works. In system inference we are trying to create a predictive model. Predictive in the sense that once we have created a computational model that can explain the generation of observed data, we can now use this model to predict future behavior data of the system not yet observed. In system design we are developing a prescriptive model, in the sense that the model prescribes a future system.

Table 5-3. System problems related to model use and types

System problems	Model use	Transition between Klir's levels	Type of computational model
System Analysis	The system exists, and we try to understand its behavior.	Moving from a higher to a lower level of description, e.g. using information at the generative level to <i>generate</i> the source data at the data level.	<i>Descriptive model</i>
System Inference	The system exists, and we try to infer how it works from observations of its behavior.	Moving from a lower to a higher level, e.g. having data and trying to find a means to generate it. Then using the generative system to infer future behavior.	<i>Predictive model</i>
System Design	The system does not yet exist in the form we're contemplating, and we try to come up with a good design for it.	Moving from a lower to a higher level, e.g. having a means to generate data based a design at the generative level.	<i>Prescriptive model</i>

5.3 THE CASE STUDIES

To show that Brahms is a modeling and simulation environment for work practice, it needs to be shown that we can use Brahms to develop valid models for the three model uses described in the previous section.

To relate the types of uses of computational models (from Table 5-3) back to the research process shown in Figure 5-1, I instantiate three cycles in the research process used in this thesis. At the end of each cycle I will have an increased understanding of the research problem, and will be able to draw on the knowledge gained in that cycle to start the next. The sequence of the three cycles follows the sequence of system problem description in Table 5-3. The reason for this is that the complexity of system problem increases in that order, and thus the learning from a case study performed in a cycle can be used in the next.

The domain for the three case studies is a non-traditional work system of *scientific fieldwork on the Moon*. The reasons for choosing this domain are the following:

1. Working for NASA, while finishing this thesis, means that the research performed should be beneficial to NASA in some form. Understanding how astronauts worked on the Moon, and how people and robots could collaborate in future missions to planets, is an important research topic.
2. Scientific fieldwork on the Moon is *work*, even though we might not view astronauts “hopping” around on the Moon, or robots driving on the Moon as working. However, if we consider what is being accomplished on such missions, and the amount of collaboration that takes place, both on the Moon and on Earth, it is less of a stretch to view NASA missions as complex work systems.
3. The nature of the technology that was developed during the Apollo era, and the technology that is being developed for future NASA missions, is so important in the performance of the work during these missions that it is a perfect domain for studying how technology needs to be integrated with practice. This becomes especially important when, in the future, humans will need to collaborate with robots to build factories on the Moon or on Mars.

Therefore, the three case studies I present in the next three chapters are not about a more traditional workplace on Earth, but about people and robots working on the Moon. Table 5-4 gives a short description of the three case studies and their objectives. First, I will show the use of Brahms to *describe* a work practice from the Apollo days. Next, I will show the use of Brahms to *predict* future behavior of astronauts, based on a model of one of the Apollo missions. Last but not least, I will show the use of Brahms to *design* a work system for a future robotic mission to the Moon.

Table 5-4. Case study descriptions

Case Study	Objective	Type of Brahms Use
1. Apollo 12 ALSEP Offload	Model and simulate the ALSEP Offload activity during the Apollo 12 mission.	Descriptive modeling
2. Apollo Heat Flow Experiment Deployment	Model and simulate the Heat Flow Experiment deployment, based on Apollo 15 and 16, and predict activity and communication behavior in error situations.	Predictive modeling
3. Victoria Robotic Mission	Model and simulate missions operations of the proposed Victoria mission, performing scientific exploration with a semi-autonomous rover.	Prescriptive modeling

6. CASE STUDY 1: APOLLO 12 ALSEP-OFFLOAD

In this chapter, I report on the results of the *descriptive modeling* case study (Sierhuis 2000) (Sierhuis et al. 2000a) (Sierhuis et al. 2000c) (Sierhuis et al. 2000d). A descriptive model is an abstraction of an existing or historical system, and preserving the relations between system states—system morphism. I describe the development of a Brahms model and simulation of the ALSEP Offload activity that was part of the ALSEP instrument deployment during the Apollo 12 Lunar mission.

This chapter is divided into a number of sections that could be read or skipped independent of the other sections. Section 6.1 gives a short introductory description of the ALSEP Offload task that the astronauts performed during the Apollo 12 mission. Sections 6.2, 6.3, and 6.4 should be read together. They describe the design of the agent-, object-, and geographical models of the Brahms model. Section 6.5 describes the design of the activity model. Section 6.6 describes the behavioral model. This section describes how the workframes of an agent are executed, and explains the relationship between time, beliefs, workframes, activities and detectables during the simulation of the model. Section 6.7 describes how the communication between the lunar surface astronauts, as well as the lunar surface astronauts and the Capcom agent in mission control is simulated. Included in the communication model is the simulation of the communication time delay between the Moon and the Earth. Section 6.8 describes in more detail how we can model the interaction between people and artifacts. I describe this by explaining how we can model the interaction with a photo camera, while performing the activity of taking a photograph. Section 6.9 describes the process of verifying and validating the Apollo 12 ALSEP Offload model. In this section I describe how the Brahms model and simulation is verified and validated against the available historical Apollo 12 data. Last, section 6.10 describes my conclusions for this case study.

Goals and Objectives

The goal of this experiment was to investigate the use of the Brahms-language in order to *describe an existing work practice*. The challenge I faced in this experiment was to investigate if the theory of modeling work practice, implemented in the Brahms language (Chapter 4), is sufficient to describe the work practice in the chosen domain. The objectives of this first experiment were:

1. Being able to represent the people, things, and places relevant to the domain.
2. Represent the actual behavior of the people, second by second, over time.
3. Show which of the tools and artifacts are used when, and by whom to perform activities.
4. Include the communication between co-located and distributed people, as well as the communication tools used, and the effects of these communication tools on the practice.

The domain I chose for this experiment is the work practice of the Apollo 12 astronauts in the deployment of the Apollo Lunar Surface Experiments Package (ALSEP) on the Moon. The reasons for choosing this domain are the following:

1. The work performed by the astronauts requires unique and highly skilled individuals. The complexity of the work to be described is high enough to argue that if we can model this type of work practice within Brahms, we can model most other work practices as well.
2. The ALSEP deployment process is performed by a relatively small number of people. This has a positive impact on the modeling and simulation effort, in terms of the time it takes to develop the model, as well as the time it takes to simulate the model.
3. The ALSEP deployment work is distributed over the people involved, and is collaborative in nature.

4. There is no work product “flowing” through the work process. This means that this type of work is not easily represented in a workflow model. Being able to model this type of work in Brahms supports the argument for developing Brahms.
5. In order to develop a descriptive model of an existing work practice, we need to have access to a significant amount of data about the actual work. This often means a long observational and/or ethnographical study of the participants. This takes an enormous amount of effort and is a grounded research process in and of itself. However, the Apollo project has been well documented by NASA and numerous institutions, and writers (Compton 1989) (Wilhelms 1993) (Chaikin 1994) (Godwin 1999). Specifically, there is a significant library of video and audiotapes taken during the actual missions (NASA 1972)]. This allows us to develop, verify and validate our models using independent data from the real events.
6. Although a Human Mission to Mars is not an official NASA supported activity at this point in time, more and more researchers in and outside of NASA are informally studying what it would take to have humans go to Mars and do scientific work for an extended period of time. We know very little about how people can or should work on Mars. The only reference point we have about humans working on extra-terrestrial planets is the work that humans did on the Moon during the Apollo project. Developing models of the work practices on the Moon might allow us to extrapolate these models and investigate people working on Mars, before we can physically go there.
7. There is, to certain extent, a real possibility that before we will go to Mars we will first go back to the Moon. The reasons to do this might be of a scientific or a commercial nature. Regardless of the reason to go back to the Moon, a model that describes the existing, but mostly forgotten work practices for deploying instruments on the Moon is self-evident.

6.1 APOLLO 12 AND THE ALSEP OFFLOAD

One of the biggest objectives of the Apollo 12 mission was to deploy the Apollo Lunar Surface Experiments Package (ALSEP). It would be the first time to deploy the ALSEP on the moon. The earlier Apollo 11 mission only deployed a preliminary version, called the EASEP (Early Apollo Surface Experiment Package). The ALSEP consisted of a number of independent scientific instruments that were to be deployed on the moon. The instruments were data collection devices for different scientific experiments about the moon's internal and external environment. By deploying similar ALSEP instruments over multiple Apollo missions (A12, 14, 15, 16 and 17), the ALSEP deployments created an array of data gathering instruments at different locations on the lunar surface. Table 6-1 shows a list of deployed instruments by mission.

Table 6-1. ALSEP experiments for Apollo missions

	A 12	A 13	A 14	A 15	A 16	A 17
Passive Seismic Experiment (PSE)	X	(X) ³²	X	X	X	
Active Seismic Experiment (ASE)			X		X	
Suprathermal Ion Detector Experiment (SIDE)	X		X			
Solar-Wind Spectrometer	X			X		
Lunar Surface Magnetometer (LSM)	X			X		X
Cold Cathode Gage (CCG)	X	(X)	X	X		
Charged Particle Lunar Environment Experiment (CPLEE)		(X)	X			
Solar Wind Spectrometer (SWS)					X	
Heat Flow Experiment (HFE)			(X)	X	(X)	X
Lunar Ejecta and Meteorites Experiment (LEAM)						X
Lunar Seismic Profiling Experiment (LSPE)						X
Gravimeter, Lunar Surface (LSG)						X
Lunar Atmosphere Composition Experiment (LACE)						X

To deploy the ALSEP on the lunar surface, the astronauts had to accomplish three high-level tasks. First, they had to *offload* the ALSEP from the Lunar Module (LM). Second, they had to *traverse* with the ALSEP packages to the deployment area, away from the LM. Third, they had to *deploy* each ALSEP instrument onto the surface. In this chapter, I discuss the development of a work practice model for the first task, the *ALSEP Offload*.

³² The round brackets mean that these were planned experiments that failed to be deployed properly.

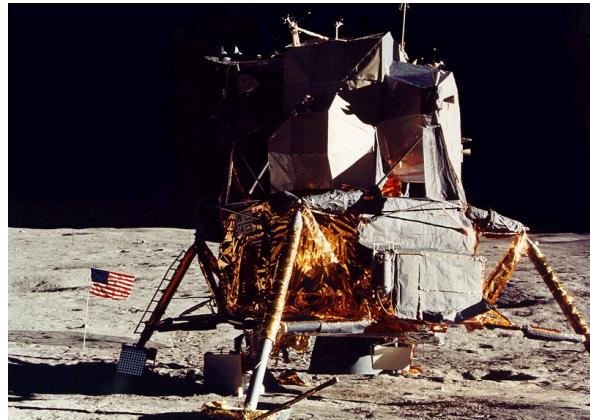


Figure 6-1. SEQ Bay and RTG Cask located on the side of the LM

All the ALSEP instruments and tools, used for deployment, were stored on two sub-pallets ("packages") in the Scientific Equipment Bay (SEQ Bay) during flight. Figure 6-1 shows the SEQ Bay located on the LM, on the opposite side of the ladder from which the astronauts descended to the lunar surface.

The offload consisted of a number of specified (sub-)activities that were trained extensively and assigned to each of the astronauts. The order in which these tasks were to be performed, and whether the Lunar Module Pilot or the Commander was to perform the task, i.e. the plan, was the same for all five missions. Figure 6-2 shows the plan and start-time for the Apollo 12 ALSEP Offload.

CDR	LMP
<u>1+15 ALSEP OFFLOAD</u>	<u>1-15 ALSEP OFFLOAD</u>
	OPEN SEQ BAY DOOR
<div style="border: 1px solid black; padding: 5px; width: fit-content; margin-bottom: 5px;"> <u>REMOVE PKG 1</u> <u>REMOVE PKG 2</u> <u>STOW BOOMS</u> <u>DEPLOY HTC</u> <u>UNSTOW UHT (2)</u> <u>UNSTOW CASK TOOLS</u> <u>CONNECT BAR</u> <u>TIP PKG 2</u> <u>REMOVE SIDE</u> <u>LOWER CASK</u> <u>FUEL RTG</u> <u>CLOSE DOORS</u> <u>CONNECT PKGS</u> </div> <u>(BACK WALL OF SEQ BAY)</u>	
<u>REST/CHECK EMU</u> <u>SUR-47</u>	<u>REST/CHECK EMU</u> <u>SUR-47</u>
<u>Basic Date</u> <u>October 27, 1969</u> <u>Changed</u> _____	

Figure 6-2. Apollo 12 Surface Checklist 47 for the ALSEP Offload

The order in which the astronauts were to perform their tasks was pre-specified and trained. In other words, offloading the ALSEP was a highly choreographed collaborative activity performed by two astronauts working in parallel.

However, even though this high-level task was planned and choreographed up front, the plan did not include the situational variations, the actual communication and collaborative activities between the astronauts, and the communication between and coordination of activities by the Manned Spaceflight Center (MSC) in Houston. MSC, also known as Mission Control, kept track of where the astronauts were on the plan, solving unplanned problems, and monitoring and communicating life support status for the astronauts. Central in this collaborative activity is the person who played the role of Capsule Communicator (CapCom). The CapCom was the "voice" of Houston and the only person in direct communication with the astronauts. This communication happened through the voice-loop (see section 6.7 on modeling the voice-loop).

The work practice of the ALSEP Offload, or any work practice for that matter, consists of more than the sequence and distribution of tasks. What constitutes the practice of the ALSEP Offload is the way the actual plan is carried out; The situational activities of the collaborators, the way they react to their environment, the way they communicate, what is said, the way they “know” how to do their tasks given the situation. It is *situated action* (Suchman 1987). A choreographed play “executed” during the performance, planned and trained, but always different.

In the next sections, I will describe how the ALSEP Offload work is modeled in a model of work practice. The model is not a model of the problem-solving knowledge of each individual involved in this task. Instead, it is a model of the behavior of the individuals. It describes how the collaboration, coordination, and communication between the three individuals happen, and make this a fluent event. The activities of one individual are like the movements of a musician in a symphony orchestra. The communication between individuals is like the interleaved notes that seem to “tell” each musician what to play next. The artifacts and tools are like the instruments of the musician. The environment of the Moon and Mission Control is like the symphony hall. The Brahms “symphony” that is being played is planned and scored on a piece of paper (i.e. the astronaut’s checklist). The orchestra has trained the piece many times (i.e. the astronaut training on Earth). However, what comes out in the performance is due to their practice, the concert hall (i.e. the Moon), and the way they play together that specific evening (i.e. EVA 1 on Apollo 12).

6.2 THE AGENT MODEL

One of the most relevant design issues for any Brahms model is the design of the agents and the groups they belong to. The *Agent Model* describes to which groups the agents belong and how these groups are related to each other.

Designing an Agent Model is similar to the design of an Object Model in object-oriented design (Rumbaugh et al. 1998). Just as the class-hierarchy in an Object Model, we need to design the group-hierarchy in the Agent Model. As a rule of thumb, we identify the communities of practice of which the agents in the model are members, and abstract them to a common denominator for all agents. All agents are members of this abstract group. The most abstract group is called the *Base Group*. This group exists in the Brahms Base library. It contains all attribute and relation definitions that are needed by default, such as the name of the agent, the group membership relation, and the location of the agent. We specialize this group, until we have identified all the similarities and differences between the agents. It should be noted, again, that groups and agents can be members of multiple groups.

Figure 6-3 shows the Agent Model design. We start with defining our agents. Each agent represents a person in our domain, e.g. Ed Gibson, Pete Conrad, Al Bean, and Dick Gordon. We generalize the community all four agents belong to as the group of *ApolloAstronauts*.

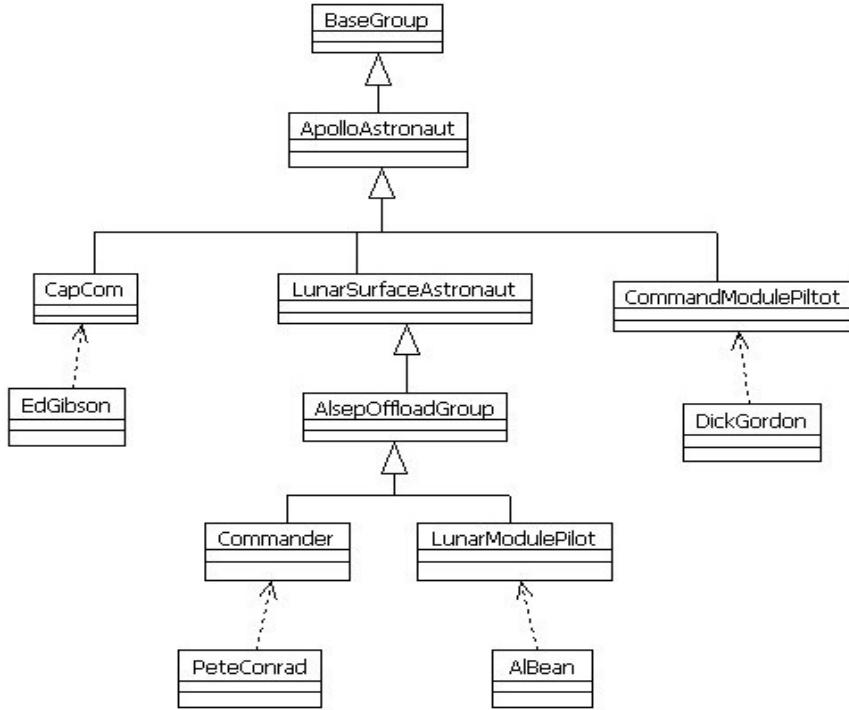


Figure 6-3. Apollo Agent Model design

The *Capsule Communicator* (CapCom) was always an astronaut. In the case of Apollo 12, Ed Gibson was a civilian chosen with the fourth group of astronauts in 1965. The role of the CapCom was to be the only person in Mission Control to talk directly to the astronauts. Dick Gordon, the *CommandModulePilot* (CMP), was a Navy Captain chosen with the third group of astronauts in 1963. The role of the CMP was to fly the Command Module (CM), named “Yankee Clipper”, circling in orbit around the moon while the Lunar Module (LM), named “Intrepid”, was on the moon. Pete Conrad, the Apollo 12 *Commander* (CDR), also a Navy Captain, was chosen with the second group of astronauts in 1962. Last, the *LunarModulePilot* (LMP) Al Bean, who was also in the Navy, was chosen with the third group of astronauts in 1963.

I represent the *role* of each of the astronauts as a group. This way I can represent role specific attributes and activities at the group level. The *AlsepOffloadGroup* is a *functional* group in the sense that it does not specify a specific role, but a task of the agent. This group represents all work activities and attributes that have to do with the ALSEP Offload task in one group. This way the group represents the community of agents that can perform the ALSEP Offload task. For Apollo 12, both the CDR and the LMP trained for the ALSEP Offload activities, and both of them could, if necessary, perform the ALSEP Offload task by themselves, and therefore belong to the group ALSEPOffloadGroup. Thus, the Commander and LunarModulePilot groups are members of the group AlsepOffloadGroup. Since both the CDR and the LMP were working on the surface there are tasks that both astronauts needed and/or could perform. The ALSEP Offload task was one of them, but there were others as well. All the activities that needed to be performed by all astronauts on the lunar surface are represented in the *LunarSurfaceAstronaut* group. Such activities include taking photographs and changing the cooling of their space suit. In conclusion, we can describe the group hierarchy of Apollo astronauts in three sub-groups, CapCom, CommandModulePilot, and LunarSurfaceAstronaut. The LunarSurfaceAstronaut group has the AlsepOffload group as a subgroup, which in turn is subdivided into the subgroups Commander and LunarModulePilot.

Figure 6-4 shows the Brahms source code of the group and agent definitions, as shown in Figure 6-3.

```
// Groups
group BaseGroup { ... }

group ApolloAstronaut memberof BaseGroup { ... }

group CapCom memberof ApolloAstronaut { ... }

group LunarSurfaceAstronaut memberof ApolloAstronaut { ... }

group CommandModulePilot memberof ApolloAstronaut { ... }

group AlsepOffloadGroup memberof LunarSurfaceAstronaut { ... }

group Commander memberof AlsepOffloadGroup { ... }

group LunarModulePilot memberof AlsepOffloadGroup { ... }

// Agents
agent PeteConrad memberof Commander { ... }

agent AlBean memberof LunarModulePilot { ... }

agent DickGordon memberof CommandModulePilot { ... }

agent EdGibson memberof CapCom { ... }
```

Figure 6-4. Brahms source code of the agent model

Figure 6-5 shows the Agent Model in the Brahms Model Builder. The Brahms Model Builder application allows the modeler to create and compile the Brahms model. Figure 6-5 shows the group and agent hierarchy compiled from the source code in Figure 6-4. Each group has a number of “folders” underneath it. Each folder is a different category of model elements for the group. The “Member Groups” folder contains all the subgroups that are a member of the group. The “Member Agents” folder contains all agents that are members of the group. The rectangles around the groups and agents are not part of the GUI, but are added for clarification purposes, so that the reader can easily identify them in the figure. The colors show the different group-levels. At the top, in black, you see the BaseGroup. All the other groups are a subgroup of the BaseGroup group, and are therefore shown in the “Member Group” folder. The yellow rectangles show the four agents, while the other colors show the intermediate groups in the group hierarchy (the colors are only visible in a color reprint).

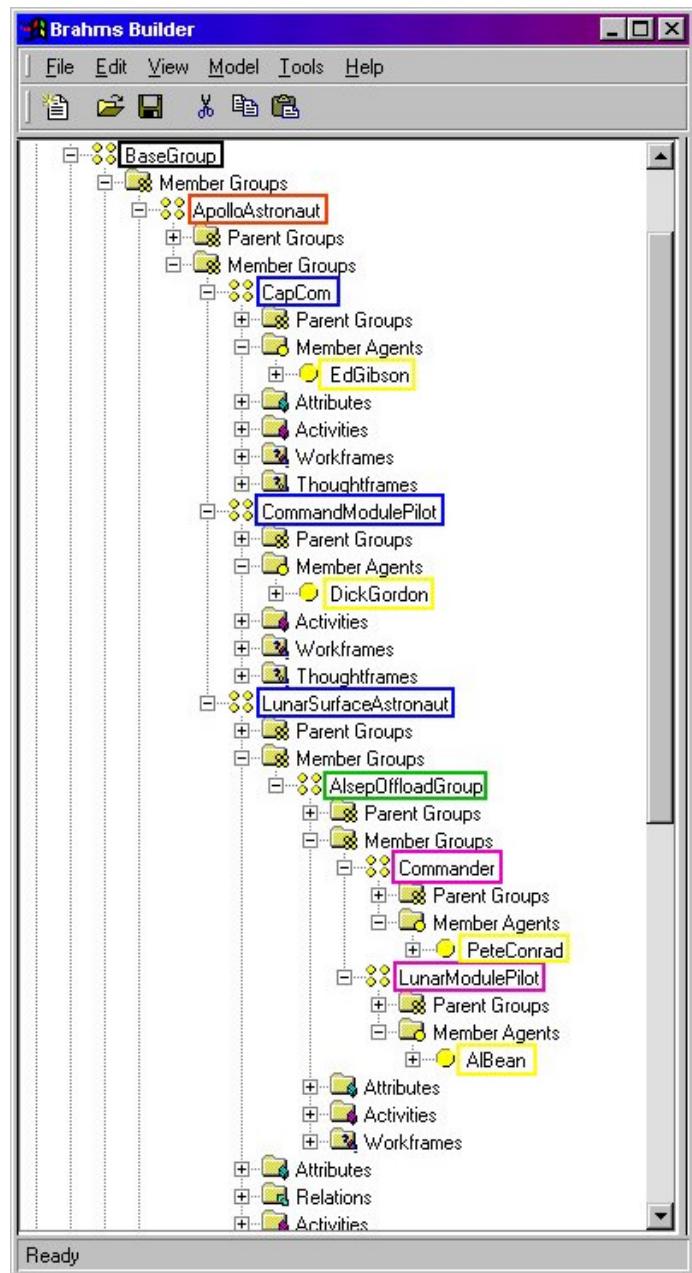


Figure 6-5. Agent Model in the Brahms Model Builder

6.3 THE OBJECT MODEL

After the Agent Model, the next model that needs to be designed is the Object Model. In this model we design the class-hierarchy of all the domain objects. Figure 6-6 shows the Object Model design in UML (Rumbaugh et al. 1998) for the Apollo 12 domain objects and artifacts. As with the Agent Model, the root-class of the class hierarchy is the class *BaseClass*. All other classes and objects inherit from this *BaseClass* class.

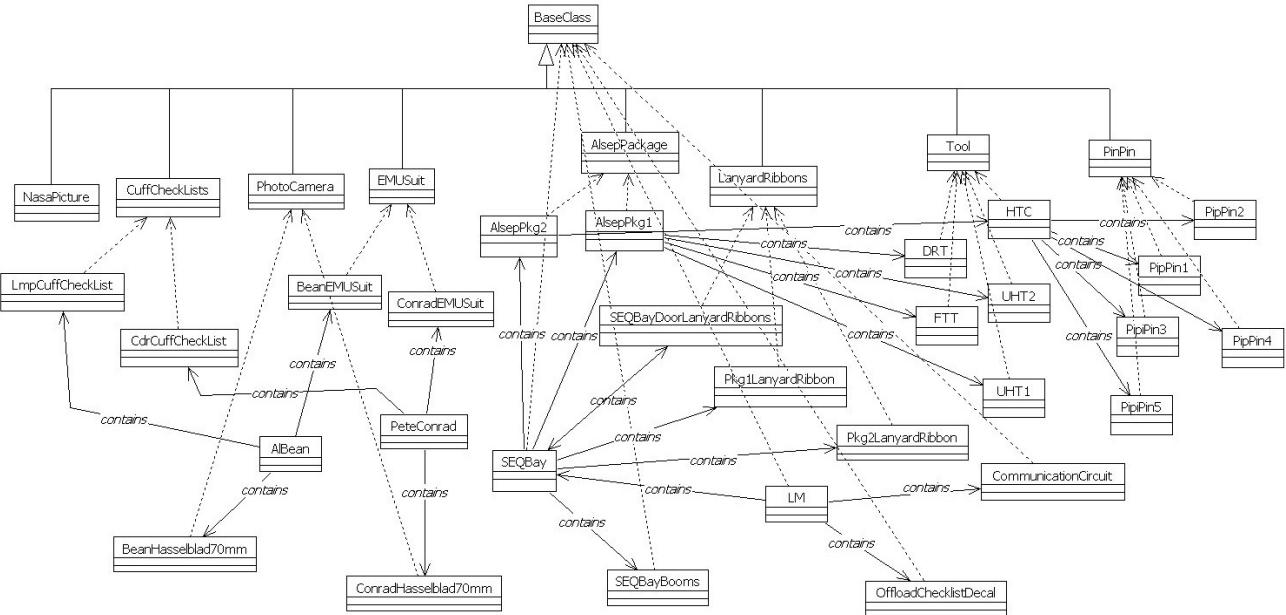


Figure 6-6. Apollo Object Model design

The objects with the dotted arrows pointing up represent the *object instances* of a class. The solid arrows show the *built-in contains relation*. This relation represents objects contained in other objects. This relation has a pre-defined semantic meaning, which is discussed in section 6.4.4.

The objects with the dotted arrows pointing up represent the *object instances* of a class. The solid arrows show the *built-in contains relation*. This relation represents objects contained in other objects. This relation has a pre-defined semantic meaning, which is discussed in section 6.4.4.

Figure 6-7 shows the Brahms model source code for the *LM* and *SEQBay* objects. Both the *LM* and *SEQBay* objects are instances of the class *BaseClass*. Besides representing the corresponding artifacts on the Apollo 12 mission, the source code also specifies the *initial location* of these objects within the Geography Model (see section 6.4). Both objects are located in the *SEQBayArea* area. Furthermore, the objects declare the *attributes* with which we can describe the different aspects of these objects. Although we could describe any number of aspects of an object, such as the color, height, et cetera, we only declare those attributes that are relevant. To model the fact that the astronauts inspect the *LM* and the *SEQ Bay's* exterior appearance after the landing, we declare the attribute *exteriorAppearance* as a type *symbol* attribute. Using this attribute we can represent the state of the exterior of these objects. Both the *LM* and the *SEQBay* objects have an *initial fact* describing the state of their exterior appearance after the landing on the moon as an initial fact for the simulation, e.g.

(the exteriorAppearance of current = SEQBayExteriorLooksGood).

The status of the door of the SEQBay is modeled with the *door* attribute of type symbol that can have a value of *closed* or *open*. The door is in the initial state (i.e. an initial fact) of being closed, e.g. (*the door of current = closed*). This represents the door of the SEQ Bay being closed at the start of the ALSEP offload. Next, we model the objects that are located within the LM and SEQ Bay. This is represented with the *contains* relation (see Figure 6-6). This relation is declared in the BaseClass class, and inherited by the LM

and SEQBay objects. The fact that the SEQBay is located on the outside of the LM is represented as an initial fact in the LM object, i.e.

(current contains SEQBay).

The object LM represents the Apollo 12 Lunar Module, named Intrepid. This is the Lunar Module in which the astronauts landed in Surveyor crater. For this model the only relevant object that is part of the LM and that is relevant for the ALSEP Offload is the SEQBay, positioned on the outside of the LM. The SEQBay contains a number of artifacts that are relevant during the ALSEP offload activity. These artifacts are also modeled as Brahms objects in the model (see Figure 6-7 and Figure 6-8).

```
// Apollo 12 objects
object LM instanceof BaseClass {
    display: "Intrepid";
    location: SEQBayArea;
    attributes:
        public symbol exteriorAppearance;
    initial_facts:
        (the exteriorAppearance of current = LmExteriorLooksGood);
        (current contains SEQBay);
}
object SEQBay instanceof BaseClass {
    location: SEQBayArea;
    attributes:
        public symbol door;
        public symbol exteriorAppearance;
    initial_facts:
        (the exteriorAppearance of current = SEQBayExteriorLooksGood);
        (the door of current = closed);
        (current contains AlsepPkg1);
        (current contains AlsepPkg2);
        (current contains OffloadChecklistDecal);
        (current contains SEQBayDoorLanyardRibbons);
        (current contains Pkg1LanyardRibbons);
        (current contains Pkg2LanyardRibbons);
        (current contains SEQBayBooms);
}
```

Figure 6-7. Apollo 12 LM and SEQ Bay Brahms objects

First, there are the LanyardRibbon objects. These objects are used to open the SEQBay door (SEQBayDoorLanyardRibbons) and lower the ALSEP packages (Pkg1LanyardRibbons and Pkg2LanyardRibbons), respectively. The lanyard ribbons are rope-like artifacts the astronauts pull on to open the door and lower the packages. The two main objects are the ALSEP packages, AlsepPkg1 and AlsepPkg2. These are the packages the astronauts have to lower from the SEQ Bay and position on to the lunar surface. The SEQ Bay also contains booms (SEQBayBooms). These artifacts are rail extension structures at the top of the SEQ Bay. When the astronaut pulls on the package lanyard ribbons, the ALSEP package comes out attached to the booms. The packages are automatically released from the booms, after which the astronaut slowly lowers them to the lunar surface by releasing the tension on the lanyard ribbons. The last artifact of interest in the SEQ Bay is the OffloadChecklistDecal object. This is the decal that is shown in Figure 6-2, and is a decal that shows the activities and their order for offloading the ALSEP. It is there as a reminder for the astronauts.

Figure 6-8 shows the objects mentioned above, as well as the objects that are contained in each of them. One last interesting note to make is that of the pippin objects. Pippins were used to fasten objects to the ALSEP packages and other artifacts. The HTC (Hand Tool Carrier) object is fastened on AlsepPkg2 with five pippins. The fact that the pippins fasten the HTC is modeled by having them be contained in both the AlsepPkg2 object and in the HTC object. Removing the HTC from AlsepPkg2 means to first “remove” the pippin objects from both the HTC and the AlsepPkg2 objects, before the HTC object can be removed from the AlsepPkg2 object.

```

// Apollo 12 objects
object SEQBayDoorLanyardRibbons instanceof LanyardRibbons {}

object Pkg1LanyardRibbons instanceof LanyardRibbons {}

object Pkg2LanyardRibbons instanceof LanyardRibbons {}

object AlsepPkg1 instanceof AlsepPackage {
    initial_facts:
        //carries objects
        (current contains DRT);
        (current contains FTT);
        (current contains UHT1);
        (current contains UHT2);
}

object AlsepPkg2 instanceof AlsepPackage {
    initial_facts:
        //carries objects
        (current contains PipPin1);
        (current contains PipPin2);
        (current contains PipPin3);
        (current contains PipPin4);
        (current contains PipPin5);
        (current contains HTC);
}

object HTC instanceof Tool {
    initial_facts:
        //carries objects
        (current contains PipPin1);
        (current contains PipPin2);
        (current contains PipPin3);
        (current contains PipPin4);
        (current contains PipPin5);
}

object PipPin1 instanceof PipPin {}
object PipPin2 instanceof PipPin {}
object PipPin3 instanceof PipPin {}
object PipPin4 instanceof PipPin {}
object PipPin5 instanceof PipPin {}

object DRT instanceof Tool {} //Dome Removal Tool
object FTT instanceof Tool {} //Fuel Transfer Tool
object UHT1 instanceof Tool {} //Universal Handling Tool
object UHT2 instanceof Tool {}

object OffloadChecklistDecal instanceof BaseClass {}

```

Figure 6-8. Apollo 12 contained artifacts

Now that the agents and artifacts are represented, the next section describes the geography model in which the agents and artifacts are located during the simulation.

6.4 THE GEOGRAPHY MODEL

In Brahms we model geographical locations using two concepts, *area-definitions* and *areas*. Area-definitions are user-defined types of areas. Areas are instances of area-definitions. Thus an area is an instance of a specific location in the real world that is being modeled. Furthermore, areas can be *part-of* other areas. With this representation scheme we can represent any location at any level of detail.

For the Apollo 12 ALSEP Offload activity, the following locations are important; Earth, the Manned-Space Center (MSC), the Moon, the Apollo 12 landing-site ("Surveyor Crater"), the area where the SEQ Bay is located, the ALSEP deployment area, an area away from the SEQ Bay to place artifacts after offloading,

and last, the lunar orbit and the Command Module ("Yankee Clipper"). Figure 6-9 shows the geography model design.

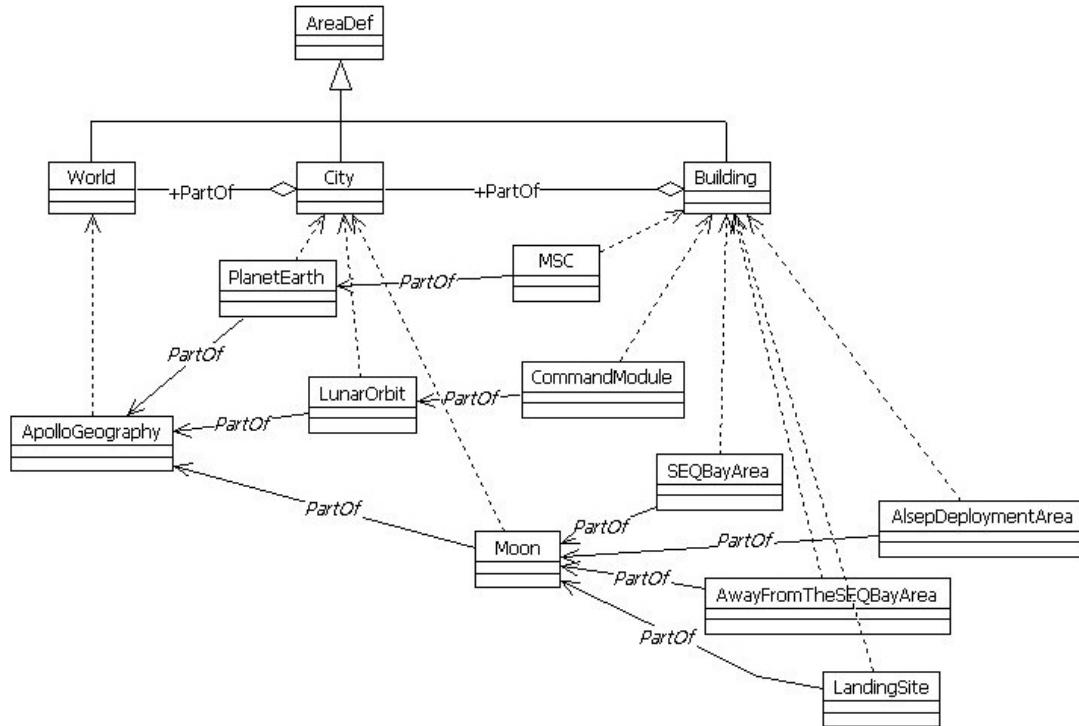


Figure 6-9. Apollo Geography Model design

Figure 6-10 shows the Brahms source code of the area definitions (areadef) and area objects (area). The *area definition* types used to represent the area-instances are World, City and Building.

```

areadef World {}
areadef City {}
areadef Building {}

area ApolloGeography instanceof World {}

// back on Earth!
area PlanetEarth instanceof City partof ApolloGeography {}
area MissionControlCenter instanceof Building partof PlanetEarth {}

// on the Moon!!
area Moon instanceof City partof ApolloGeography {}
area LunarOrbit instanceof City partof ApolloGeography {}
area SEQBayArea instanceof Building partof Moon {}
area AwayFromTheSEQBayArea instanceof Building partof Moon {}
area AlsepDeploymentArea instanceof Building partof Moon {}

// Apollo 12 Geography
area CommandModule instanceof Building partof LunarOrbit {
    display: "Yankee Clipper";
}
area LandingSite instanceof Building partof Moon {
    display: "Surveyor Crater";
}
  
```

Figure 6-10. Geography Model Brahms source code

It does not seem logical to give the area-definitions the names “World”, “City”, and “Building,” and indeed it is not. The reason for this is the limitation of the current Brahms simulation engine³³. The current engine only accepts three types of areas, namely World, City and Building. Also, in the current engine there can only be one world-area. This limitation stems from the fact that our initial designed use of Brahms was for work practice models for the type of work that is performed within buildings, such as the more traditional office-work. This creates an obvious limitation in our representational needs for this extra-terrestrial work domain. First, the work happens in two different worlds, namely on Earth and on the Moon. We therefore would like to create two world-areas in our model. However, because of the current limitation of the engine we need to create the Earth and the Moon as type city-areas, being part of one world. We therefore create one world-area called *ApolloGeography*. This area represents the total “world” for our simulation. An area of type World can contain only areas of type City, therefore the Earth and Moon are areas of type City. Now we have our two planets—Earth and Moon—represented as cities. Secondly, the work on the Moon does not happen within buildings. However, we can only represent areas within a city-area as a type Building area. Thus, the Moon, being of type City, can only have areas of type Building located within it. We therefore represent the locations in which the astronauts perform their work as building-areas. A third “city” is created namely the orbit of the Command Module around the Moon. Since we are not concerned about the location of the Command Module with respect to the Moon and the Earth, we represent the orbit as a city-area within our world. The reason for this is that the Command Module Pilot “lives” within this area. It is therefore easier to locate the Command Module Pilot within his “building” location.

The geographical areas are hierarchically represented as instances of Buildings, which are part of Cities, which in turn are part of the World. This leads to the Compiled Geography Model as represented in Figure 6-11³⁴.

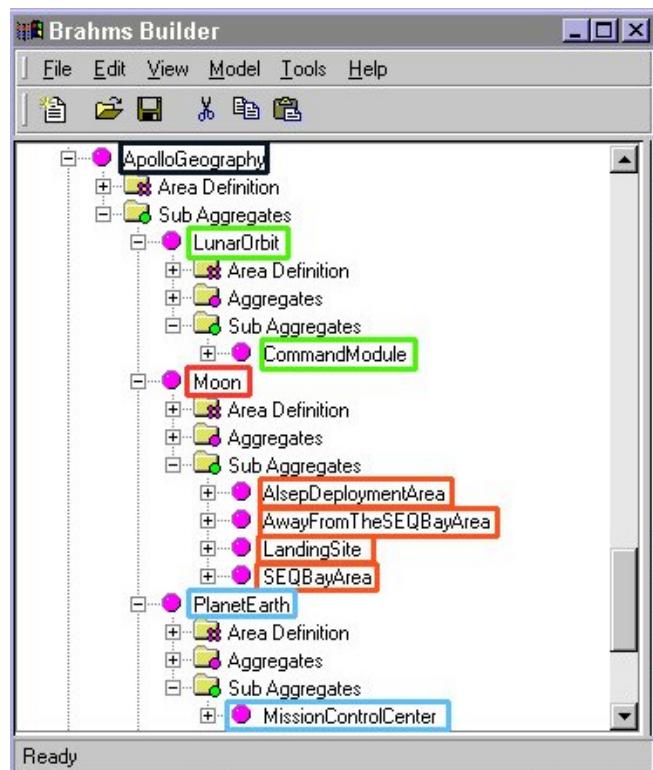


Figure 6-11. Apollo 12 ALSEP compiled Geography Model

³³ We have re-implementing the engine in Java.

³⁴ Figure 6-11 is a part screen capture from the Brahms Builder application.

6.4.1 Initial locations

Each agent and object has an initial location in one of the lowest-level areas, (CommandModule, AwayFromTheSEQBayArea, AlsepDeploymentArea, LandingSite, SEQBayArea, or MissionControlCenter). Initial locations are locations in which an agent or object is placed during the initialization phase of the simulation. This way each agent and object starts out being located in a geographical location (an area). To define an initial location for an agent the modeler uses the *location* attribute at the group or individual agent level. Figure 6-12 shows the initial location for each agent.

```
agent PeteConrad memberof Commander {
    location: LandingSite;
    ...
}
agent AlBean memberof LunarModulePilot {
    location: LandingSite;
    ...
}
agent DickGordon memberof CommandModulePilot {
    location: CommandModule;
    ...
}
agent EdGibson memberof CapCom {
    location: MissionControlCenter;
    ...
}
```

Figure 6-12. Agent initial location

6.4.2 Movement

Agents and objects can move from one area to another. Moving from one location to another removes the agent from the starting location and moves the agent to the new location. This is accomplished by having the agent perform a move-type activity. The time the activity is active (i.e. the activity duration-time) determines how long it takes the agent to move from location A to location B. Figure 6-13 shows an example of a move-activity.

```
move Moving(Building loc, int pri, int dur) {
    priority: pri;
    max_duration: dur;
    resources: MoveActivity;
    location: loc;
}
```

Figure 6-13. Move activity source code

The move-activity *Moving* starts in the area the agent is located at the moment the move-activity gets activated, and ends at the new area location given by the *loc* parameter. When both agents, PeteConrad and AlBean, perform the activity *Moving(SEQBayArea, 0, 5)* they both move independently from the LandingSite area (Surveyor Crater), their initial location, to the SEQBayArea in 5 seconds, as shown in Figure 6-14.

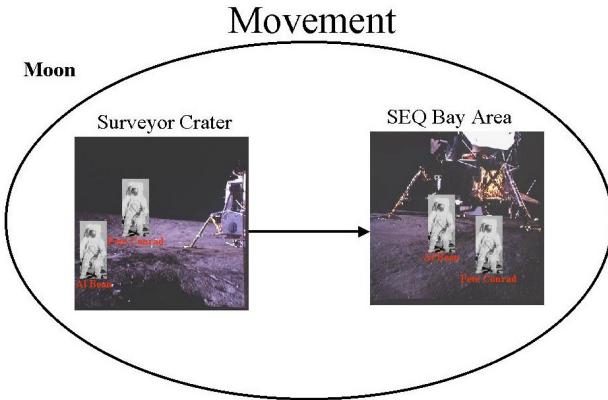


Figure 6-14. Pete Conrad and Al Bean moving to the SEQBayArea

Figure 6-15 gives a from-above view of the LM landing site and the ALSEP Offload Area of activity (the SEQBayArea in the model) from the Apollo 14 Press Kit (NASA 1971).

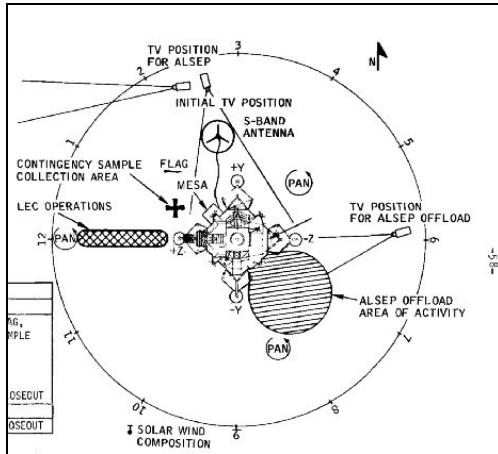


Figure 6-15. Apollo 14 Landing site and ALSEP Offload area of activity

6.4.3 Detecting agents and objects

As both agents arrive at their new location area they will immediately detect facts about the location of other agents and objects that are also in the area they arrive at. The simulation engine automatically creates beliefs for the agent from the facts about other objects and agents that are in the same location. The agents already in that location will get the belief that the agent that arrived is now also in the location. This way, agents will always notice other agents and objects that are in the location the same area.

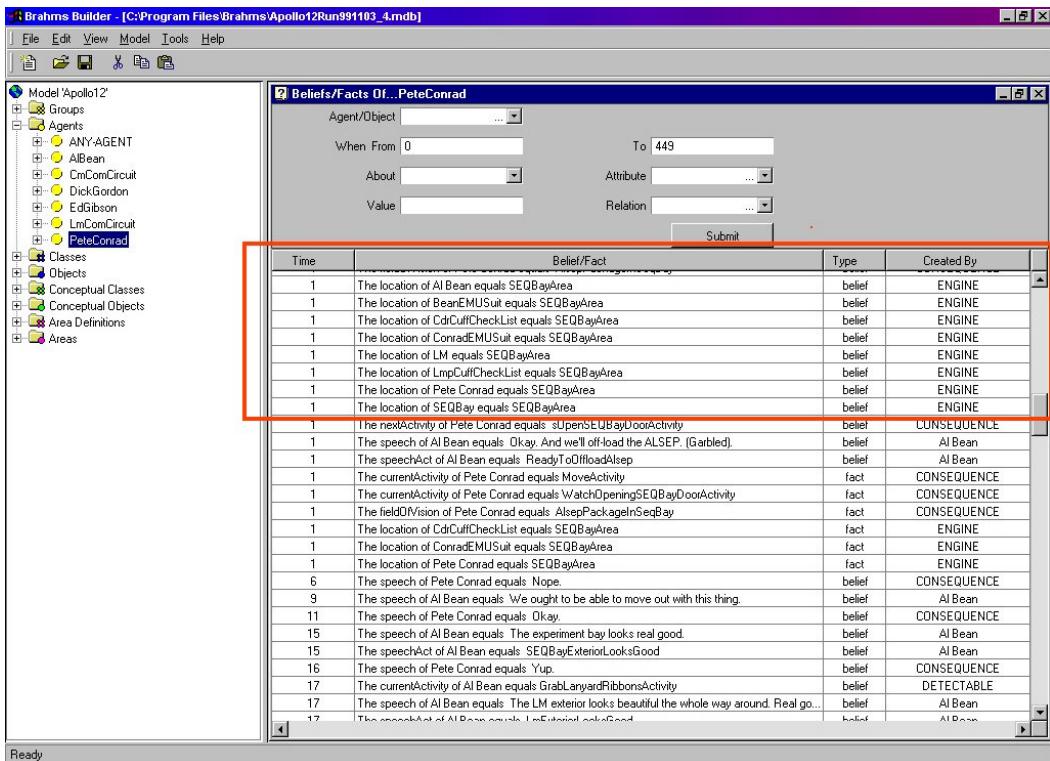


Figure 6-16. Pete Conrad's location beliefs

Figure 6-16 shows the beliefs and facts of the PeteConrad agent in the Brahms Builder application. By opening a simulation history database³⁵ the modeler can investigate what happened during a specific simulation run. Figure 6-16 shows all the beliefs the agent PeteConrad received and the facts it created during the specific simulation run. The columns show the time the agent created the belief or fact, the type (belief/fact) and how it was created (Created by). The (red) rectangle shows the *location* beliefs agent PeteConrad received at time one second into the simulation, created by the simulation engine (Created By: ENGINE). The agent received these beliefs due to the move activity *Moving* that moved the agent from the LandingSite area to the SEQBayArea. As you can see in Figure 6-16, at the moment the PeteConrad agent arrived at the SEQBayArea location it noticed (i.e. received the beliefs) that Al Bean, his EMU space suit and cuff checklist, the LM and the SEQBay, and he himself are all in the SEQBayArea location. Figure 6-16 also shows other beliefs and facts of the PeteConrad agent. The *Created By* column shows who or what created the belief/fact for the agent. *ENGINE* means that the simulation engine created the belief/fact, *CONSEQUENCE* shows that a consequence in a workframe or thoughtframe of the agent created the belief/fact. *DETETABLE* shows a detectable in a workframe created the belief. The name of an agent or object in the column shows that that agent or object communicated the belief to the agent.

6.4.4 Containment relation

During this case study I ran into a Brahms language limitation. To model the movement of agents and objects correctly I had to add the notion of *containment* to the language (see Figure 6-6). An agent or object can “carry” other agents and objects. Consequently, when an agent or object moves locations all the objects or agents that are “carried” by the moving agent or object should also move to the new location.

This is best explained with a simple example from the domain. As shown by the *contains-relation* in the object model in Figure 6-6, the lunar surface astronauts carry their EMU suit and their cuff checklist. As the astronauts move from location to location we want these carried objects to move with them, without having to specify this moving behavior separately for these objects. Instead, to accomplish this automatically we specified a built-in semantic relation called *contains*.

³⁵ a Microsoft® Access database

When the simulation engine executes a move-activity for an agent (or object) it first checks which objects or agents the moving agent contains. The simulation engine checks this by finding existing *facts* of the form:

Fact: (*[moving agent-or-object]* contains *[contained agent-or-object]*)

For every such fact the *contained agent-or-object* is moved as well. To simulate that an agent or object is no longer containing another object or agent the above containment-fact needs to be negated:

Fact: (*[moving agent-or-object]* contains *[contained agent-or-object]* is false)

Such a negation undoes the containment, and the previously contained agent or object will not be moved in case the agent or object moves.

Following is a small example of the use of the containment relation in the Apollo 12 model. Consider the following scenario; while the LMP agent is offloading an ALSEP package from the SEQ Bay, the CDR agent needs to move the first ALSEP package (AlsepPkg1) out of the way, so that the LMP can put the second ALSEP package down. Figure 6-17 shows the source code of the activity.

1. **conclude((current contains AlsepPkg1), bc:100, fc:100);**
2. **MovePkgOutOfTheWay(AlsepPkg1, AwayFromTheSEQBayArea, 100, 5);**
3. **conclude((current contains AlsepPkg1 is false), bc:100, fc:100);**
4. **Move(SEQBayArea, 10, 5);**

Figure 6-17. Moving contained object source code

In step 1, the agent “picks up” the object AlsepPkg1. This is modeled by creating a contains-relation fact (fc:100). A belief is also created for the agent (bc:100), because it is obvious that the agent knows he picked up the object. Next, in step 2, he performs a move-activity that moves both him and the contained objects to the AwayFromTheSEQBayArea area. Then, in step 3, the agent “sets down” the AlsepPkg1 object. This is modeled by negating the previously created containment fact and belief. Last, step 4 moves the agent, and its current contained objects, back to the SEQBayArea area. Consequently, the AlsepPkg1 object remains in the AwayFromTheSEQBayArea area.

Figure 6-18 shows the simulation output of the execution of the MovingPkg1OutOfTheWay workframe described above³⁶. The focus in the picture is on the area within the (red) rectangle. The picture shows the activity time-line of the CDR (agent PeteConrad) and that of the ALSEP package (AlsepPkg1) being moved. It can be seen that agent PeteConrad is performing step 2 and step 4 from Figure 6-17. The two rectangle boxes with the text “mv:³⁷” and “mv: Move” in it, show the duration of the two move activities. It can be seen that after step 2 (rectangle with text “mv:”) the location of both the agent PeteConrad and the object AlsepPkg1 has changed from the SEQBayArea area to the AwayFromTheSEQBayArea area³⁸. After agent PeteConrad has performed step 4 (rectangle with text “mv: Move”), only agent PeteConrad (and its contained objects not shown in Figure 6-18) has moved back to the SEQBayArea area. Consequently, due to step 1 and step 3 (the creation and negation of the containment fact), not shown in the figure but executed nonetheless, object AlsepPkg1 has been moved out of the way.

³⁶ Figure 6-18 is a screen dump of the AgentViewer tool that shows the result of a simulation. This interface is described in section 6.9.5, as well as the loose insert that is provided.

³⁷ Due to a lack of space in the rectangle, the name of the move activity “MovePkgOutOfTheWay” is not shown.

³⁸ Figure 6-18 only shows the “Away” text in the agent location bar, again, because of space limitation for the complete text string.

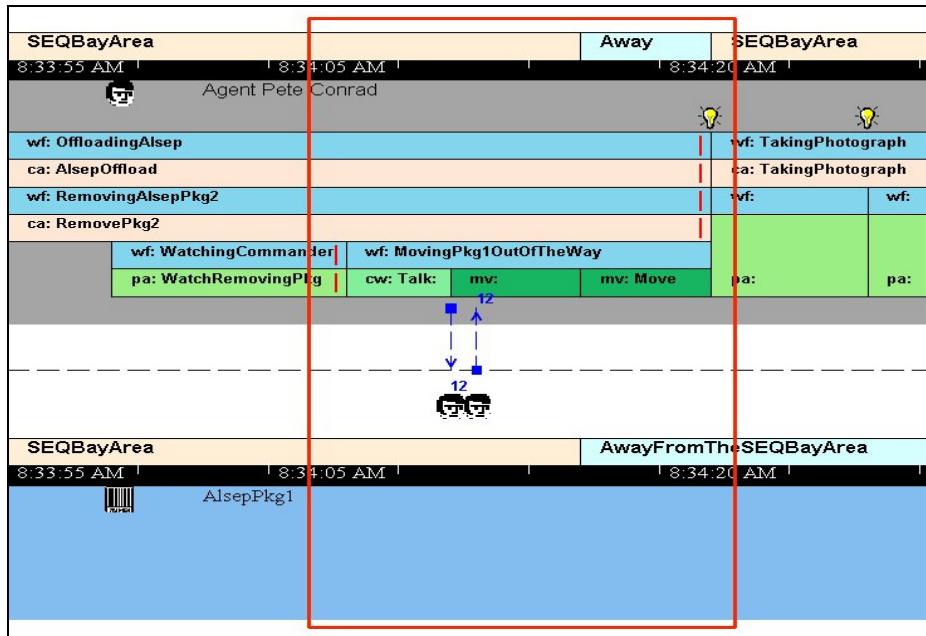


Figure 6-18. Moving contained object simulation

6.5 THE ACTIVITY MODEL

In this section, I describe the ALSEP Offload activities that are performed on the lunar surface, and I describe the Brahms model of the Apollo 12 ALSEP Offload. This model represents a part of the work practice of the Apollo 12 lunar surface astronauts as they performed the ALSEP Offload activity. As shown in section 6.2, there are four people relevant to the Apollo 12 ALSEP Offload; the lunar surface astronauts; Pete Conrad the Commander (CRD), Al Bean the Lunar Module Pilot (LMP), as well as Ed Gibson the Capsule Communicator (CapCom), and Dick Gordon the Command Module Pilot (CMP).

There are three separate areas where these four people are located during the Apollo ALSEP Offload activity (described in section 6.4). The CapCom sits in the Mission Control Center (MCC) located in the Manned Spaceflight Center in Houston, Texas³⁹. His main function is to listen to and communicate directly over the voice-loop with the astronauts. The CDR and LMP are the astronauts on the lunar surface and are located at or near the area of the SEQ Bay, which is located on the backside of the Lunar Module (LM) "Intrepid". The CMP is orbiting around the moon in the Command Module (CM) "Yankee Clipper." The main characters in the ALSEP Offload activity are CDR Pete Conrad, and the LMP Al Bean. Their work activities were planned and trained according to the ALSEP Offload checklist (see Figure 6-2).

Figure 6-19 shows that although the sequence of removing ALSEP packages during the mission was planned, there were more activities performed in practice. After the LMP identified that it is time to start the ALSEP Offload, he walks to the SEQ Bay and picks up the SEQ Bay door lanyard from outside of the SEQ Bay, and uses it to pull the SEQ Bay door open. The CDR is watching the LMP opening the door, and is not as suggested in the plan "doing-nothing". Once the SEQ Bay door is open, the CDR grabs the lanyard for lowering the first ALSEP package. He walks back from the SEQ Bay with the lanyard in his hand. Meanwhile, the LMP is warm and decides to lower the temperature in his EMU suit (Extra-vehicular Mobile Unit suit, i.e. his space suit). The CDR pulls the lanyard to move the first ALSEP package from the SEQ Bay and lowers it to the ground. While the CDR is performing this activity, the LMP is watching him. When the LMP sees a nice reflection in the CDR's helmet visor he decides to take a couple of photographs of the CDR. After the CDR has lowered the first ALSEP package to the surface, it is the LMP's turn to get the second ALSEP package out of the SEQ Bay (compare Figure 6-2 and Figure 6-19). The LMP performs the same activities as the CDR to lower the second ALSEP Package to the lunar surface. While lowering the second ALSEP package, it is the CDR who is watching the LMP. However, when the LMP is lowering the

³⁹ During the Apollo days the NASA center in Houston was called the Manned Spaceflight Center (MSC). Today it is referred to as Johnson Space Center (JSC).

package the CDR notices that the first ALSEP package is in the way, and mentions that he will take the first package and carry it away from the SEQ Bay area. Once he has done that, and is back at the SEQ Bay, he will take three photographs. One photograph of the first ALSEP package as he placed it away from the SEQ Bay, and two more photographs of the LMP lowering the second ALSEP package from the SEQ Bay. During these activities of the two astronauts on the lunar surface the CapCom is listening to the conversation of the astronauts.

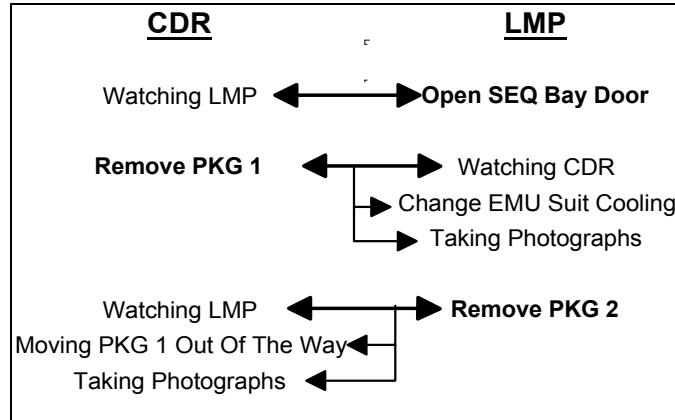


Figure 6-19. Activities in practice

There were activities that the astronauts performed that were not planned or trained. This has to do with the nature of what happens in practice, and is precisely what we want to capture in the model, since it defines how the work actually happened, i.e. the work in practice.

6.5.1 Data sources

I have identified the similarities and differences between the planned activities and the real performed activities during the mission by studying the transcribed data of the communications between the CDR, LMP and CapCom from the Apollo Lunar Surface Journal (Jones 1997). These communication transcriptions have been my major source of data for the Apollo 12 mission. Another source of information has been the Apollo 12 Press Kit (NASA 1969) and the Apollo 12 NASA Mission Reports (Godwin 1999). Unfortunately, there is no video data for the Apollo 12 ALSEP Offload available. This is due to the fact that an unforeseen problem with the TV camera lens and the bright sun on the Moon left the TV camera incapacitated from the beginning of the first EVA, right before the ALSEP Offload. Nevertheless, an accurate account of what happened during the ALSEP Offload can be derived from the second by second verbal communication between the astronauts, in combination with the mission plans. Also, there is video data available from the Apollo 14 ALSEP Offload activities. Although the specifics are somewhat different, the opening of the SEQ Bay door and lowering the ALSEP packages are similar, and the video is therefore a good source for filling in the gaps found in the transcribed communication data. Furthermore, the mission photographs are available as well, and provide some extra visual data.

6.5.2 The Apollo 12 ALSEP offload model

To reiterate, the goal of this modeling experiment is to *describe* the work activities of the lunar surface astronauts of the Apollo 12 mission as they are offloading the two ALSEP packages from the SEQ Bay. The hypothesis is that with Brahms we can describe (model and simulate) the work practice of these Apollo astronauts.

The data paints an accurate picture of the two lunar surface astronauts communicating. However, the data does not provide an accurate description of the activities of the LMP and CDR. Although the data provides some of the communication from the CapCom and the CMP, there is no detail data of the activities of the CapCom and the CMP. However, I will show that the model proofs the hypothesis, by accurately modeling and simulating the work practice of the Apollo 12 *lunar surface astronauts* during the ALSEP Offload, while including, where possible, some of the activities of the CapCom and the CMP.

The model describes the activities listed in Figure 6-2: Open SEQ Bay Door, Remove PKG-1, Remove PKG-2, Deploy Hand Tool Carrier, Unstow Cask Tools, Stow Booms, Unstow Universal Handling Tools, and Close SEQ Bay Door. The activity start- and end-times are computed from the Apollo Lunar Surface Journal (see Table 6-2) (Jones 1997).

Table 6-2. ALSEP Offload activity timetable

Activity	Performer	GET Begin Time	GET End Time	Total Time
1. Open SEQ Bay Door	LMP	116:31:34	116:32:22	0:00:48
2. Remove PKG-1	CDR	116:32:22	116:33:53	0:01:31
3. Remove PKG-2	LMP	116:33:53	116:34:44	0:00:51
4. Deploy Hand Tool Carrier	LMP	116:34:44	116:38:46	0:04:02
5. Unstow Cask Tools	LMP	116:34:44	116:36:25	0:01:41
6. Stow Booms	CDR	116:34:44	116:36:25	0:01:41
7. Unstow UHT	CDR	116:34:44	116:36:25	0:01:41
8. Close SEQ Bay Door	CDR	116:36:25	116:36:49	0:00:24

Figure 6-20 shows the activities Table 6-2 in the Brahms model. The model is viewed within a tree-view. Figure 6-20 shows the *AlsepOffload Group* in the Groups folder of the *Apollo 12 Model*. The parent groups of a group are positioned under the Parent Groups folder. The parent group of the AlsepOffload group is the *LunarSurfaceAstronaut* group (see also Figure 6-3), which means that the AlsepOffload group inherits all elements from that group. The subgroups of a group are positioned under the Member Groups folder. The subgroups are the *Commander* and *LunarModulePilot* groups, according to the design of the Agent Model (see Figure 6-3). The *PeteConrad* agent is a member of the *Commander* group, while the *AIBean* agent is a member of the *LunarModulePilot* group. Consequently, both the *PeteConrad* and *AIBean* agent inherit all the model elements defined in the AlsepOffload group, as well as all model elements inherited by the AlsepOffload group from its parent groups. This means that both agents can theoretically perform all the ALSEP offload activities. In reality this was also the case, since both astronauts trained the ALSEP offload activities together on Earth many times before the mission. If, for some reason, one astronaut would not be able to perform his planned activity, the other could perform it for him. This was shown in later missions, when some activities were performed by the astronaut who was not planned to perform the activity (e.g. during the ALSEP Offload on the Apollo 15 mission).

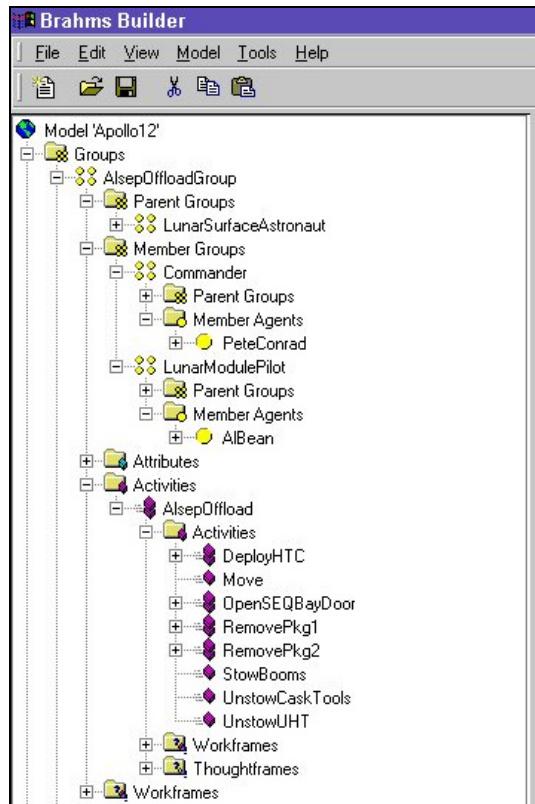


Figure 6-20. The Brahms ALSEP Offload group and activities model

The ALSEP offload activities from Table 6-2 are modeled as sub-activities of the *AlsepOffload composite-activity*, and can be seen in Figure 6-20 under the Activities Folder of the *AlsepOffloadGroup*. In the next sections, I will describe these activities in more detail, and will explain the Brahms model accordingly.

6.5.3 The open SEQ Bay door activity

The ALSEP Offload starts at 116 hours 31 minutes and 34 seconds ground-elapsed time (GET)⁴⁰, with the LMP announcing that they're starting the offload of the ALSEP (see Table 6-2 and Figure 6-21). The next activity is for the LMP to open the SEQ Bay door. In this section, I describe how I modeled this activity in Brahms, based on the available Lunar Surface Journal data (Jones 1997). Figure 6-21 is the transcription from the actual voice loop communication between the CDR and the LMP during the opening of the SEQ Bay door (Jones 1997, Apollo 12 ALSEP Off-load).

116:31:34 Bean: Okay. And we'll off-load the ALSEP. (Garbled).

116:31:39 Conrad: Nope. (Pause)

116:31:42 Bean: We ought to be able to move out with this thing.

116:31:44 Conrad: Okay.

116:31:48 Bean: The experiment bay looks real good.

116:31:49 Conrad: Yup.

116:31:50 Bean: The LM exterior looks beautiful the whole way around. Real good shape. Not a lot that doesn't look the way it did the day we launched it.

⁴⁰ The ground-elapsed time (GET) was the time clock in Houston that was started at the moment of launch.

116:32:02 Conrad: (Possibly pulling a lanyard to open the SEQ bay doors) Light one. (Pause)

116:32:12 Bean: Okay. Here we go, Pete. Ohhhh, up they go, babes. One ALSEP. (Pause)

[They have raised the doors that cover the cavity where the ALSEP packages are stowed.]

116:32:22 Conrad: There it is.

Figure 6-21. Apollo 12 LSJ: ALSEP Offload transcription (Jones 1997) (with permission)

There are three high-level (sub)activities that one can identify in this *OpenSEQBay* activity. First, there is a communication to MSC in Houston that they are ready to offload the ALSEP. This is the communication starting at 116:31:34. The issue to solve for the modeler is when this activity ends and the next activity begins. From the CDR communication at 116:32:02 we can infer that this is the time that the LMP actually opens the SEQ Bay door by pulling at the SEQ Bay door lanyard ribbons. So, we could start the activity of raising the SEQ Bay door around that time. However, from the video of the Apollo 14 ALSEP Offload it can be shown that before the LMP can pull the lanyard ribbons he has to grab them from the SEQ Bay, walk back until the ribbons are tight, and only then pull the ribbons to raise the SEQ Bay door. These activities have to happen before 116:32:02.

Table 6-3 shows the activities and sub-activities of the *Open SEQ Bay Door* activity for both LMP and CDR, mapped onto the communication transcribed in the Apollo LSJ. The actual names of the activities and sub-activities are more or less arbitrary, and conceptualize the modeler's *interpretation* of the observations of the Apollo 12 communication data and the Apollo 14 video data. However, these data and observations are strong evidence that these are the actual activities that are performed during the OpenSEQBay activity.

Table 6-3. Open SEQ Bay door activity

LMP		CDR	
Communicate Ready To Offload		Watching Opening SEQ Bay Door	
Activity	Communication	Communication	Activity
<i>Communicate Open Door</i>	116:31:34 Bean: Okay. And we'll off-load the ALSEP. (Garbled).		<i>Watch Opening SEQ Bay Door</i>
<i>Inspect SEQ Bay</i>		116:31:39 Conrad: Nope. (Pause)	
	116:31:42 Bean: We ought to be able to move out with this thing.		
		116:31:44 Conrad: Okay.	
	116:31:48 Bean: The experiment bay looks real good.		
		116:31:49 Conrad: Yup.	
Raising SEQ Bay Door			
Activity	Communication		
<i>Grab Lanyard Ribbons</i>	116:31:50 Bean: The LM exterior looks beautiful the whole way around. Real good shape. Not a lot that doesn't look the way it did the day we launched it.		
<i>Walk Back To Pull Ribbons Tight</i>			
<i>Pull Lanyard Ribbons</i>		116:32:02 Conrad: (Possibly pulling a lanyard to open the SEQ bay doors) Light one. (Pause)	
	116:32:12 Bean: Okay. Here we go, Pete. Ohhhh, up they go, babes. One ALSEP. (Pause)		
		116:32:22 Conrad: There it is.	

6.6 THE BEHAVIORAL MODEL

The activities from Table 6-3 are implemented in the Brahms model as the *OpenSEQBayDoor* composite-activity. Figure 6-22 shows this activity, its sub-activities and workframes.

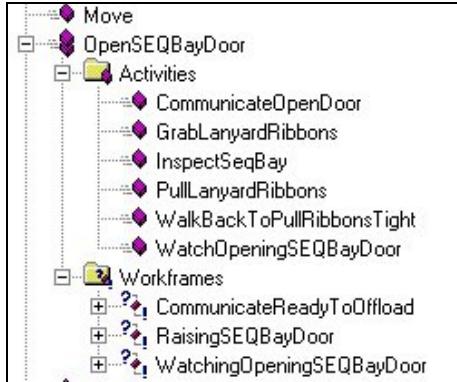


Figure 6-22. The OpenSEQ BayDoor composite-activity, sub-activities, and workframes

Each (sub)activity is “executed” by a workframe, which means that when an agent executes the workframe the activity is performed within the context of that workframe. As the first activity during the ALSEP offload, the CDR and LMP start walking to the area of the SEQ Bay. Walking to the SEQ Bay area to start opening the SEQ Bay door is modeled by the *Move* activity, seen at the top of Figure 6-22. Now that we defined the sub-activities and workframes of the *OpenSeqBayDoor* activity the question is; how do the CDR and LMP agents start this activity during the simulation? Figure 6-23 shows the workframes of the *AlsepOffload* activity that both agents can execute to offload the ALSEP.

The first workframe to fire—the highest-level workframe, but lowest in Figure 6-23—is the *OffloadingAlsep* workframe, which executes the *AlsepOffload* activity. Executing the *AlsepOffload* activity enables all the workframes, shown in Figure 6-22, it to potentially fire for the agent. Each of these workframes will execute lower-level activities, which are subsumed by the higher-level *AlsepOffload* activity.

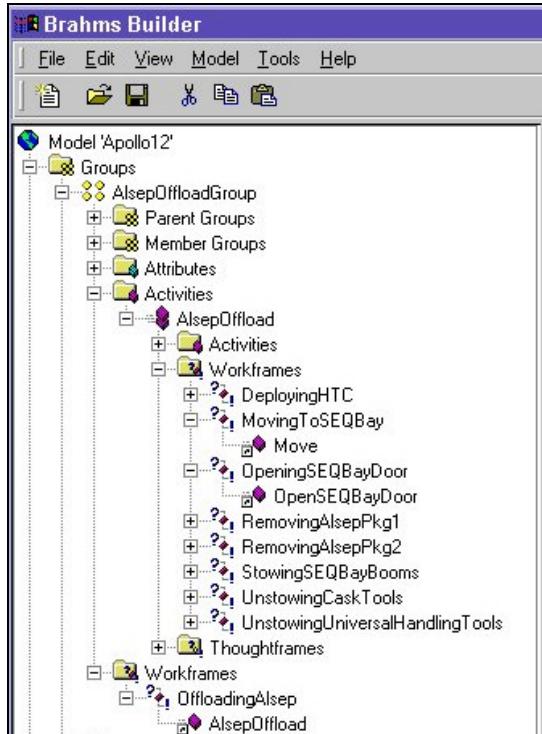


Figure 6-23. The AlsepOffload workframes

We can represent the relationship between workframes executing activities, containing other workframes that execute activities, etc, in a workframe-activity subsumption hierarchy as shown in Figure 6-24.

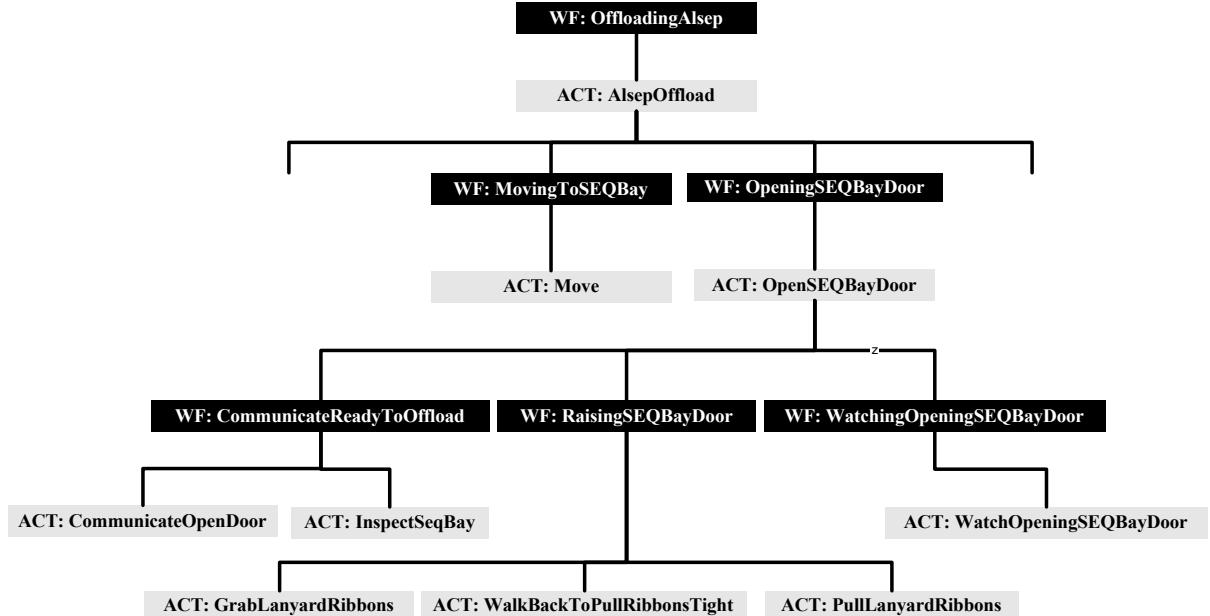


Figure 6-24. AlsepOffload workframe-activity subsumption hierarchy

Only one primitive activity can be active at any given time (i.e. at any clock-tick). This means that the order in which workframes at the same level in the hierarchy fire depends on two things; first, the conditions of the workframe that are to be matched to the beliefs of the agent, and second, the priority of the activities within the workframes.

6.6.1 Representing the work context

Figure 6-25 represents the parallel sequential order of the activities of the CDR and LMP from Table 6-3 and Figure 6-22. However, Figure 6-25 does not represent how the CDR and LMP came to do what they did. The question is not if we can describe the sequential activities of each of the astronauts, but rather, what makes the astronauts do what they do at each moment in time. What influence does the specific Apollo 12 situation have on when and how they do things? What influence do they have on each other's activity? Are they merely executing the OpenSeqBayDoor plan? Or, are they deciding what to do based on their personal knowledge of that plan? If so, we can represent the knowledge of the plan for each individual agent, and be done. This is the traditional knowledge-based systems approach, in which we represent the knowledge "inside people's heads" as production rules. However, what makes AI Bean know that he needs to open the door now, and what makes Pete Conrad know that he has to just watch the commander. What makes them react?

As much as it has to do with their knowledge of the plan for opening the SEQ Bay door, e.g. the steps that they have to go through, it is also a function of the *situation*, i.e. the situation specific context which they are part of. To start the opening door activity they not only need to know what is the right activity to be performing at that moment (according to the plan), but they also need to know that they need to go to the SEQ Bay. To go to the SEQ Bay they need to know where the SEQ Bay is. Once they are at the SEQ Bay, they can see if the door of the SEQ Bay is already open or not. They need to know that the ALSEP packages are located inside the SEQ Bay, and where the lanyard ribbons are located, et cetera. All this has to do with the context of the Apollo 12 mission.

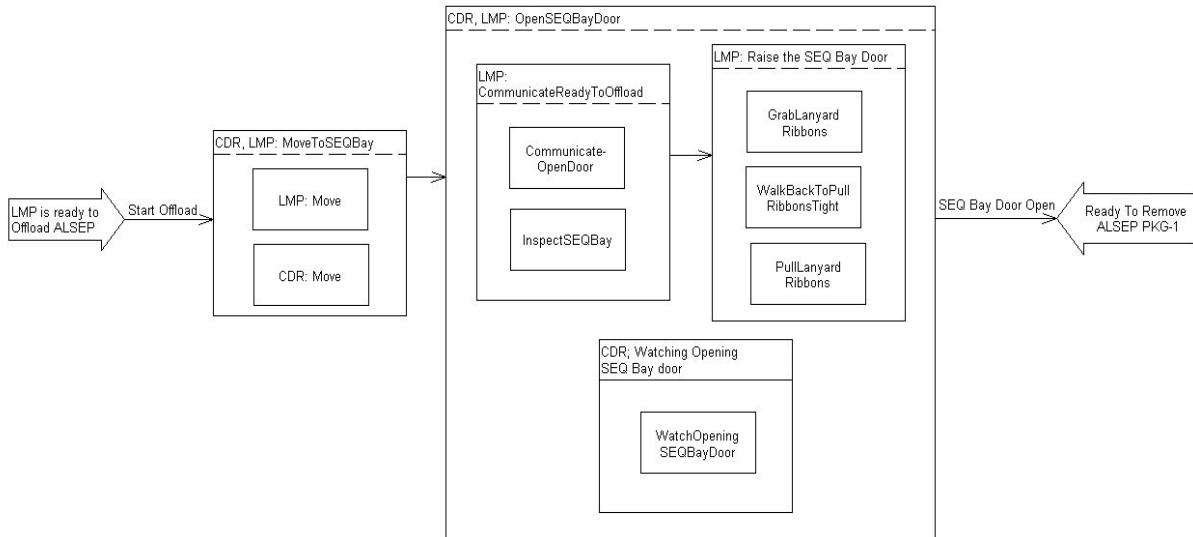


Figure 6-25. Open SEQ Bay door activity sequence model

In Brahms, we model context using three different modeling concepts. First, we model the geographical places at which people perform the work we are interested in (see the section on the Geographical Model, section 6.4). Second, we model all the objects and artifacts that are important in the work. In the case of opening the SEQ Bay door, we model the LM, the SEQ Bay, the ALSEP packages, as well as the lanyard ribbon used to pull open the SEQ Bay door (see the section on the Object Model, section 6.3). Third, we model the state of the world in terms of facts that can be detected by our agent astronauts. For example, when the astronauts walk over to the SEQ Bay, they immediately detect the state of the SEQ Bay door; is it open or closed? They notice the location of all objects in their surroundings. These are the world-facts that trigger the agents to react in certain ways, given the activity they are currently performing. For example, Al Bean would not always open the SEQ Bay door when he comes to the SEQ Bay and notices that the door is closed. He will only do this when he is in the activity of offloading the ALSEP, in particular when his next activity is to open the door. This is where the plan interacts with the situation specific context.

Why does Pete Conrad watch the LMP? What makes him perform that activity? His plan does not say to perform that activity (see Figure 6-2). Rather, this activity is a reaction on the LMP's activity of opening the door during the ALSEP Offload. It is the collaboration between the two astronauts that makes Pete Conrad watch his partner. He sees his partner grabbing the lanyard ribbons. He therefore knows what activity his partner is performing. It is a reaction to the situation and the context, as well as the fact that he is done performing his previous activity.

Figure 6-25 does not represent this context. The influence the context has on the sequence of the activities within Figure 6-25 determines the transitions. The interesting parts of Figure 6-25 are the transitions between the activities. What makes the model go from one state to another? This is what we want to uncover in the understanding of the work practice of the ALSEP offloading.

6.6.2 A narrative description of what happens in practice

Following is my interpretation of what happened during the opening of the SEQ Bay door and why the LMP and CDR do what they do:

When they are ready to offload the ALSEP, they first have to walk over to the SEQ Bay,

Once they arrive at the SEQ Bay, the two astronauts can see the SEQ Bay and can immediately notice that the door of the SEQ Bay is still closed. Of course, they both know that the SEQ Bay contains the two ALSEP packages, and since they are in the activity of offloading the packages they first need to open the door.

This triggers them to start the activity of opening the SEQ Bay door, and since the Apollo 12 ALSEP Offload plan states that the LMP, Al Bean, is to open the SEQ Bay door, he is the one that announces that they will now start with the ALSEP offload.

Since the LMP is the one who is to perform the activity of opening the SEQ Bay door, he is taking the first action and the next activity that is performed is the LMP inspecting the SEQ Bay. However, this is not a planned activity. It seems very likely that this activity is performed based on the astronauts' knowledge that mission control is interested in knowing how the SEQ Bay and the LM have withstood the long travel to the Moon. It is therefore a very important piece of information that the LMP communicates to Mission Control at this point.

Next, the LMP is ready to start raising the door. To do this he needs to grab the lanyard ribbon with which to pull open the door of the SEQ Bay. This means he must know where the lanyard ribbon is to pick-up the ribbon. He then walks back with the ribbon. He needs to tighten the ribbon to have enough leverage to pull the ribbon. Meanwhile, the CDR is standing close by and is watching the LMP, ready to help in case it is needed. Even though the offload plan does not specify any activity for the CDR at this moment, it is logical to infer that the CDR's objective is to closely watch what is happening just in case something happens. This is an activity in and of itself. The CDR would not do anything else even though he could. It seems that the two astronauts always know what high-level activity the other is performing. This means they are always ready to help each other.

After the SEQ Bay door is all the way open, the LMP lets the lanyard ribbon drop to the lunar surface.

I created this narrative, based on my analysis of the available mission data. Based on this short description of what is happening and what makes the two astronauts do what they do, we can list those elements in the context that are most important to include in the model of work practice of the astronauts.

- The SEQ Bay area location near the LM where this all takes place.
- The SEQ Bay and the fact that the SEQ Bay is part of the LM located in the SEQ Bay area location.
- The fact that the exterior of the LM and SEQ Bay are in good condition.
- The two ALSEP packages and the fact that they are located inside the SEQ Bay.
- The door of the SEQ Bay, and that it is closed.
- The lanyard ribbon with which to open the SEQ Bay door.
- The fact that both astronauts detect each other's activity.
- The fact that the LMP needs to carry the lanyard ribbon, and thus must know where this ribbon is located for him to pick it up.
- The fact that after the LMP has completed the activity of opening the door, the SEQ Bay door is open.
- The fact that after the SEQ Bay door is open the lanyard ribbon's location is the lunar surface, because the LMP lets it fall to the surface.
- The fact that both astronauts are noticing all these events and become aware of them, and react to them appropriately.

The challenge is to include these independent context elements into the model. Being able to include these elements in the model is what makes a Brahms model different from the sequential model of Figure 6-25. A sequential model, such as Figure 6-25, can only be executed in the pre-specified order, and does not allow for variations based on context. However, work practice is not the rigid execution of a pre-specified activity sequence. In practice, the sequence of activities depends on the situation. Is the door already open? Are the

packages inside the SEQ Bay or are they already on the ground, et cetera? In the next section, I will describe how these contextual and situational elements are included in the model.

6.6.3 Executing the OffloadingAlsep workframe

In this section, I describe the execution of the OffloadingAlsep workframe (Figure 6-26). The OffloadingAlsep workframe is the highest-level workframe (see Figure 6-23) in the AlsepOffload group (see Figure 6-24). Both LunarSurfaceAstronaut agents (LMP and CDR) inherit this workframe, and independently, can execute this workframe in order to start the ALSEP offload.

```

workframe OffloadingAlsep {
    repeat: false;
    variables:
        collectall(LmVoiceLoop) vlcoms;
        forone(LunarSurfaceAstronaut) pagt;
        forone(EMUSuit) emusuit;

    detectables:
        detectable DetectPartnerActivity {
            when (whenever)
                detect((the currentActivity of pagt = the currentActivity of pagt));
        }

        detectable DetectCoolingLevel {
            when (whenever)
                detect((the coolingLevel of emusuit = value));
        }

        detectable NoticeAlsepPkg1LocationChange {
            when (whenever)
                detect((the objectLocation of AlsepPkg1 = anylocation));
        }

        detectable NoticeAlsepPkg2LocationChange {
            when (whenever)
                detect((the objectLocation of AlsepPkg2 = anylocation));
        }

    when (knownval(the currentConceptualActivity of current = AlsepOffload) and
        not(the name of current = the name of vlcoms) and
        knownval(the communicationType of vlcoms = LmVoiceLoop) and
        knownval(the partner of current = pagt) and
        knownval(current contains emusuit))
        do {
            AlsepOffload(vlcoms, pagt);
        }
}

```

Figure 6-26. AlsepOffloading workframe

6.6.3.1 Variable bindings and preconditions

In order for the agents to execute a workframe (or thoughtframe) all preconditions of the workframe must evaluate to true. The Brahms scheduler will test each precondition and match the precondition to the beliefs of the agent. If there is a belief that matches the precondition, the precondition evaluates to true. The AlsepOffloading workframe in Figure 6-26 uses three variables within the preconditions to bind to objects and agents in the model. The first variable, *vlcoms* (i.e. voice-loop communicators), is used to match to the list of all agents (a “collectall” variable) who are members of the group LmVoiceLoop (see the “communicationType” precondition), except for the agent itself (see the “not” precondition). This variable is passed as a parameter to the AlsepOffload activity, where it is used to communicate to all the agents who are listening to the voice loop (see section 6.7). The second variable, *pagt*, is used to bind to the partner of the agent in the “partner” precondition. In case the agent executing the workframe (i.e. *current*) is the CDR, *pagt* is bound to the LMP agent (i.e. AlBean). In case the agent executing the workframe is the LMP, *pagt* is bound to the CDR agent (i.e. PeteConrad). This is because the LMP agent, AlBean, has an initial-belief

(the partner of current = PeteConrad),

while the CDR agent, PeteConrad, has an initial-belief

(the partner of current = AlBean).

For each of these two agents the precondition matches, and the *pagt* variable gets bound to the matched agent. The third variable, *emusuit*, is bound to the EMUSuit object of the *current* agent in the “contains” precondition.

All the above-mentioned preconditions are not used to actually guard the workframe from firing. These are all preconditions that are used to bind the variables to the appropriate objects and agents. The only real guard for the workframe is the “currentConceptualActivity” precondition. Not until the agent has the belief that his current (conceptual) activity is to offload the ALSEP, will this workframe fire. This shows that writing workframe preconditions has all the similar precondition control characteristics, such as the ordering, as writing preconditions for production rules in traditional expert systems (Clancey 1983), (Clancey 1988) and (Clancey 1992).

6.6.3.2 Detectables

The workframe in Figure 6-26 contains four detectables that are active as long as the agent is executing the AlsepOffload activity within the workframe (this is due to the “whenever” condition in each detectable). All four detectables have a “continue” action part (this is the default action of a detectable). This means that all the detectables are defined in the workframe so that the agent executing the workframe will detect any of the facts that match the detect-conditions, while performing the AlsepOffload activity without disrupting the activity itself. This makes it possible for the agent to notice certain facts in the world and react to them, because the facts turn into beliefs for the agent. This allows for the following reactive behavior on the part of the agent:

- The DetectPartnerActivity detectable makes sure that during the ALSEP offload activity the CDR and LMP are always aware of the activity their partner is performing. This enables the agent to react to their partner’s activity. There are multiple reasons for modeling that the astronauts on the lunar surface are always aware of this. The first one is that their activities are well choreographed and trained, and the second reason is that it is part of the NASA policy that there is a “buddy system” for EVA work performed by astronauts. This “buddy system” is a safety precaution. This way there is always someone who can help out. This means that the two lunar surface astronauts were very much in tune with what their partner was doing, even if they would be working on their own activity.
- The DetectCoolingLevel detectable models the fact that both astronauts are always aware of the cooling-level of their space suit. The fact that the astronauts are wearing their space suit makes this obvious. This detectable allows the agents to react to the cooling-level, and chance the level of cooling accordingly.
- The NoticeAlsepPkg1(2)LocationChance detectables speak for themselves. Whenever the location of either ALSEP package is changed, the agent will notice this, and can react accordingly. This is used to simulate the fact that when one of the agents lowers the ALSEP package from the SEQ Bay, both agents will become aware of the fact that the ALSEP package has changed its location from the SEQ Bay to the SEQ Bay area (i.e. the lunar surface). This belief is used to start/stop the activities for offloading the actual packages

6.6.3.3 Workframe execution

Following is a description of how Brahms executes the AlsepOffloading workframe during simulation of both the AlBean and the PeteConrad agent. The workframe is executed at time t=0 for both agents. This means that both agents are executing an instance of the workframe (Workframe Instantiation or WFI) at the same time. I’ll show the WFI for both agents by repeating the workframe from Figure 6-26, but then showing the bindings of the variables in preconditions, consequences, detectables, and activity parameters. Figure 6-27 and Figure 6-28 show the WFI for the agents AlBean and PeteConrad respectively.

```

workframe OffloadingAlsep {
    repeat: false;
    variables:
        collectall(LmVoiceLoop) vlcoms; => (PeteConrad, LmComCircuit)
        forone(LunarSurfaceAstronaut) pagt; => PeteConrad
        forone(EMUSuit) emusuit; => BeanEmuSuit

    detectables:
        detectable DetectPartnerActivity {
            when (whenever)
                detect((the currentActivity of PeteConrad = the currentActivity of PeteConrad));
        }

        detectable DetectCoolingLevel {
            when (whenever)
                detect((the coolingLevel of BeanEmuSuit = value));
        }

        detectable NoticeAlsepPkg1LocationChange {
            when (whenever)
                detect((the objectLocation of AlsepPkg1 = anylocation));
        }

        detectable NoticeAlsepPkg2LocationChange {
            when (whenever)
                detect((the objectLocation of AlsepPkg2 = anylocation));
        }

    when (knownval(the currentConceptualActivity of AIBean = AlsepOffload) and
           not(the name of AIBean = the name of (PeteConrad, LmComCircuit)) and
           knownval(the communicationType of (PeteConrad, LmComCircuit) = LmVoiceLoop) and
           knownval(the partner of AIBean = PeteConrad) and
           knownval(AIBean contains BeanEmuSuit))
    do {
        AlsepOffload((PeteConrad, LmComCircuit), PeteConrad);
    }
}

```

Figure 6-27. AlsepOffloading WFI for agent AIBean

```

workframe OffloadingAlsep {
    repeat: false;
    variables:
        collectall(LmVoiceLoop) vlcoms; => (AIBean, LmComCircuit)
        forone(LunarSurfaceAstronaut) pagt; => AIBean
        forone(EMUSuit) emusuit; => ConradEmuSuit

    detectables:
        detectable DetectPartnerActivity {
            when (whenever)
                detect((the currentActivity of AIBean = the currentActivity of AIBean));
        }

        detectable DetectCoolingLevel {
            when (whenever)
                detect((the coolingLevel of BeanEmuSuit = value));
        }

        detectable NoticeAlsepPkg1LocationChange {
            when (whenever)
                detect((the objectLocation of AlsepPkg1 = anylocation));
        }

        detectable NoticeAlsepPkg2LocationChange {
            when (whenever)
                detect((the objectLocation of AlsepPkg2 = anylocation));
        }

    when (knownval(the currentConceptualActivity of PeteConrad = AlsepOffload) and
          not(the name of PeteConrad = the name of (AIBean, LmComCircuit)) and
          knownval(the communicationType of (AIBean, LmComCircuit) = LmVoiceLoop) and
          knownval(the partner of PeteConrad = AIBean) and
          knownval(PeteConrad contains ConradEmuSuit))
        do {
            AlsepOffload((AIBean, LmComCircuit), AIBean);
        }
}

```

Figure 6-28. AlsepOffloafing WFI for agent PeteConrad

The next two sections describe how the CDR and LMP agents are both performing the AlsepOffload activity, and in doing so collaborating in opening the SEQ Bay door.

6.6.4 Performing the AlsepOffload activity

After the firing of the OffloadingAlsep workframe both agents execute the AlsepOffload composite activity (see Figure 6-29). For each agent, the simulation engine changes the agent Activity-Context Tree (ACT) based on the workframes and thoughtframes in the composite activity that execute. An ACT consists of WFI's and the current activity context of the selected workframe⁴¹. In this section, I will show how the simulation engine scheduler schedules the activities for each of the lunar surface astronaut agent. To do this, I first provide the source code of the workframes of the AlsepOffload composite activity. Next, I will show the ACT for both the AIBean and the PeteConrad agent for two simulation events (steps). I will show the change in the ACTs as the beliefs of the agents and the world-facts change over time, due to the workframe execution and the agent's reasoning, interaction with other agents/objects and their environment.

⁴¹ Only one workframe instantiation can be fired at any time, which means that there is always only one current activity and therefore only one current activity-context.

```

workframe MovingToSEQBay {
    repeat: false;
    detectables:
        detectable DetectSEQBayDoor {
            when (100)
                detect((the door of SEQBay = value));
        }
    when (not(the agentLocation of current = SEQBayArea))
    do {
        conclude((the currentActivity of current = MoveActivity), bc:100, fc:100);
        Move(SEQBayArea, 5, 1);
        conclude((the nextActivity of current = OpenSEQBayDoorActivity), bc:100, fc:0);
    }
}

workframe OpeningSEQBayDoor {
    repeat: false;
    variables:
        collectall(AlsepPackage) alseppkgs;

    when (knownval(the agentLocation of current = SEQBayArea) and
        knownval(the door of SEQBay = closed) and
        not(the objectLocation of alseppkgs = SEQBayArea))
    do {
        conclude((the currentActivity of current = OpenSEQBayDoorActivity), bc:100, fc:100);
        OpenSEQBayDoor(vlcoms);
        conclude((the door of SEQBay = open), bc:100, fc:100);
    }
}

```

Figure 6-29. Workframes within the composite AlsepOffload activity

1. STEP 1: time t = 0

For each Lunar Surface Agent, the scheduler checks the preconditions of all the workframes and thoughtframes in the AlsepOffload activity, based on the agent's current belief set.

AIBean:

Current Belief Set:

```

t=0 => BELV: The currentConceptualActivity of AIBean = AlsepOffload
t=0 => BELV: The agentLocation of AIBean = Surveyor Crater
t=0 => BELV: The agentLocation of PeteConrad = Surveyor Crater
t=0 => BELV: The agentLocation of DickGordon = Yankee Clipper
t=0 => BELV: The agentLocation of EdGibson = MissionControlCenter
t=0 => BELV: The partner of AIBean = PeteConrad
t=0 => BELV: SEQBay contains AlsepPkg1
t=0 => BELV: SEQBay contains AlsepPkg2
t=0 => BELV: SEQBay contains OffloadChecklistDecal
t=0 => BELV: SEQBay contains Pkg1LanyardRibbons
t=0 => BELV: SEQBay contains Pkg2LanyardRibbons
t=0 => BELV: SEQBay contains SEQBayDoorLanyardRibbons
t=0 => BELV: AIBean contains BeanEMUSuit
t=0 => BELV: AIBean contains LmpCuffCheckList
t=0 => BELV: BeanEMUSuit contains BeanHasselblad70mm
t=0 => BELV: PeteConrad contains ConradEMUSuit
t=0 => BELV: PeteConrad contains CdrCuffCheckList
t=0 => BELV: ConradEMUSuit contains ConradHasselblad70mm

```

Precondition Matching:

workframe MovingToSEQBay:

Prec: not(the agentLocation of current = SEQBayArea)
 TRUE, based on BELV: The agentLocation of AIBean = Surveyor Crater

workframe OpeningSEQBayDoor

Prec: knownval(the agentLocation of current = SEQBayArea)
 FALSE, based on BELV: The agentLocation of AIBean = Surveyor Crater
 Prec: knownval(the door of SEQBay = closed)
 FALSE, based on NO belief about the door of SEQBay

Prec: not(the objectLocation of alseppkgs = SEQBayArea)
 TRUE, based on NO belief about the location of AlsepPkg1 and AlsepPkg2

Activity-Context Tree:

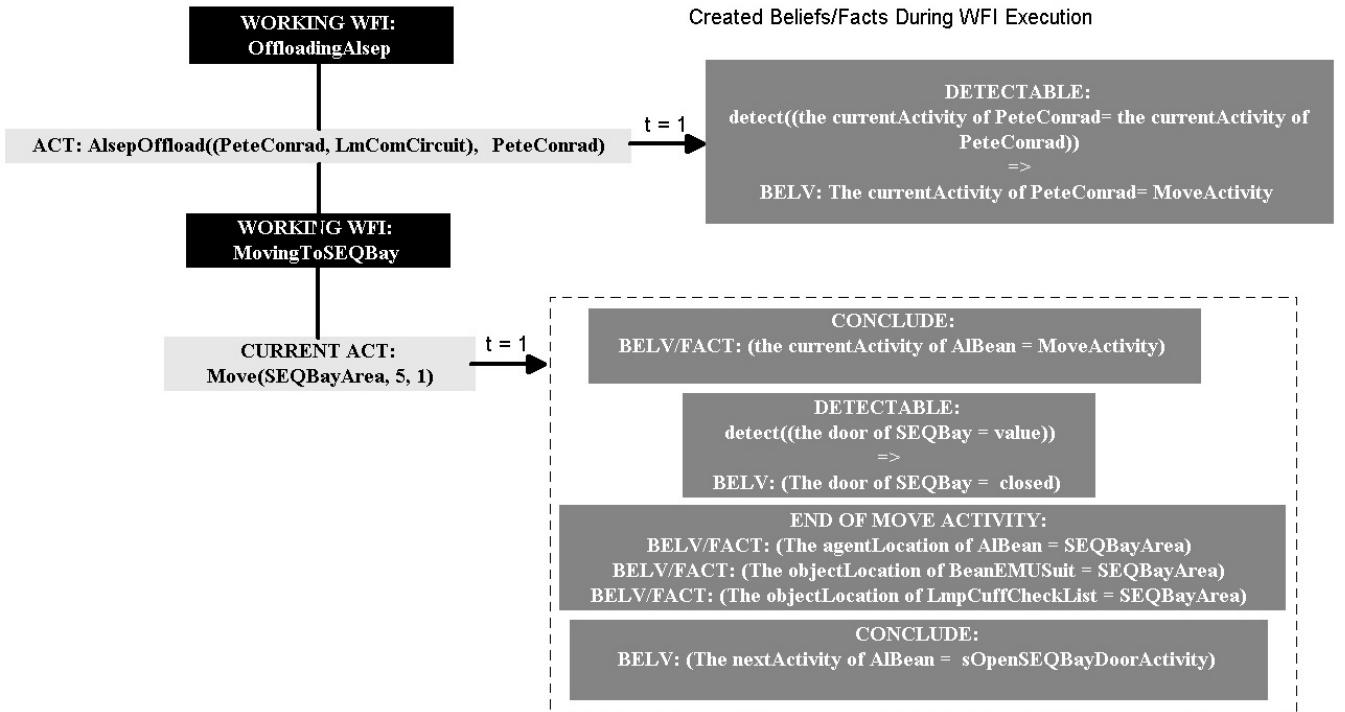


Figure 6-30. AIBean's Step 1 Activity-Context Tree

PeteConrad:

Current Belief set:

t=0 => BELV: The currentConceptualActivity of PeteConrad = AlsepOffload
 t=0 => BELV: The agentLocation of PeteConrad = SurveyorCrater
 t=0 => BELV: The agentLocation of AlBean = SurveyorCrater
 t=0 => BELV: The agentLocation of DickGordon = YankeeClipper
 t=0 => BELV: The agentLocation of EdGibson = MissionControlCenter
 t=0 => BELV: The partner of PeteConrad = AlBean
 t=0 => BELV: SEQBay contains AlsepPkg1
 t=0 => BELV: SEQBay contains AlsepPkg2
 t=0 => BELV: SEQBay contains OffloadChecklistDecal
 t=0 => BELV: SEQBay contains Pkg1LanyardRibbons
 t=0 => BELV: SEQBay contains Pkg2LanyardRibbons
 t=0 => BELV: SEQBay contains SEQBayDoorLanyardRibbons
 t=0 => BELV: AlBean contains BeanEMUSuit
 t=0 => BELV: AlBean contains LmpCuffCheckList
 t=0 => BELV: BeanEMUSuit contains BeanHasselblad70mm
 t=0 => BELV: PeteConrad contains ConradEMUSuit
 t=0 => BELV: PeteConrad contains CdrCuffCheckList
 t=0 => BELV: ConradEMUSuit contains ConradHasselblad70mm

Precondition Matching:

workframe MovingToSEQBay:

Prec: not(the agentLocation of current = SEQBayArea)
 TRUE, based on BELV: The agentLocation of PeteConrad = SurveyorCrater

workframe OpeningSEQBayDoor

Prec: knownval(the agentLocation of current = SEQBayArea)
 FALSE, based on BELV: The agentLocation of PeteConrad = SurveyorCrater
 Prec: knownval(the door of SEQBay = closed)
 FALSE, based on NO belief about the door of SEQBay
 Prec: not(the objectLocation of alseppkgs = SEQBayArea)

TRUE, based on NO belief about the location of AlsepPkg1 and AlsepPkg2

Activity-Context Tree:

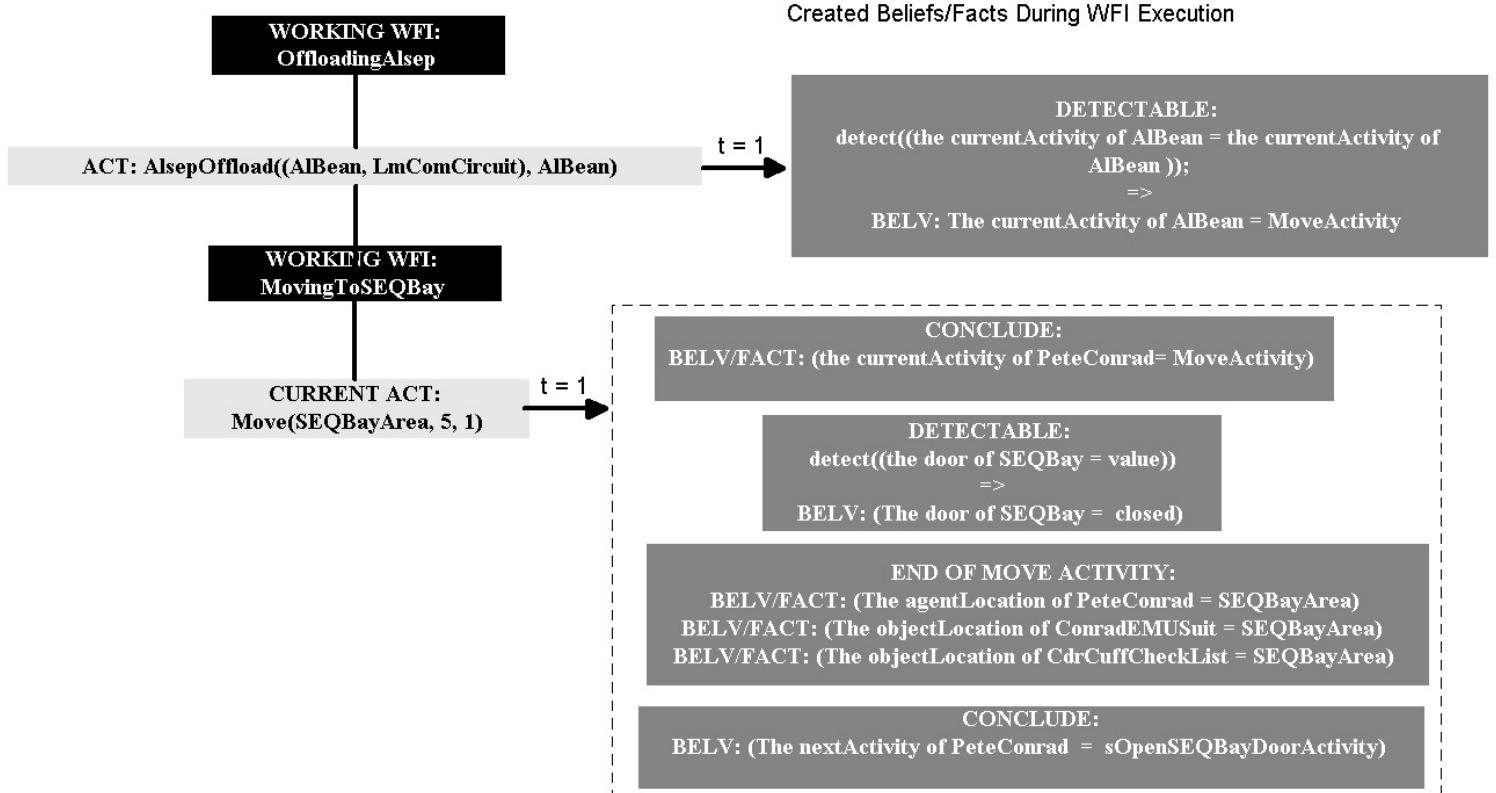


Figure 6-31. PeteConrad's Step 1 Activity-Context Tree

Both agents are executing a move-activity from their current location (i.e. Surveyor Crater) to the SEQBayArea, as can be seen in Figure 6-30 and Figure 6-31. As the agents move to the new location, the objects that they are containing (i.e. cuff checklist and EMU suit) are automatically moved with them to the new location. As the agents arrive in the new location, they detect that the SEQ Bay door is still closed. Also shown in Figure 6-30 and Figure 6-31, the agents automatically notice (i.e. the engine automatically creates the beliefs for the agents) the location of all other objects and agents that are also in the new location; i.e. the location of the other agent, its cuff checklist and EMU Suit, the LM, and the SEQ Bay. Both agents also detect each other's current activity, through the DetectPartnerActivity detectable in the AlsepOffload activity. Lastly, the agents receive a belief about their next activity to open the SEQ Bay door.

When the simulation clock has increased by one, the following (partial) situation exists:

2. STEP 2: time $t = 1$

For each Lunar Surface Agent, the scheduler checks the preconditions of all the workframes and thoughtframes in the AlsepOffload activity, based on the agent's current belief set.

AIBean:

Current Belief Set:

- $t=1 \Rightarrow BELV: \text{The currentActivity of AIBean} = \text{OpenSEQBayDoorActivity}$
- $t=1 \Rightarrow BELV: \text{The nextActivity of AIBean} = \text{OpenSEQBayDoorActivity}$
- $t=1 \Rightarrow BELV: \text{The door of SEQBay} = \text{closed}$
- $t=1 \Rightarrow BELV: \text{The currentActivity of PeteConrad} = \text{MoveActivity}$
- $t=1 \Rightarrow BELV: \text{The objectLocation of SEQBay} = \text{SEQBayArea}$
- $t=1 \Rightarrow BELV: \text{The objectLocation of LM} = \text{SEQBayArea}$
- $t=1 \Rightarrow BELV: \text{The agentLocation of AIBean} = \text{SEQBayArea}$
- $t=1 \Rightarrow BELV: \text{The objectLocation of BeanEMUSuit} = \text{SEQBayArea}$
- $t=1 \Rightarrow BELV: \text{The objectLocation of LmpCuffCheckList} = \text{SEQBayArea}$

```

t=1 => BELV: The agentLocation of PeteConrad = SEQBayArea
t=1 => BELV: The objectLocation of ConradEMUSuit = SEQBayArea
t=1 => BELV: The objectLocation of CdrCuffCheckList = SEQBayArea
t=1 => BELV: The currentActivity of AlBean = MoveActivity
t=0 => BELV: The currentConceptualActivity of AlBean = AlsepOffload
t=0 => BELV: The agentLocation of DickGordon = YankeeClipper
t=0 => BELV: The agentLocation of EdGibson = MissionControlCenter
t=0 => BELV: The partner of AlBean = PeteConrad
t=0 => BELV: SEQBay contains AlsepPkg1
t=0 => BELV: SEQBay contains AlsepPkg2
t=0 => BELV: SEQBay contains OffloadChecklistDecal
t=0 => BELV: SEQBay contains Pkg1LanyardRibbons
t=0 => BELV: SEQBay contains Pkg2LanyardRibbons
t=0 => BELV: SEQBay contains SEQBayDoorLanyardRibbons
t=0 => BELV: AlBean contains BeanEMUSuit
t=0 => BELV: AlBean contains LmpCuffCheckList
t=0 => BELV: BeanEMUSuit contains BeanHasselblad70mm
t=0 => BELV: PeteConrad contains ConradEMUSuit
t=0 => BELV: PeteConrad contains CdrCuffCheckList
t=0 => BELV: ConradEMUSuit contains ConradHasselblad70mm

```

Precondition Matching:

workframe MovingToSEQBay:

Prec: not(the agentLocation of current = SEQBayArea)
 FALSE, based on BELV: The agentLocation of AlBean = SEQBayArea

workframe OpeningSEQBayDoor

Prec: knownval(the agentLocation of current = SEQBayArea)
 TRUE, based on BELV: The agentLocation of AlBean = SEQBayArea
 Prec: knownval(the door of SEQBay = closed)
 TRUE, based on BELV: The door of SEQBay = closed
 Prec: not(the objectLocation of alseppkgs = SEQBayArea)
 TRUE, based on NO belief about the location of AlsepPkg1 and AlsepPkg2

Activity-Context Tree:

As the move-activity in Step 1 (Figure 6-30) ends, in the next clock-tick (t=1) the ACT for agent AlBean changes. The agent is still within the OffloadAlsep activity, because there are still workframes that are in the working-state. The MovingToSEQBay workframe has finished executing, its preconditions are false, and its repeat-variable has the value “false”. Therefore, the working WFI is finished and stops. However, the preconditions of the OpeningSEQBayDoor workframe have become true in the same clock-tick (t=1), and a new working WFI for this workframe is created (see Figure 6-32). Next, the composite activity OpenSEQBayDoor in this WFI gets executed. Consequently, the preconditions of all workframes in it are checked. It turns out that for agent AlBean, the preconditions of two of the three workframes evaluate to “true”. This means that WFI’s are created for both the RaiseSEQBayDoor and CommunicateReadyToOffload workframes, and their state becomes “available”. Since there can only be one WFI working at that level in the ACT, the engine solves the conflict by comparing the priorities of the two available WFI’s. The priority of a WFI is equal to the priority of the highest activity priority within it. In this case, the priority of the RaiseSEQBayDoor WFI is zero (0) and that of the CommunicateReadyToOffload WFI is ten (10). Consequently, the CommunicateReadyToOffload WFI becomes the working WFI, and its first activity Talk the current activity, i.e. the agent’s activity that is being executed.

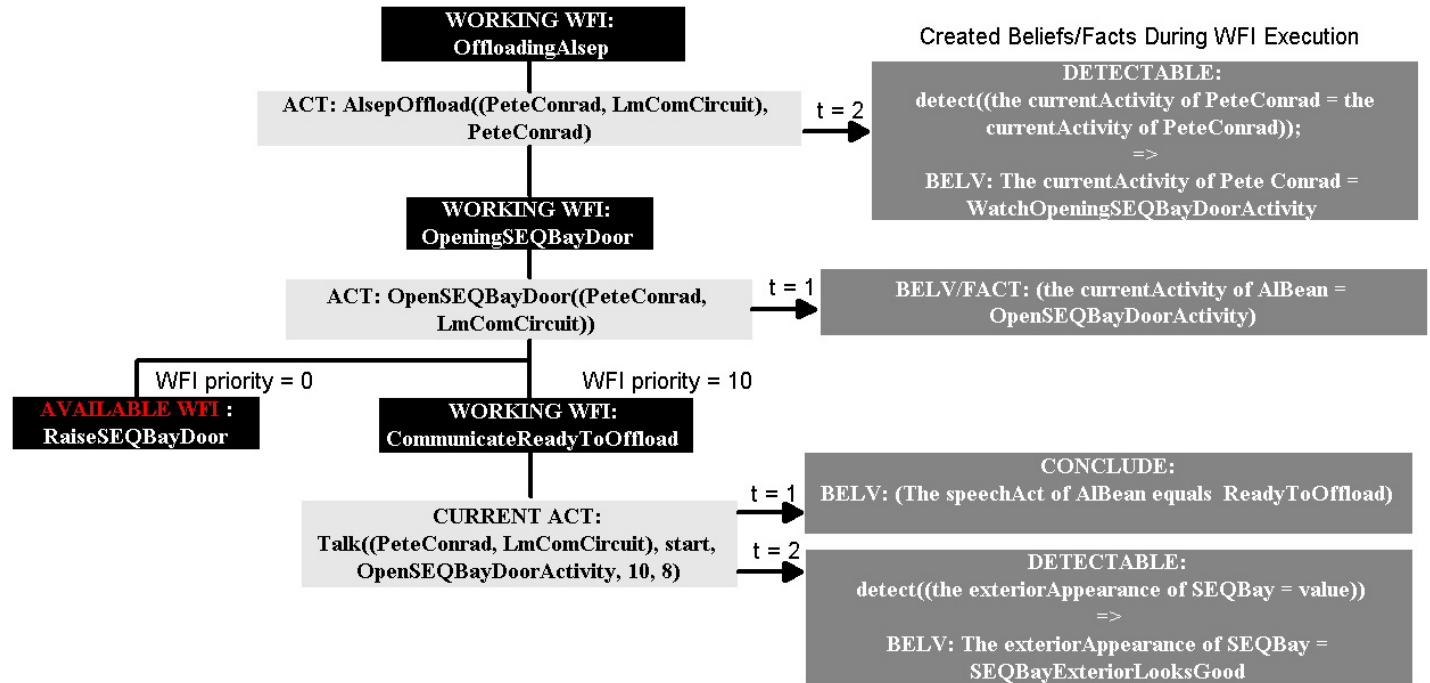


Figure 6-32. AIBean's Step 2 Activity-Context Tree

Also shown in Figure 6-32 are the detectables in the workframes OffloadingAlsep and CommunicateReadyToOffload firing in step 2 ($t=1$). However, in both cases the beliefs are not created until $t=2$, as a result of detecting the facts at $t=1$. This occurs at $t=2$ and not $t=1$, due to the clock-based simulation engine. The current activity Talk starts execution at $t=1$. This means that all the detectables in the working WFIs are checked at $t=1$. The beliefs are not created until the next clock-tick, $t=2$.⁴²

PeteConrad:

Current Belief Set

```

t=1 => BELV: The currentActivity of PeteConrad = WatchOpeningSEQBayDoorActivity
t=1 => BELV: The currentActivity of PeteConrad = OpenSEQBayDoorActivity
t=1 => BELV: The nextActivity of PeteConrad = OpenSEQBayDoorActivity
t=1 => BELV: The door of SEQBay = closed
t=1 => BELV: The currentActivity of AlBean = MoveActivity
t=1 => BELV: The fieldOfVision of PeteConrad = AlsepPackageInSeqBay
t=1 => BELV: The objectLocation of SEQBay = SEQBayArea
t=1 => BELV: The objectLocation of LM = SEQBayArea
t=1 => BELV: The agentLocation of AlBean = SEQBayArea
t=1 => BELV: The objectLocation of BeanEMUSuit = SEQBayArea
t=1 => BELV: The objectLocation of LmpCuffCheckList = SEQBayArea
t=1 => BELV: The agentLocation of PeteConrad = SEQBayArea
t=1 => BELV: The objectLocation of ConradEMUSuit = SEQBayArea
t=1 => BELV: The objectLocation of CdrCuffCheckList = SEQBayArea
t=1 => BELV: The currentActivity of PeteConrad = MoveActivity
t=0 => BELV: The currentConceptualActivity of PeteConrad = AlsepOffload
t=0 => BELV: The agentLocation of DickGordon = YankeeClipper
t=0 => BELV: The agentLocation of EdGibson = MissionControlCenter
t=0 => BELV: The partner of PeteConrad = AlBean
t=0 => BELV: SEQBay contains AlsepPkg1
t=0 => BELV: SEQBay contains AlsepPkg2
t=0 => BELV: SEQBay contains OffloadChecklistDecal
t=0 => BELV: SEQBay contains Pkg1LanyardRibbons
t=0 => BELV: SEQBay contains Pkg2LanyardRibbons
t=0 => BELV: SEQBay contains SEQBayDoorLanyardRibbons
t=0 => BELV: AlBean contains BeanEMUSuit
t=0 => BELV: AlBean contains LmpCuffCheckList

```

⁴² In our new Java-based discrete event simulation engine the beliefs will be created at the same clock-tick, i.e. $t=1$.

$t=0 \Rightarrow \text{BELV: BeanEMUSuit contains BeanHasselblad70mm}$
 $t=0 \Rightarrow \text{BELV: PeteConrad contains ConradEMUSuit}$
 $t=0 \Rightarrow \text{BELV: PeteConrad contains CdrCuffCheckList}$
 $t=0 \Rightarrow \text{BELV: ConradEMUSuit contains ConradHasselblad70mm}$

Precondition Matching:

workframe MovingToSEQBay:

Prec: not(the agentLocation of current = SEQBayArea)
 FALSE, based on BELV: The agentLocation of PeteConrad = SEQBayArea

workframe OpeningSEQBayDoor

Prec: knownval(the agentLocation of current = SEQBayArea)
 TRUE, based on BELV: The agentLocation of PeteConrad = SEQBayArea
 Prec: knownval(the door of SEQBay = closed)
 TRUE, based on BELV: The door of SEQBay = closed
 Prec: not(the objectLocation of alsepPkgs = SEQBayArea)
 TRUE, based on NO belief about the location of AlsepPkg1 and AlsepPkg2

Activity-Context Tree:

As the PeteConrad agent also comes into the SEQBayArea location, he also starts working on the OpenSeqBayDoor activity. Potentially the agent can execute the same workframes as AlBean. However, due to the belief-set of the agent PeteConrad, it will fire the WatchingOpeningSEQBayDoor workframe, which therefore becomes the working WFI.

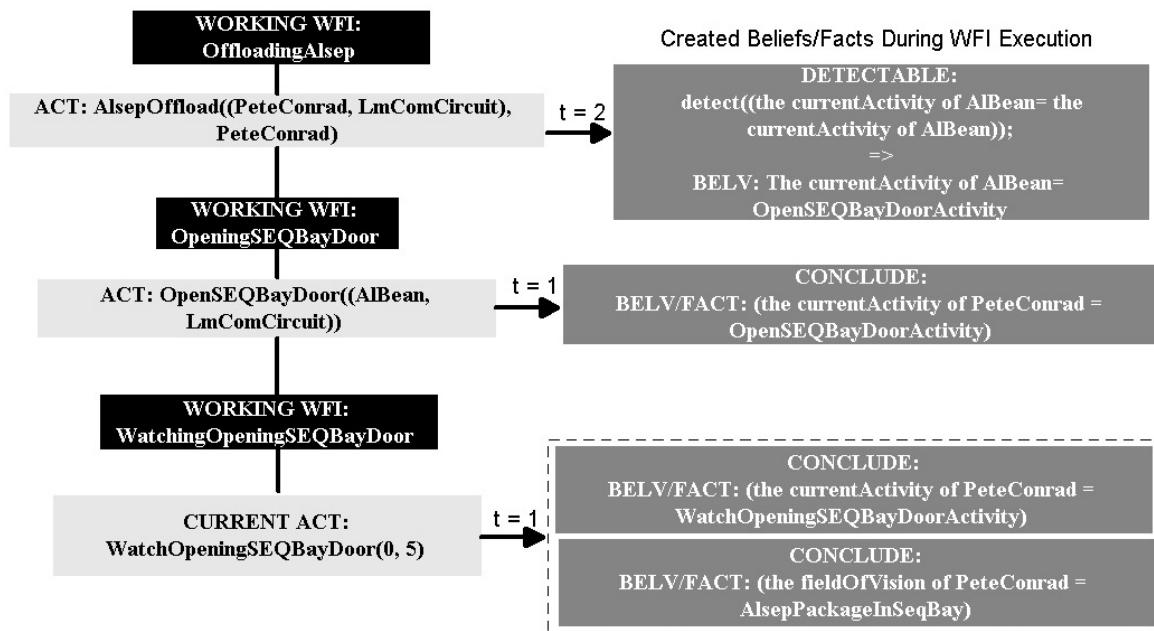


Figure 6-33. PeteConrad's Step 2 Activity-Context Tree

6.6.5 Viewing the simulation results

In this section I show the results of the simulation of the OpenSEQBayDoor activity, as described in the previous sections. Figure 6-34 shows the ACTs of the AlsepOffload activity performed by both the AlBean and the PeteConrad agent, as described in section 6.6.4, as well as the communication between the two agents. While performing the AlsepOffload composite activity, both agents are within the *OpenSEQBayDoor* activity. While AlBean is performing the activities within the *CommunicateReady* and the *RaisingSEQBayDoor* workframe, the PeteConrad agent is performing the activities within the *WatchingOpenSEQBayDoor* workframe. The grain-size of the simulation is one second. This means that the simulation engine changes the ACT for every agent and object every second of simulated time. We can therefore say that the simulation is a second by second model of the work practice of the lunar surface astronauts. Figure 6-34 also shows the location the agent was in when performing the activity. As an

overlay, the dotted arrows show the communication of beliefs between agents AlBean and PeteConrad. The direction of the arrows show the direction in which the beliefs are being communicated, while the little square box at the start of the arrow shows the agent that is performing the communication.

Figure 6-34 is a screen shot from the AgentViewer application⁴³. The AgentViewer application takes as input a Brahms Simulation History database⁴⁴. This history database contains the historical situation-specific model data of a particular simulation run. The AgentViewer application creates a graphical representation of the activity of agents and objects during a simulation.

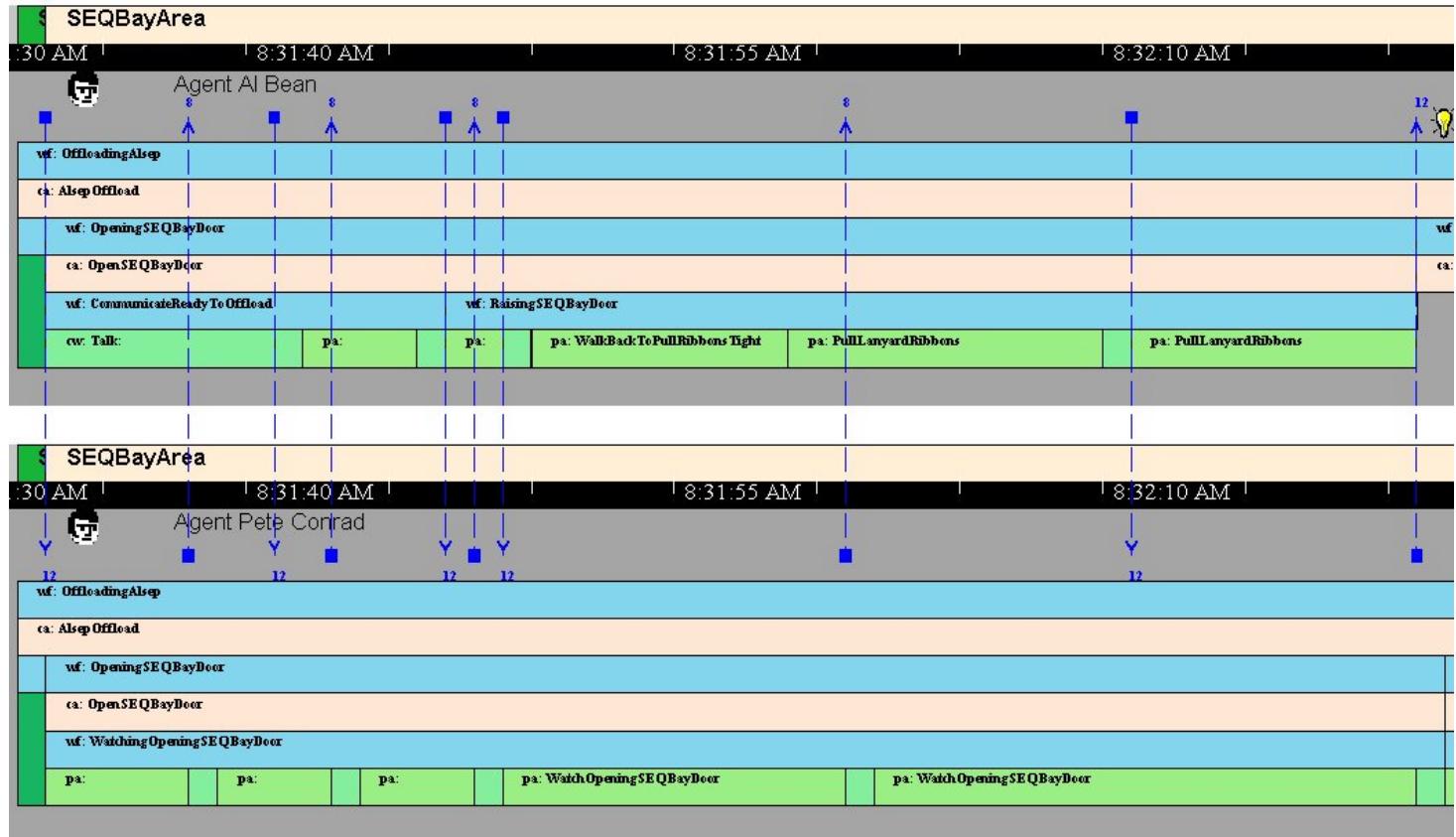


Figure 6-34. AlsepOffload activity agent timeline

Grouping a number of important data about the activity of the agent during the simulation into an agent workframe-activity hierarchy shows the ACT of an agent or object, at any time during the simulation. Each agent's ACT consists of a number of "bars." Each bar is an object that can be manipulated in the AgentViewer.

At the top of each agent's ACT there is the *location bar*. The location bar shows the movement of the agent throughout its activities. When an agent changes location the color of the location bar changes⁴⁵. In Figure 6-34 both agents start in the same initial location. This is the Apollo 12 LandingSite area (Surveyor Crater). You can see that the next location both agents are in is the SEQBayArea location.

The next bar in the agent's ACT is the *time-line bar*. This bar shows the simulation time. Figure 6-34 shows that the AlsepOffload activity starts just after 8:31:30 AM (in fact the simulation clock starts at time 8:31:32 AM). Each thin white line in the time-line bar shows a 5-second interval. Consequently, Figure 6-34 shows an activity interval of about 50 seconds (from 8:31:30 AM until about 8:32:23 AM).

⁴³ The AgentViewer application is a stand-alone Visual Basic application we developed for viewing the results of a simulation.

⁴⁴ The history database is a complex relational database containing the simulation data preserving their relationships.

⁴⁵ An agent does not effectively change its location until the simulation engine has finished a move activity and consequently positions the agent into the new location. The agent's location during the move activity stays unchanged, even though the agent is moving, and should thus not be in any location. Brahms is not modeling the movement of agents during the execution of a move-activity.

The third bar is the agent's *name bar*, with the name of the agent and an agent icon). This bar shows which agent or object⁴⁶ is being displayed

The fourth and final bar is the *workframe-activity bar*. This is the bar that shows the execution of the workframes, activities, and thoughtframes for the agent. Workframes are represented as blue bars that start with the letters "wf", for workframe, and the name of the workframe (if it fits within the graphics block). Underneath a workframe bar there can either be a flesh-colored bar, or a green-colored bar. A flesh-colored bar represents a composite activity, and starts with the letters "ca", for composite activity. A green-colored bar is a primitive-, move-, communicate-, or create-object activity. These are always the lowest level activities. Each type of primitive activity is indicated by a different shade of green. Other than a color indication; a primitive activity is indicated by the letters "pa", for primitive activity; a move activity by the letters "mv", for move; a communicate activity by the letters "cw", for communicating-with; a create-object activity by the letters "co", for create-object. When the size of the graphics block is large enough to contain the name of the activity it is shown as well. If not, the name is shown when the user moves the mouse over the activity or workframe box.

6.7 VOICE-LOOP COMMUNICATION

One of the most important aspects of work practice is the way people communicate. The communication to and from the Apollo Lunar Surface was made possible by the Extra-Vehicular Communication System (EVCS). The EVCS was a communication relay system that communicated voice from the astronauts via their EMU suits to the LM and via the LM, using a S-Band antenna, to mission control. The voice of the CapCom was communicated back to the LM and the astronauts via the same system (see Figure 6-35). This way the lunar surface astronauts and CapCom were in constant two-way communication. The CMP and CapCom had a similar communication system via the CM. The two lunar surface astronauts where operating their communication system in dual mode, which meant that they were always able to hear each other. However, the CMP was not in direct communication with the lunar surface astronauts, and was therefore not always able to hear them.

In this section I describe how the EVCS, or as I have named it, the voice-loop communication has been modeled in Brahms.

6.7.1 Communication delay

Conversational overlaps are a normal part of human dialog, and humans are pretty well apt to deal with this phenomenon. However, the communication delay from Earth to the Moon is significantly larger than the face-to-face or phone communication on Earth. The one-way delay, to Earth and to the Moon, is one and a quarter (1.25) second. This means a minimum of two and a half (2.5) seconds round-trip communication delay. If one of the astronauts made an utterance, the CapCom would hear the utterance one-and-a-quarter second later. If the CapCom would respond immediately, the astronauts would not hear this response until one and a quarter second later, which means a total of, at minimum, two and a half seconds.

⁴⁶ Objects have a different object icon.

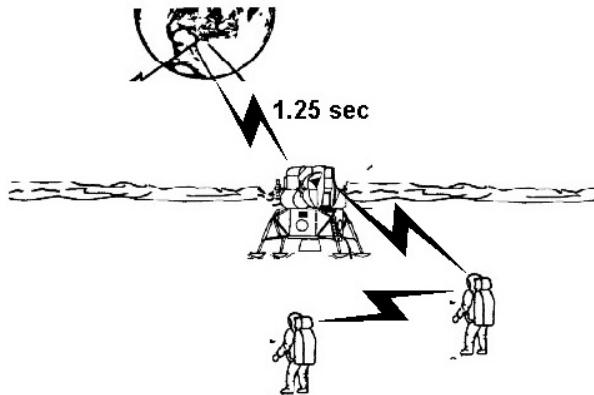


Figure 6-35. Extra-Vehicular Communication System

During the Apollo missions the communication delay sometimes lead to problematic communication patterns, as is shown in the example from Apollo 17 from the section "Journal Preparation and Structure" in the Apollo LSJ (Jones 1997).

In the following example, we imagine CapCom Bob Parker giving Gene Cernan instructions on parking the Rover. Before Cernan hears Parker, he starts to make a comment about where he is parked. He then stops talking, listens to Parker (who doesn't stop talking), responds, and then continues with his comment.

Parker: Gene, just a reminder that we want a Rover (garbled)...

Cernan: Bob, we've stopped next to...(Hears Parker)

Parker: ...(heading) of 045; and, when you get out, we'll need readouts.

Cernan: (Responding to Parker) Okay, Bob. We've parked next to one of the fresh craters that shows up on the map.

Generally, when someone's utterance ends with ellipses and his next utterance begins with ellipses, the reader should infer that the speaker kept talking under the overlapping remark. When someone's utterance ends with ellipses but his next utterance does not begin with ellipses, the reader should infer either a break in thought or a pause to listen. Unintelligible dialog is indicated by the editorial comment "garbled". Unintelligible dialog is often associated with overlapping conversations and in this illustration, on the continuation of Parker's utterance I have indicated the likely missing word. "

Although in this example Eric Jones is referring to the transcription as done for the Apollo LSJ, the fact of the matter is that if one wants to model the communication utterances of the astronauts and their impact on work practice, we have to model the one and a quarter delay for each communication event.

6.7.2 Modeling the communication to Earth

The voice-loop in context of the Apollo missions is the inter-communication system between the astronauts on the Moon and the CapCom at MSC in Houston.

There is a significant difference between voice-loop communication and face-to-face (f-2-f) communication. First, and foremost, f-2-f communication is bounded to geographical location of the agents. This means that the agents have to be in the same location to be able to engage in a f-2-f communication activity. This is referred to as Same Time/Same Place (STSP) communication (Chapter 3.2.4.4). In voice-loop communication there is no restriction on the geographical location of the engaging agents. The agents can be in any location, indeed even on Earth and on the Moon.

Another difference is the fact that in a voice-loop communication there is no need to “go to” somebody before a communication can take place. This is similar to a phone communication (Same Time/Different Place (STDP) communication). However, different from a phone communication, there is no need to “call” someone before the communication can start. Therefore, a voice-loop communication is a combination of f-2-f and phone communication, it is a STSP/DP communication form.

To model the communication delay over the EVCS, I have developed a *voice-loop communication model* that includes the LM communication circuit as an additional agent with behavior. When a lunar surface agent makes an utterance (i.e. performs a communication transfer), this utterance is communicated to his partner and to the LM communication circuit agent. The LM communication circuit agent (LmComCircuit) communicates the utterance to the CapCom agent with a delay of one second⁴⁷. The result is that the CapCom agent will receive the communicated belief a second (a clock-tick) later, while the partner on the lunar surface will receive the belief instantaneous, i.e. at the moment of the communication.

Figure 6-36 shows how the voice-loop communication model lets the lunar surface agent AlBean communicate to both his partner PeteConrad and CapCom EdGibson that he is ready to start with the offload activity. First, the agent that speaks, AlBean in this case, has to have a belief about what needs to be spoken. Figure 6-36 shows this belief about the *speechAct* attribute being created in the *CommunicateReadyToOffload* workframe:

```
conclude((the speechAct of current = ReadyToOffloadAlsep));
```

The agent AlBean can now communicate this belief in the *Talk* communicate-activity:

```
Talk(vlcoms, start, OpenSEQBayDoorActivity, 10, 8);
```

This *Talk* activity transfers the belief at the start of the activity to all the agents bound to the *vlcoms* variable (PeteConrad and LmComCircuit in this case). The *Talk* activity is part of the *VoiceLoopCommunicator* group shown in Figure 6-37. Every member of the *VoiceLoopCommunicator* group, which the AlBean agent (as well as the PeteConrad and LmComCircuit agents) is a member of, inherits this *Talk* activity and will therefore be able to communicate its current belief about the *speechAct* attribute. Consequently, both the PeteConrad and the LmComCircuit agent receive AlBean’s belief about the *speechAct* attribute. Next, the LmComCircuit agent performs the *SendComToEarth* activity, which actually transfers the *speechAct* belief it just received from AlBean to the agents bound to the *vlagts* variable.

```
SendComToEarth(AlBean, vlagts);
```

The *vlagts* variable is bound to just the EdGibson agent (since he is the only agent with the *communicationType* equal to “MscVoiceLoop,” meaning he is the only agent listening to the voiceloop in MSC. The *SendComToEarth* activity, shown in Figure 6-36, has a duration of one second and transfers the *speechAct* belief at the end of the activity. Consequently, this describes the communication delay from the Moon to Earth. For longer delays one would simply increase the duration of *SendComToEarth* activity, making this a general model for voice-loop communication with communication delay.

⁴⁷ The Brahms clock grain-size cannot be set to 1.25 seconds, but has to be set to an integer number.

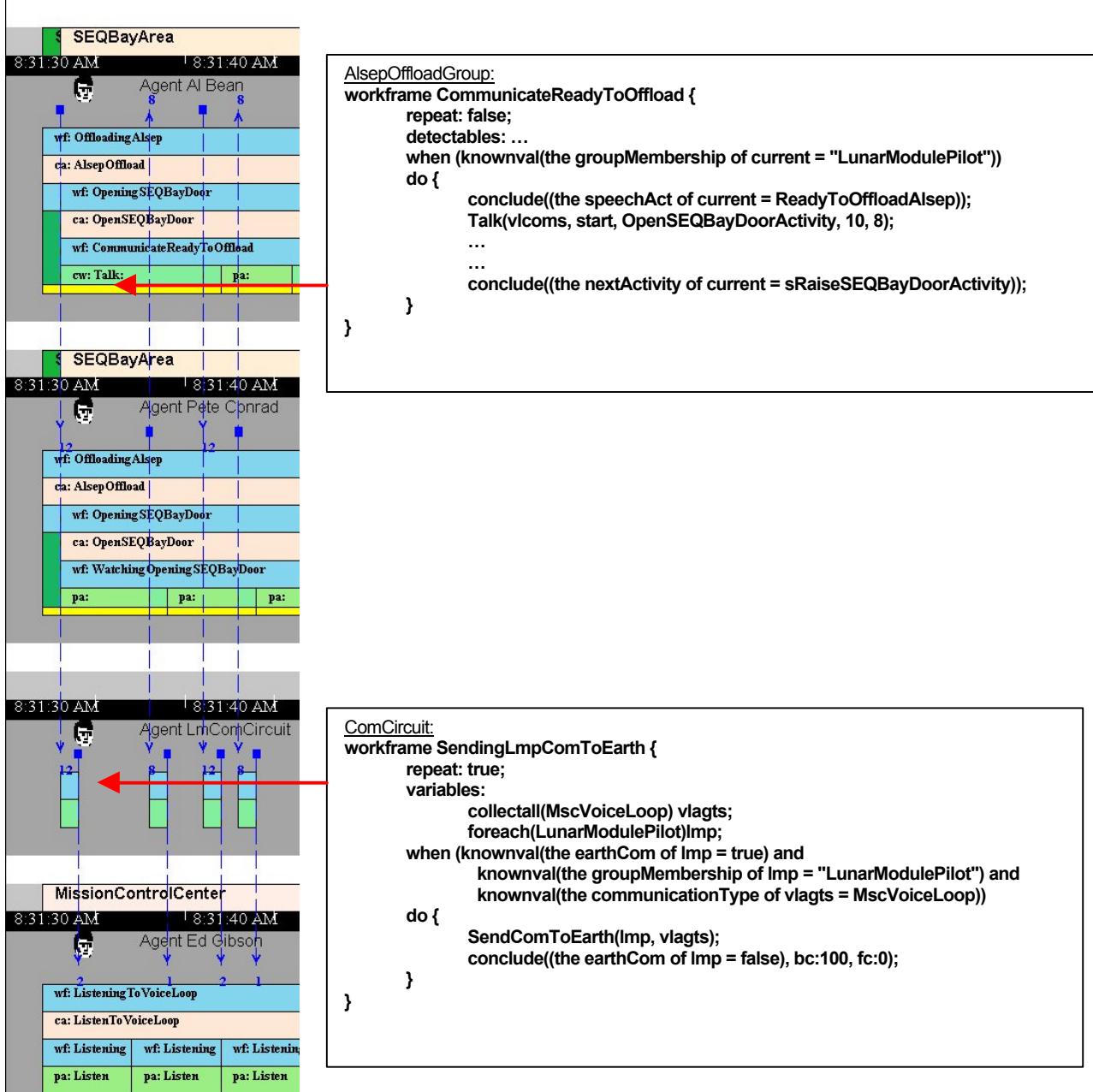


Figure 6-36. Voice-loop communication via LmComCircuit

6.7.3 The voice-loop library model

This voice-loop behavior is something that we want to re-use in other modeling efforts. I therefore developed this behavior as a library model that can be re-used over and over again. To do this we need to abstract the functionality of the voice-loop into separate functional groups. In this section, I describe the design of the voice-loop library model as it is shown working in Figure 6-36.

We can abstract the workings of the voice-loop system into two separate groups. First, there is a group of agents that can communicate over a voice-loop together. These agents are all members of the *VoiceLoopCommunicator* group. The *VoiceLoopCommunicator* group in turn is a member of the more abstract *Communicator* group. This group specifies those agents that can communicate in one way or another with each other, be it using a voice-loop, a telephone, e-mail, et cetera. There are three subgroups of the *VoiceLoopCommunicator* group, namely the *LmVoiceLoop*, the *CmVoiceLoop*, and the *MscVoiceLoop* group.

Then there is a group of agents that represent the communication circuits for each voice-loop. This is the *ComCircuit* group. This group has two member agents, namely one for the CM voice-loop, *CmComCircuit*, and one for the LM voice-loop, *LmComCircuit*. These two agents represent the communication circuits that create the delay of the communication between Earth and the Moon. The reason for modeling these as a group with agents, as opposed to a class with objects, is because we need these communication circuit agents to react to the communication transfers of *beliefs* from the “talking” agent. Although objects can receive and communicate beliefs, they cannot react to the beliefs they receive (objects only react to facts). All in all, it makes things easier from a modeling standpoint to model the communication circuits as agents, and since Brahms does not prescribe when to use agents versus objects this is a perfectly fine decision. The group hierarchy of the voice-loop model is presented in Figure 6-37.

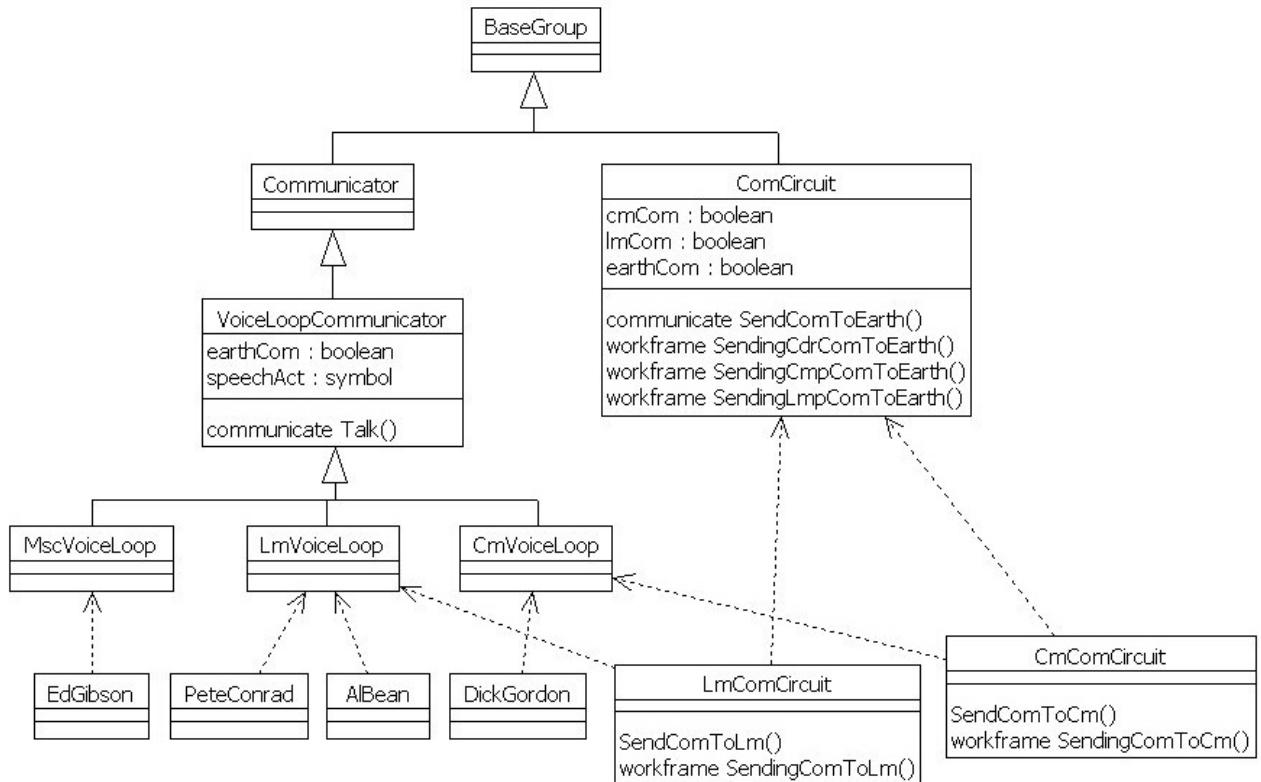


Figure 6-37. Voice-loop library model group hierarchy

6.8 OBJECT INTERACTION

We live in a world with objects. We look at them, touch them, and use them in our every day lives. When people work they use tools to accomplish what needs to be done. Interacting with objects in our environment is something so natural that we almost take it for granted when we consider how we do things. If we take a closer look at the work practice level, we need to include the way people interact with objects to describe what they do. On the moon the astronauts were together. However, they had artifacts with them, and objects that they needed to work on, and tools to use in their work. In this section, I describe how in Brahms we can model the interaction between objects and agents. I show the astronauts taking photographs and describe the model of the activities of the agent, and how the object it uses in these activities reacts and the way they both interact.

6.8.1 Lunar surface photography

Imagine taking a photograph. What do you do? What do you need? What does the camera do? Is it you or the camera that creates the photo? As I described before, all the tasks of the astronauts were planned and well trained. However, taking photographs was an acceptance to that rule. As it turns out, the Apollo

photographs were one of the most important scientific data returned to Earth. Some photographs were planned, but most were not, as is shown in the following example.

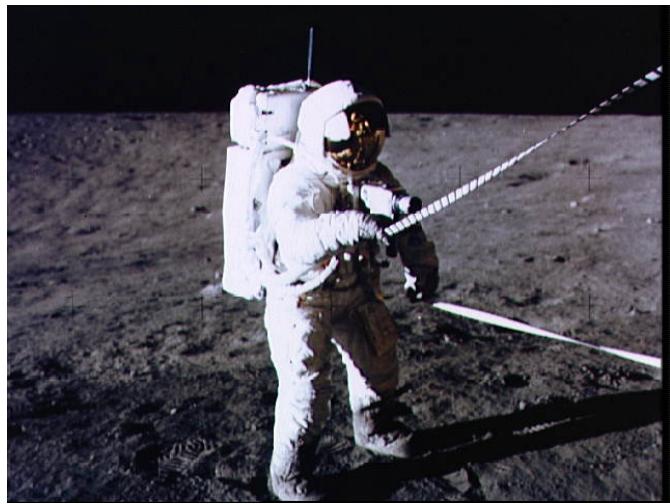


Figure 6-38. NASA picture AS12-47- 6913

Figure 6-38 shows a photograph that Al Bean took of CDR Pete Conrad, when he was lowering ALSEP Package-1 to the lunar surface. How did he do it? It is a subtle point, but it shows the collaboration between the two astronauts through the use of the photo camera.

116:32:48 Bean: Sure do. (Pause) Here it (probably the first package) comes.

116:32:53 Conrad: Coming right out.

116:32:54 Bean: And just about right. Riding right out on the boom, Houston. Sure looks pretty.

116:33:02 Gibson: (Making a mis-identification) Roger, Pete. We copy. (Long Pause)

116:33:36 Bean: (Wanting to take a picture) Look at me, Pete. (Pause) It's a good shot, babe. The LM and everything's reflecting in your visor. (Pause)

[Al's photos AS12-47- [6913](#) (***) and [6914](#) (****) show Pete using a tape to guide the first of the ALSEP packages out of the SEQ Bay. Photo [47-6915](#) (****) was probably taken late in the ALSEP off-load.]

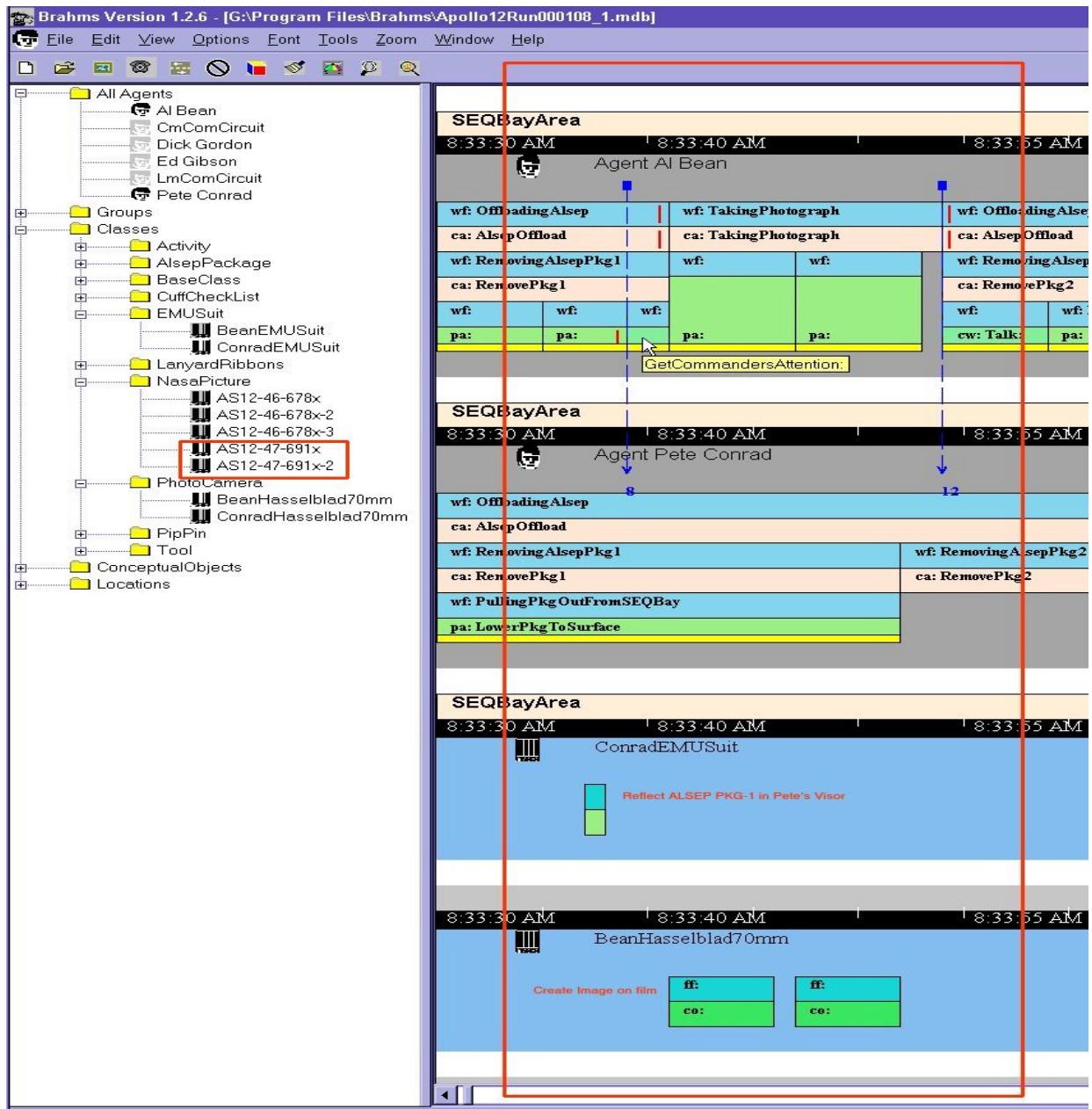


Figure 6-39. Al Bean taking two photographs of Pete Conrad

Figure 6-39 shows what happened during the simulation of the activity of taking this picture. First, it is important to realize how Al Bean decides to take a picture, and how this is modeled in Brahms. If we look at the utterance of Al Bean, we get some clues as to how this interaction happened. Al Bean says: "The LM and everything's reflecting in your visor." I interpret this as that the beautiful reflections in Pete Conrad's visor of his EMU suit made him want to take a picture (see the beautiful reflection in Figure 6-38). You can see in Figure 6-39 that the ConradEMUSuit object creates the fact that there is a reflection from its visor. At that moment, agent PeteConrad performs the activity LoweringPkgToSurface. Agent AlBean detects the reflecting visor fact, while watching agent PeteConrad. This detection interrupts agent AlBean's activity, and makes him perform the activity GetCommandersAttention. This activity represents the communication of Al Bean at time 116:33:36, where he says: "Look at me, Pete." This communication is shown in Figure 6-39 by the first arrow. After this activity, agent AlBean starts the *TakingPhotograph* workframe shown in Figure 6-39 and described in Figure 6-40.

First of all, taking a photograph is something that is not related specifically to the ALSEP Offload activity. Therefore, the *TakingPhotograph* workframe is not defined in the AlsepOffloadGroup group. Instead, it is part of all possible activities for members of the LunarSurfaceAstronaut group, because every lunar surface astronaut can take photographs at any moment. It is therefore that the agent AlBean interrupts the AlsepOffload activity to start the *TakingPhotograph* activity.

Group LunarSurfaceAstronaut

workframe TakingPhotograph

```
repeat true;
detectable DetectPhotoHasBeenTaken {
when (whenever)
    detect((the hasTakenPhoto of cam = true));
}
3} detectable ReleaseShutter {
when(whenever)
    detect((current PushesShutterReleaseButtonOf cam is false))
then complete;
}

when (knownval(the numberOfPhotosTaken of current <
    the numberOfPhotosToTake of current))
do {
    conclude((the hasTakenPhoto of cam = false), bc:100, fc:100);
    conclude((current PushesShutterReleaseButtonOf cam), bc: 100, fc:100);
    TakeThePicture(cam, 20, 0);
}

4} thoughtframe PhotoTaken
repeat: true;
when (knownval(the currentActivity of current = TakePhotographActivity) and
knownval(the hasTakenPhoto of cam = true))
do {
    conclude((the hasTakenPhoto of cam = false), bc:100, fc:0);
    conclude((the numberOfPhotosTaken of current = the numberOfPhotosTaken of current + 1),
bc:100, fc:0);
}
```

Class PhotoCamera

workframe OpenAndCloseShutter

```
repeat true;
variables:
    forone(EMUSuit) emu;
    forone(LunarSurfaceAstronaut) agt;
    unassigned forone(NasaPicture) photo;

when (knownval(emu contains current) and
knownval(agt PushesShutterReleaseButtonOf current))
do {
    CreateImageOnFilm(AS12-47-691x, photo);
    conclude((current hasPhoto photo), bc:100, fc:100);
    conclude((the hasTakenPhoto of current = true), bc:0, fc: 100);
    conclude((agt PushesShutterReleaseButtonOf current is false),
bc:0, fc:100);
}
```

1

4b

2a

2b

4a

}

Figure 6-40. Taking a photograph

After he has taken the photographs, he continues with the interrupted AlsepOffload activity (see the small vertical four lines in Figure 6-39, at the beginning and end of the *OffloadingAlsep* workframe of agent AlBean). Figure 6-40 describes the interaction between the agent and a *PhotoCamera* object in order for the agent to take a photograph (this numbered list refers to the numbers in Figure 6-40):

1. After agent AlBean starts the workframe *TakingPhotograph*, due to the fact that it believes that the number of photos to take is smaller than the number he has taken (see the precondition), it is simulated that the agent pushes the shutter release button on the camera. This is represented by the creation of the belief and fact

(AlBean PushesShutterReleaseButtonOf BeanHasselblad70mm)

The creation of this fact triggers the PhotoCamera object *BeanHasselblad70mm* to perform the *OpenAndCloseShutter* workframe, due to the fact that its preconditions are now satisfied.

2. Next, the camera object performs the *CreateImageOnFilm* create-object activity. On the left side of Figure 6-39 this *dynamically-created* object is shown as a *NasaPicture* object (AS12-47-691x). This actually represents the photo in Figure 6-38. After this activity, the camera object creates three facts; first, it creates the fact that the photo object has been created. Secondly, it creates the facts that it has taken a photo and that the agent AlBean stopped pushing the shutter release button on the camera. These last two facts are detected by agent AlBean, who is still performing the *TakingPhotograph* activity (arrows 2a and 2b in Figure 6-40). Arrow 2b shows that the agent stops the *TakeThePicture* activity by performing a *complete* action in the *ReleaseShutter* detectable, simulating that the agent has pushed the shutter button and has taken the picture.

3. Arrow 3, at the same time, shows that the agent fires the thoughtframe *PhotoTaken*. This thoughtframe increases the agent's belief about the number of photos it has taken.
4. Last, but not least, arrows 4a and 4b, make sure that the agent takes the right number of photographs. In the example in Figure 6-39, the agent takes two photographs, one after the other.

This example shows a general model for taking pictures. The only thing the agent needs to start out with is its camera contained on his EMU suit. Later on in the ALSEP Offload activity, during the offload of the second package, the PeteConrad agent actually takes three photos of AlBean while he is lowering object AlsepPkg2 to the ground, using the *ConradHasselblad70mm* PhotoCamera object (see Figure 6-41).

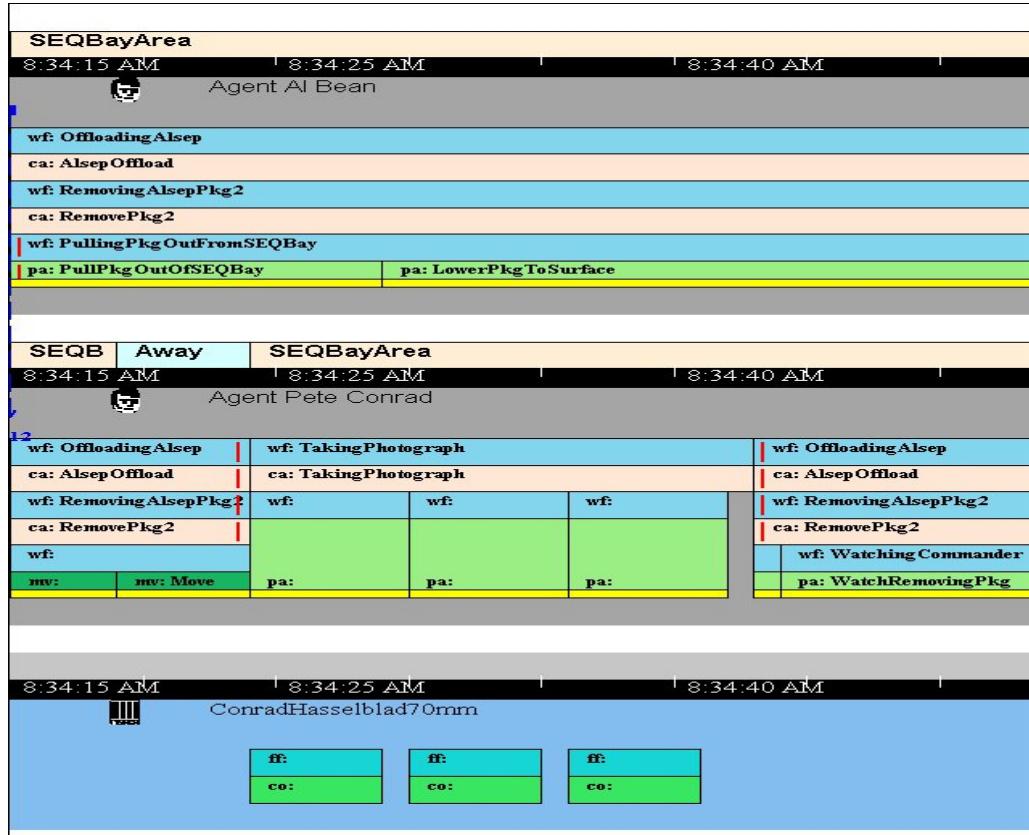


Figure 6-41. The PeteConrad agent taking photographs

Figure 6-42 shows the three actual photographs Pete Conrad took.



Figure 6-42. Photographs AS12-47-6783, 84, and 85 by Pete Conrad

A Brahms limitation

This example shows one of the limitations of a Brahms simulation. Although we can represent and simulate the taking of photographs, as well as the camera actually creating the *NasaPicture* objects, Brahms cannot show the sightlines of the camera, and that of the agents. The picture objects created do not include a

representation of what was captured on film (shown by the three photos in this example). The only way we could possibly represent this is to create beliefs that represent the scene being captured and “store” these beliefs in the NasaPicture object.

Not being able to model the line of sight of agents means that the model does not include whether the astronauts could actually see each other and/or the objects during their activities. Noticing other people and/or objects is often constrained by the line of sight. In Brahms, we can only model the detection of facts, based on the detectable being active and the existence of the fact in the world. However, in the real world the detection of certain facts depends on whether we can “observe the fact” through our field of vision, such as seeing someone in distress. Not being able to model the field of vision limits us in constraining the detection of facts based on the sightlines of the agent.

6.9 VERIFICATON AND VALIDATION

In this section, I describe the verification and validation (V & V) process I have followed to test the accuracy of the Apollo 12 ALSEP model. First, I will talk about V & V as a process and describe its elements, and some of its issues. Then, I will show in some detail the V & V steps I have followed and the results of this process in this experiment. Using this V & V process, I can say something about the accuracy of the model and my hypothesis about Brahms as a modeling and simulation language for *describing* a work practice.

To clarify the issues involved, I define the concepts *verification*, *validation*, and to be complete, *credibility* as follows:

- *Verification* is the process whereby the modeler asks if the model is performing as it was designed. In this step in the V & V process, the objective is to determine if the logic of the computer model correctly implements the assumptions made in the conceptual model.
- *Validation* is the process whereby the modeler asks how accurately the model is representing reality. That is, is it a good model of the intended work system?
- A *credible* model is one that the client accepts as being valid enough to use in making decisions. That is, is it a useful model for the task at hand? It should be noted that in this experiment we do not have a client that will make such a credibility judgment.

6.9.1 The purpose of verification and validation

An important part of modeling and simulation is the V & V of the model and the results of the simulation. Without a thorough V & V there is no ground in having any confidence in the model and the results of the simulation. Although it is important to realize that it is impossible to prove that a model is a general valid model (Robinson 1999). The reason for this is the fact that:

1. A model is only certified as valid with respect to its purpose. For instance, a model that has been created for the purpose of predicting the future state of a system might not be valid as a prescriptive model of the future system.
2. There are different interpretations of the real world possible. Depending on the worldview, or Weltanschauung, is a different interpretation of the real world and therefore, of the model and its validity (Checkland and Scholes 1990).
3. The data used to develop the model may be inaccurate. Even if that is not the case, it should be realized that the data used and the data generated by the simulation are but a small data sample. Therefore, they can only be seen as a probabilistic answer and not a definitive one.

The conclusion is that, although in theory a model is either valid or invalid, in practice it is not easy and often not possible to prove that a model is valid. Therefore, we have to think in terms of the confidence we can place in the model. The V & V of the model in this experiment is not one of demonstrating that the model is correct, but instead it is a process of *falsification*, i.e. demonstrating that the model is incorrect (Robinson

1999). In so doing, the purpose of V & V is to increase the confidence in the model, even though we might find inconsistencies and problems with the model according to the real-world data.

6.9.2 The verification and validation process

Many authors have described the process of a successful simulation (Law and Kelton 1991) (Kleindorf et al. 1998) (Banks et al. 1996) (Robinson 1994). All of them mention a series of processes that need to be followed. The high-level processes are shown in Figure 6-43, which is borrowed from (Robinson 1999). A simulation study first starts with understanding the *real world*, as well as the problem to be tackled. In this Brahms study, the real world is the Apollo 12 ALSEP Offload, with as the problem to be tackled, to test if we can describe the work practices of the lunar surface astronauts in a Brahms simulation. When the real world is sufficiently understood the modeling activity starts, and a *conceptual model* is described. For this study, I described the model as a qualitative model using a modeling approach called *World Modeling* (Sierhuis and Selvin 1996). After this, the model was coded into a computer model, in this case the Brahms language. When the model is complete, experiments are run to develop *solutions* to the real-world problem being handled. In this case, a greater *understanding* of the real world was obtained. In real-world projects it is hoped that the solutions found in the experiments can be implemented in the real world, or that the better understanding of the problem will lead to better decision making. In this experiment there has been no attempt to implement the model or change the real world based on the understanding, simply because this was not the purpose.

Even though there is a natural sequence in following these steps, it is obvious that the actual process is not strictly sequential, and that several iteration through the steps are necessary. This was also the case in this effort. First, there was no implementation phase based on the outcome of this study. Secondly, there were a number of cycles through the conceptual model, computer model and solution/understanding phase that were mostly driven by the validation and verification of the models with the real-world data. Even though this study did not end with an implementable solution in the real world, the process as depicted in Figure 6-43 still holds.

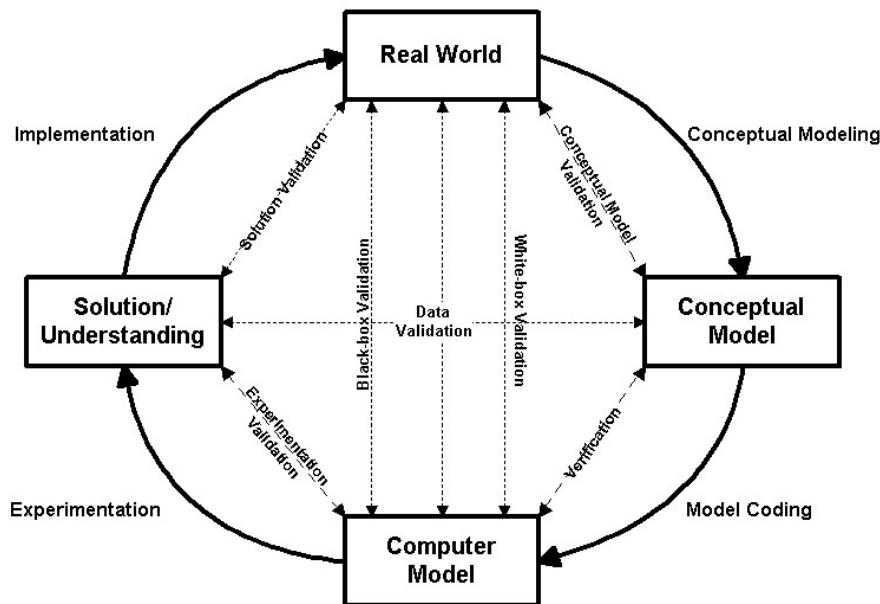


Figure 6-43. Simulation model verification and validation in the modeling process (borrowed from (Robinson 1999))

In the next sections, I will describe the activities of the three phases, conceptual model, computer model, and solution/understanding and the validation and verification methods used in each of these phases.

6.9.3 Data validation

As is shown in Figure 6-43, data validation is important at every step of the simulation process, because at each step in the process data is used. The data I used are all original NASA records of the actual Apollo missions. Table 6-4 lists all data sources that have been used in this case study. Since the Apollo missions are part of world history the facts and data are well known to the world and are therefore undisputed. By using the original lunar videos, as well as the transcripts of the original conversations of the astronauts, and the original photographs, mission reports and press releases, the validity of the data is very high. It can thus be said that, if the simulation data is validated against the original mission data, and it can be shown that the outcome is correct in relation to this data, the validity of the simulation model is high.

Table 6-4. Data sources used during experiment

Data Source	Data Type
Apollo Lunar Surface Journal	Transcriptions of actual astronaut voice loop recordings + mission photographs.
Apollo 14, 15 & 16 Video Tapes	Video Recordings of the actual Apollo missions from NASA.
Apollo 12 ,14, 15 & 16 Press Kits	Copies of the actual Apollo Press Kits from the Apollo missions, published by NASA.

6.9.4 Conceptual model validation

I started the modeling effort by creating a conceptual model of the Apollo 12 ALSEP Offload. The method used is called *Compendium*, and is described in (Sierhuis and Selvin 1996) (Selvin and Sierhuis 1999b) (Selvin and Sierhuis 1999a) (Selvin et al. 2001). The discussion of this method falls outside of this thesis. Figure 6-44 shows the *Raise SEQ Bay Door* activity described in the conceptual model. To model this activity, I used the voice loop transcription data from the Apollo LSJ (see Figure 6-45), as well as the Apollo video of the Apollo 14⁴⁸ mission. The voice loop data is modeled as the *communication* attribute in the model. By reading and listening to the communication, matching this to the mission plan, and validating this with the video, I was able to analyze who performed this activity (see Figure 6-45), and where in the voice loop transcription the astronaut was starting and ending this activity. The approach I used was to identify the activity duration based on communication sequences. The astronaut was performing the activity during the first utterance and the last utterance of a communication sequence. By using the timestamps in the Apollo LSJ, I calculated the total time of the activity (see Figure 6-47). I also represented where the agents were located while performing this activity, as well as what objects (artifacts) the astronaut was touching or using during this activity.

By analyzing the transcription of the voice loop data this way, I have represented and validated each activity of the agents. After this process was completed, the conceptual model had to be coded in the Brahms language.

⁴⁸ Due to a unfortunate problem with the camera, there is no video tape of the Apollo 12 ALSEP Offload.

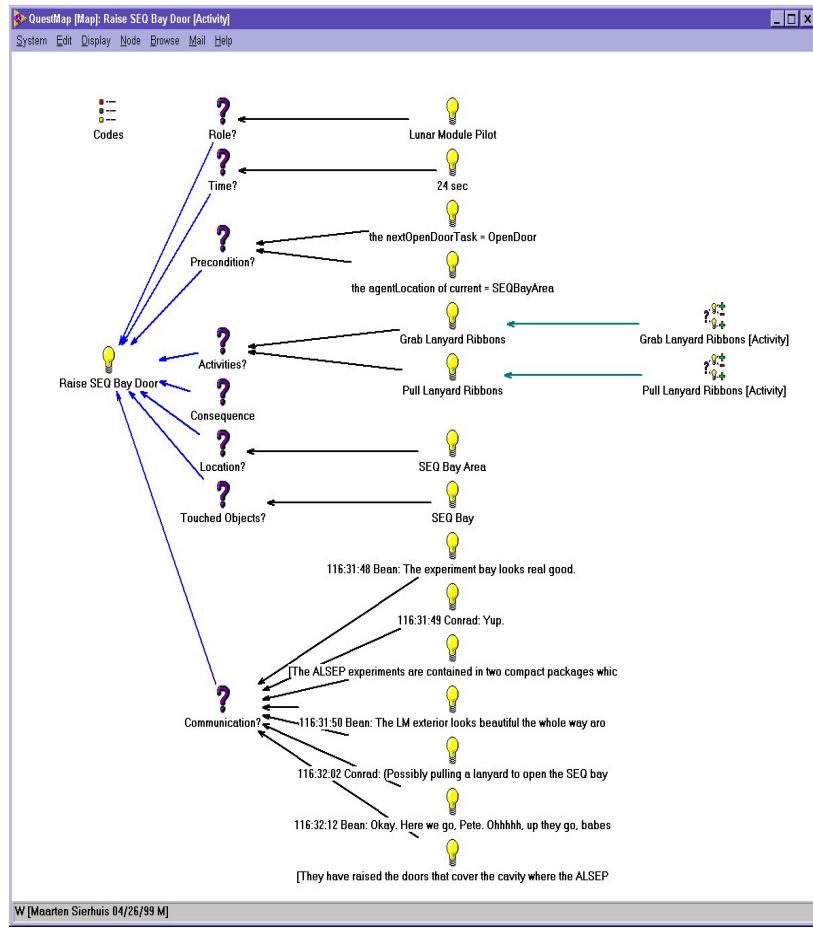


Figure 6-44. The conceptual model

The purpose of the conceptual model validation is to determine that the scope and level of detail of the proposed model is sufficient, and that all assumptions are correct (Robinson 1999). To describe this validation, let me take a step back and restate the problem I addressed in this study. The problem in this study was that of *showing that the Brahms modeling and simulation language is powerful enough to describe the work practice of the Apollo 12 lunar surface astronauts during the ALSEP Offload activity*. The level of model detail that is needed to test this hypothesis is given by the definition of what to include in a model of work practice (see chapter 3.2).

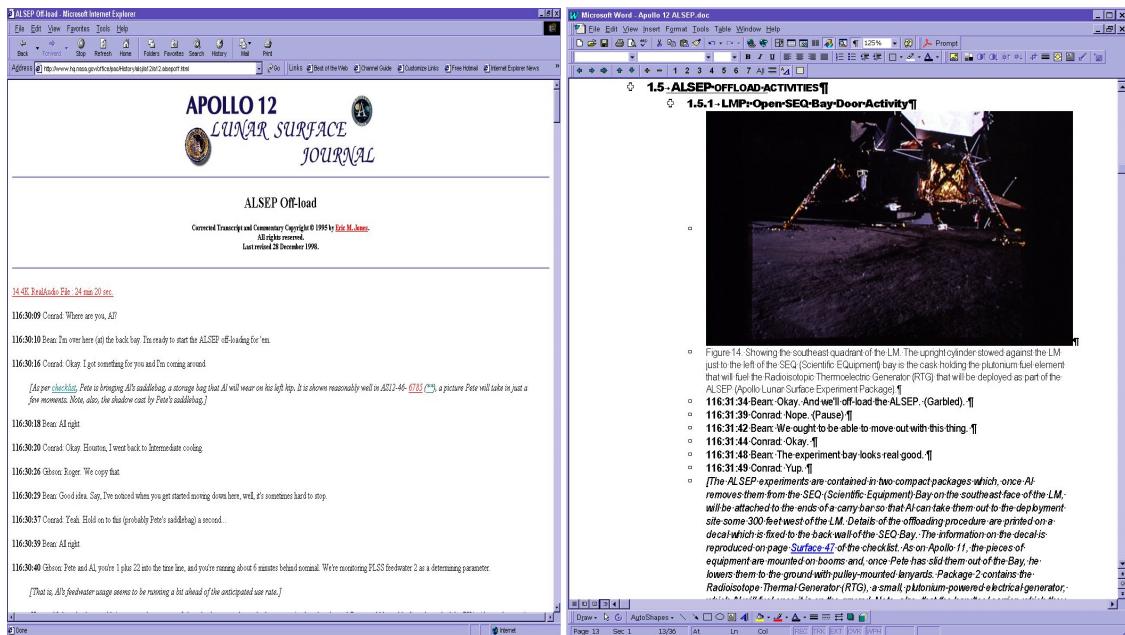


Figure 6-45. Voice loop transcription data from the Apollo LSJFigure 6-46. Voice loop transcription matched to activities

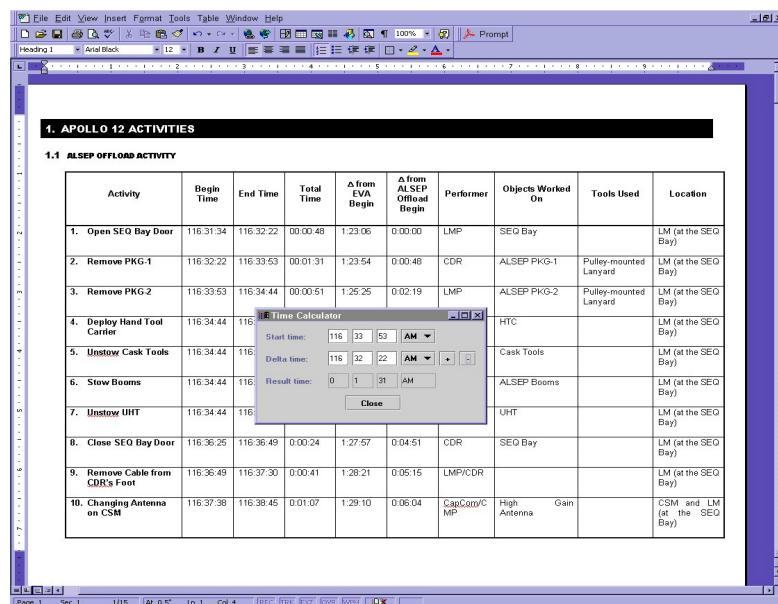


Figure 6-47. Voice loop activity time analysis

If we take as a given the aspects of work practice from chapter 3.2, then we can validate that these aspects are indeed included in the model. Therefore, the validation method I used for the conceptual model was to analyze the important aspects of modeling work practice, as described in the theory, and to make sure that the conceptual model included all of them. Table 6-5 lists the aspects that were to be included in the model, as well as how these aspects are made operational in the coded model:

Table 6-5. Aspects of modeling work practice

Aspect of Work Practice	Model
Communities of Practice	This aspect is incorporated in the model by modeling the roles and functional groups of the agents as <i>groups</i> with behavior in the model. People who belong to certain CoP are represented as agents being members of the groups, inheriting the common behavior of the group members.
Activities	The behavior of all agents and artifacts is described in terms of primitive activities taking time, and composite activities decomposed into lower-level activities, and the lowest-level into primitive activities.
Collaboration	Collaboration is an emergent aspect of the model that is shown in the output data of the simulation. By describing the activities of agents, and the interaction and constraints that make each agent perform an activity based on activities of other agents, shows that agents are collaborating together.
Communication	The model includes all the speech acts from the real voice data. Activities are sometimes dependent on whether these speech acts are performed and received.
Real world artifacts	For each activity the artifacts that are used or touched in the activity are represented in the model. Relationships between activities and artifacts are represented.
Geography and Movement	For each activity, the location where the activity is performed is represented. Agents move from location to location, and performance of an activity is sometimes dependent on being in the location or noticing other agents and/or artifacts in a location.

6.9.5 Computer model verification

The next phase in the modeling process is the design and implementation of the Brahms model source code. In this phase, the modeler needs to translate the activities, groups, agents, classes and objects represented in the conceptual model into the Brahms language. To do this, the modeler needs to be proficient in the Brahms language, and specifically in the multiagent and activity programming concepts in Brahms. For first time Brahms modelers this is a painstaking process, and is similar to the compile-debug cycle in traditional programming languages, such as C++ or Java.

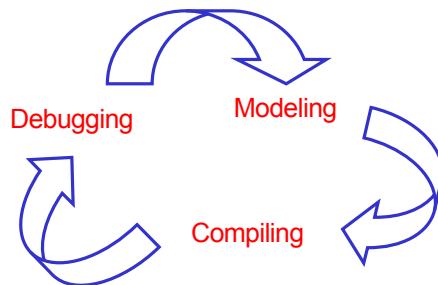


Figure 6-48. Brahms compile-debug cycle

Figure 6-48 shows the modeling cycle, which first continues until the complete model can be compiled without syntax errors by the Brahms compiler. However, verifying the model is more than getting the Brahms compiler to compile the model without syntax errors. Although this is of course a first and important step in the process, the most important step is to compare the “functioning” of the model with the conceptual model. The model validation and verification steps are driving the Brahms model development process, shown in Figure 6-49

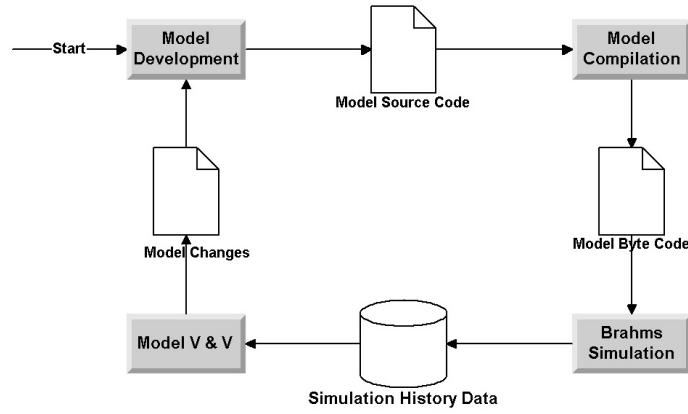


Figure 6-49. Brahms model development cycle

The functioning of the model is visually verified using the *AgentViewer* application. The *AgentViewer* is a separate Brahms application that uses the simulation history data to display a 2-dimensional graphical timeline view of the activities of agents and objects. The timeline figures in this and other subsequent chapters are all screenshots from selected agents and objects in the *AgentViewer*. Using the *AgentViewer* application the modeler can investigate the simulation run.

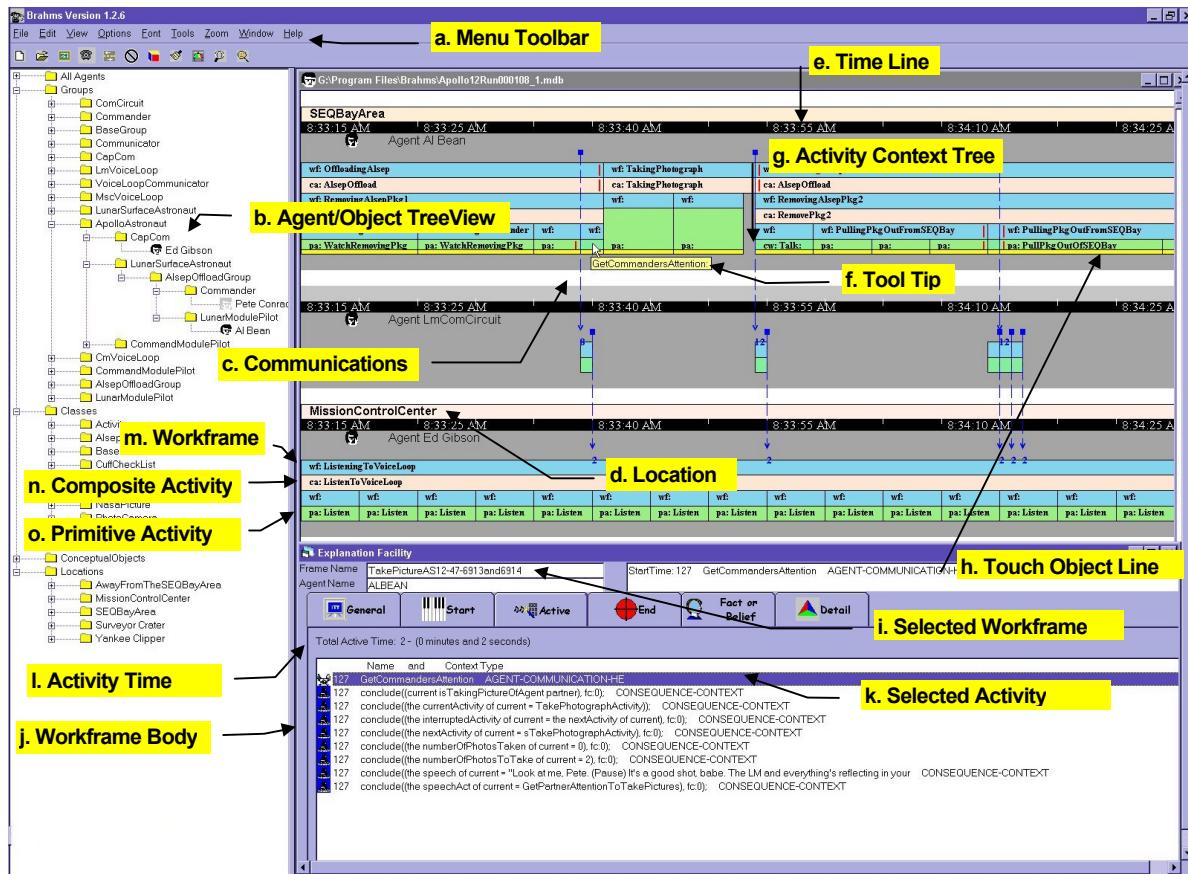


Figure 6-50. AgentViewer application

Figure 6-50 shows the *AgentViewer*. Using this application the end-user can select which agents and objects to view in the time-line view, and investigate the exact behavior of those agents and objects during the simulation (see a-l explanations in Figure 6-50):

- Using the menu-bar, the end-user can parse the simulation history data into a history database, and open a history database for viewing.

- b. When the database is opened all the agents and objects are loaded into the tree view. Using the tree view, the end-user can select which agents and/or objects (s)he wants to view in the time-line view.
- c. By selecting to view the agent/object communication, the (blue) arrows show all the communication activities, and the direction of the communication (sender and receivers). The communicated beliefs are also accessible by clicking on the square at the top of the sender side of the communication arrow.
- d. For each agent/object the "current" location is shown. When the agent/object moves to a new location, it is shown as a change in the location name and color.
- e. The time-line can show the time in different time-intervals, therewith zooming in and out.
- f. The tool-tip pops up when the mouse is moved over "hot spots". The hot spots are those areas where more information is available than can be shown on the screen. By moving the mouse over those areas the hidden information pops up in a tool-tip, such as the name of a workframe or activity.
- g. The Activity-Context Tree is the central piece of the agent/object time-line. It shows the workframe and activities hierarchy of the agent or object.
- h. The touch-object line is a (yellow) line that is shown when the agent/object is using certain objects in its activity. "Touch objects" are used to calculate the time those objects are used in activities.
- i. The explanation facility view is used to display more detailed information about the execution of workframes. By clicking on any workframe (light blue in color), an explanation facility window is opened for the workframe at hand.
- j. By selecting the "Active" tab in the explanation facility view, the executed statements in the workframe body are shown.
- k. You can select the statements in the workframe body to get more info.
- l. When you select a statement in the body of the workframe, the total time the activity was active is shown. Using the other tabs in this view, you can find out the exact time the workframe became available, as well as the exact time it became active and ended.
- m. Workframes are situated-action rules that execute activities. The top of a Activity-Context tree is always a workframe. You can recognize a workframe by the "wf:" symbol, followed by the name of the workframe. When the zoom-level is too high to contain the name of the workframe it is left out of the display. Using the tool-tip the user can find out the name.
- n. Composite Activities are executed by workframes, and contain lower-level workframes. You can recognize Composite Activities by the "ca:" symbol followed by the name of the activity. When the zoom-level is too high to contain the name of the activity it is left out of the display. Using the tool-tip the user can find out the name.
- o. Primitive Activities are executed by workframes, and are always at the bottom of the Activity-Context Hierarchy. You can recognize Primitive Activities by the following symbols, depending on the type of primitive activity: "pa:" (for a primitive activity), "mv:" (for a move activity), "cw" (for a communicate activity), "co:" (for a create object activity), followed by the name of the activity. When the zoom-level is too high to contain the name of the activity it is left out of the display. Using the tool-tip the user can find out the name.

Using this AgentViewer I have visually inspected the simultaneous behavior of the agents and objects, and compared the expected behavior from the conceptual model with the actual behavior during the simulation.

6.9.6 Experimentation validation

Comparing the model output to data from the real system is the most objective and scientific method of validation. Of course, this type of validation can only be performed if there is a real system, and real-world data that correspond to the simulation parameters. In this descriptive modeling experiment there was a real system back in the Apollo days. That system does not exist anymore, but what is most important is the fact that there is historical data available to validate our model.

I describe two types of quantitative data validation of the simulation output data of the Apollo 12 model, based on the historical data from the Apollo missions:

1. Validate the simulated activity times and duration with the activity times and duration derived from the timestamps in the Apollo 12 communication transcript from the Lunar Surface Journal (Jones 1997).
2. Validate the simulated voice loop communication with the voice loop recordings from the actual mission, which are transcribed in the same Apollo 12 communication transcript (Jones 1997).

6.9.6.1 White-box versus black-box validation

We consider two types of real-world data validation, *white-box* and *black-box* validation. The model verification described in section 6.9.5 is considered a white-box validation. Validating the simulated activity times with the timing of the activities based on the transcript of the voice loop communication is a white-box validation. The second validation, that of the actual voice loop data, is a *black-box validation*.

White-box validation is a *micro validation of the content* of the model. In a white-box validation we try to validate the model by investigating the model content in detail. The purpose of this type of validation is to ensure that the content of the model is true to the real world. The use of a graphical visualization and spreadsheet tools are very appropriate in this type of validation.

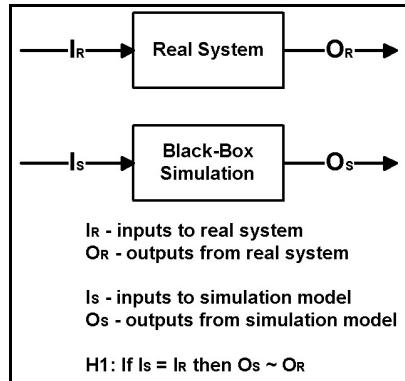


Figure 6-51. Black-box validation: comparison with the real system (from (Robinson 1994))

In a black-box validation we are not looking inside the model, but we are *validating the overall behavior of the model* with the output of prespecified real-world data. In this type of validation we need to validate that when we specify input data to the simulation model that is similar to that of the real system, the output data from the simulation should be relatively similar to that of the real system as well. This is a validation of the alternative hypothesis H1 (Figure 6-51).

6.9.6.2 Validate activity times

To validate the timing and duration parameters of the simulation model, we measure the activity times of the individual activities performed by each astronaut. Initially I had identified the activities of the astronauts based on the Apollo 12 communication transcripts (see Figure 6-45). Based on this and the fact that each communication utterance in the transcript is timestamped with the actual mission clock at MSC, I was able to calculate the ground-estimated time (GET) start and end times of the activity. From this the total activity time could be calculated (Figure 6-47). Here I am only showing the validation of the first three high-level

activities (Table 6-6). This is only in the interest of space, and I hope that with this example the reader is satisfied and can infer that the same holds for the other activities.

Table 6-6. Calculated activity times based on real-world data

Activity	Start GET	End GET	Total Time	? from ALSEP Offload Begin	Performer
Open SEQ Bay Door	116:31:34	116:32:22	00:00:48	0:00:00	LMP
Remove PKG-1	116:32:22	116:33:53	00:01:31	0:00:48	CDR
Remove PKG-2	116:33:53	116:34:44	00:00:51	0:02:19	LMP
		Total	0:03:10 (190 sec)		

An issue is the fact that the start and end times of the activities were chosen based on a thorough reading of the Lunar Surface Journal transcriptions, books and reports on the Apollo 12 mission, as well as the videos of the ALSEP Offload activities in subsequent missions. The choices I made are subjective to my own interpretation, as well as that of Erik Jones, the editor and creator of the Apollo Lunar Surface Journal (Jones 1997). It might well be possible that someone else would make a different interpretation of the timing based on the same data. Although this may be the case, it should not have much influence on the outcome of this study, since the goal of this validation is in context of the objective of the experiment. As mentioned before, the objective is to show that with Brahms we can describe the work practice of a real human activity system. We can still make a judgment on this, regardless of the fact that the subjectivity of the modeler is unavoidable in a modeling activity.

Table 6-7. Activity times for LMP AI Bean from simulation history database

DoneByID	DisplayText	Start ⁴⁹	Start SET ⁵⁰	End	End SET	TotalTime	Status
ALBEAN	OpenSEQBayDoor	1	8:31:34	49	8:32:22	48	COMPLETED
ALBEAN	RemovePkg1	49	8:32:22	53	8:32:26	4	INTERRUPTED
ALBEAN	ChangeEMUSuitCooling	53	8:32:26	58	8:32:31	5	COMPLETED
ALBEAN	RemovePkg1	58	8:32:31	124	8:33:37	66	INTERRUPTED
ALBEAN	TakingPhotograph	124	8:33:37	137	8:33:50	13	COMPLETED
ALBEAN	RemovePkg1	137	8:33:50	140	8:33:53	3	COMPLETED
ALBEAN	RemovePkg2	140	8:33:53	191	8:34:44	51	COMPLETED
					Total	190	

⁴⁹ The times in the Start, End, and TotalTime columns are in seconds.

⁵⁰ The times in the Start Simulation Elapsed Time (SET) and End SET are in the format h:mm:ss.

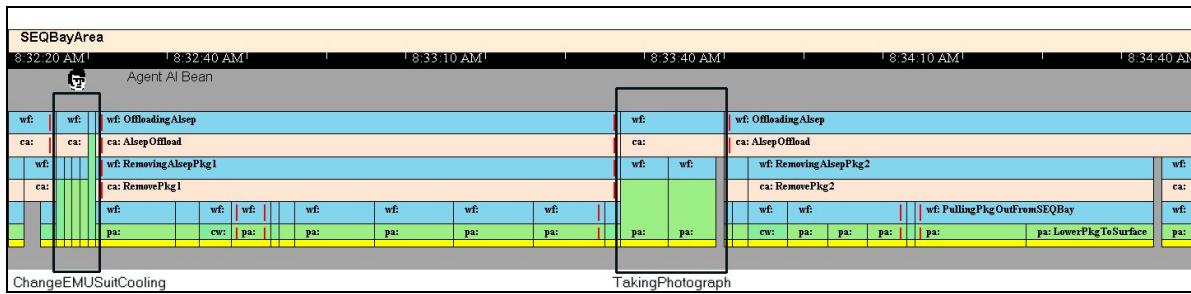


Figure 6-52. AI Bean's RemovePkg1 and RemovePkg2 activities

The activity times from Table 6-7 are the result of the emergent performance of lower-level activities of AI Bean, as can be seen in Figure 6-52. The timing of the composite activities from Table 6-7 are based on the cumulative times of the lower-level primitive activities performed as part of these composite activity.

Table 6-7 shows that AI Bean interrupts the RemovePkg activities twice to perform activities that are not necessarily part of the high-level AlsepOffload composite activity. The ChangeEMUSuitCooling activity is an activity that can be performed at the moment the astronaut feels too warm or too cold. Performing this activity is an interruption of the AlsepOffload activity and its underlying subactivities that are being performed at that moment. Consequently, the current RemovePkg1 activity will continue *after* the ChangeEMUSuitCooling activity is finished. You can see this represented in both Table 6-7 and Figure 6-52. Table 6-8 and Figure 6-53 show the activities for Pete Conrad. Pete Conrad does not perform the ChangeEMUSuitCooling activity (he is too busy offloading the package!), but he is interrupting his RemoveAlsepPkg2 activity taking three photographs while AI Bean is lowering the second ALSEP package.

Table 6-8. Activity times for CDR Pete Conrad from simulation history database

DoneByID	DisplayText	Start	Start SET	End	End SET	TotalTime	Status
PETECONRAD	OpenSEQBayDoor	1	8:31:34	50	8:32:23	49	COMPLETED
PETECONRAD	RemovePkg1	50	8:32:23	140	8:33:53	90	COMPLETED
PETECONRAD	RemovePkg2	140	8:33:53	170	8:34:23	30	INTERRUPTED
PETECONRAD	TakingPhotograph	170	8:34:23	189	8:34:42	19	COMPLETED
PETECONRAD	RemovePkg2	189	8:34:42	191	8:34:44	2	COMPLETED
				Total		190	

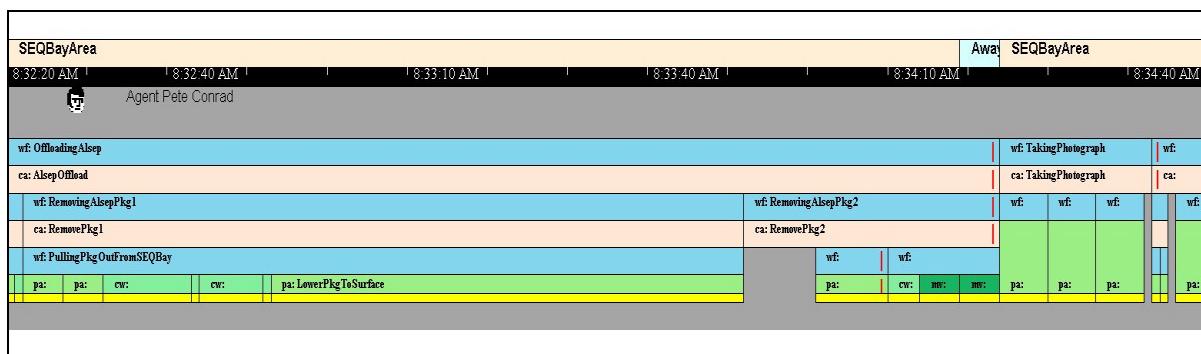


Figure 6-53. Pete Conrad's RemovePkg1 and RemovePkg2 activities

From Table 6-6, Table 6-7 and Table 6-8 it can be seen that the timing for both the AIBean and PeteConrad agents are similar as the timing data from the Apollo LSJ. With this verification the white-box validation of the model is completed, and we can state that the computer model content (i.e. the Brahms model) is a valid implementation of the conceptual model, which in turn is based on the Apollo 12 data.

6.9.6.3 Subtracting communication delay

The times, as shown in Table 6-6, present the following small but significant validation issue. The times based on the actual voice transcriptions are the mission times as they were measured by mission control. Since the activities are those of the lunar surface astronauts being performed on the moon, the actual times that the astronauts spoke the words transcribed in the Apollo LSJ documents would have had to be one and a quarter (1.25) second earlier. This is because there was a one and a quarter second delay between earth and moon communications (see section 6.7.1 about communication delay).

The start and end GET times are not the actual start and end times of the activities performed on the moon. Although the total activity time stays the same, to be correct, the activities of the astronauts need to start one and a quarter second earlier. To get to this point, I followed a two-step validation process:

1. Validation of the simulated activity times by making the times match up exactly with those measured on earth, shown in Table 6-6. The result of this validation was shown in Table 6-7 and Table 6-8.
2. After the simulation model is validated according to (1), we transpose the simulation times to include the earth/moon delay. Because only the start time is different, we can simply start the simulation clock earlier. This results in the activities of the astronauts starting at the actual start time, and thus in communication utterances arriving at mission control at the time measured by the GET clock. The result for agent AlBean is shown in Table 6-9

There is an issue with the capability of the simulation engine only being able to have an integer clock-grain-size. This means that we cannot simulate the one and a quarter second delay. The closest we can get is to have a clock-grain-size of one (1) second. Therefore, the delay I have been able to introduce in the simulation is one second. The numbers that have consequently been generated are still off by a quarter (0.25) of a second. However, this is a consistent error rate, and thus could easily be subtracted from the generated numbers.

Table 6-9. Activity times for LMP Al Bean including communication delay

DoneByID	DisplayText	Start	Start SET	End	End SET	TotalTime	Status
ALBEAN	OpenSEQBayDoor	1	8:31:33	49	8:32:21	48	COMPLETED
ALBEAN	RemovePkg1	49	8:32:21	53	8:32:25	4	INTERRUPTED
ALBEAN	ChangeEMUSuitCooling	53	8:32:25	58	8:32:30	5	COMPLETED
ALBEAN	RemovePkg1	58	8:32:30	124	8:33:36	66	INTERRUPTED
ALBEAN	TakingPhotograph	124	8:33:36	137	8:33:49	13	COMPLETED
ALBEAN	RemovePkg1	137	8:33:49	140	8:33:52	3	COMPLETED
ALBEAN	RemovePkg2	140	8:33:52	191	8:34:43	51	COMPLETED
					Total	190	

6.9.6.4 Validate output with real-world data

Next is the black-box validation. The purpose is to validate that the simulation can recreate the communication utterances by the astronauts exactly and at the same ground-elapsed time as the data from the Apollo LSJ. I show this validation of the model for the OpenSEQBayDoor activity as described in section 6.5.3. For ease of the reader, I repeat here the activity/communication table for the OpenSEQBayDoor activity.

Table 6-10. OpenSEQBayDoor activity with communication

LMP		CDR	
Communicate Ready To Offload		Watching Opening SEQ Bay Door	
activity	Communication	communication	activity
<i>Talk</i>	116:31:34 Bean: Okay. And we'll off-load the ALSEP. (Garbled).		<i>Watch Opening SEQ Bay Door</i>
		116:31:39 Conrad: Nope. (Pause)	<i>Talk</i>
<i>Talk</i>	116:31:42 Bean: We ought to be able to move out with this thing.		<i>Watch Opening SEQ Bay Door</i>
<i>Inspect SEQ Bay</i>		116:31:44 Conrad: Okay.	<i>Talk</i>
<i>Talk</i>	116:31:48 Bean: The experiment bay looks real good.		<i>Watch Opening SEQ Bay Door</i>
		116:31:49 Conrad: Yup.	<i>Talk</i>
Raising SEQ Bay Door			<i>Watch Opening SEQ Bay Door</i>
activity	Communication		<i>Watch Opening SEQ Bay Door</i>
<i>Grab Lanyard Ribbons</i>	116:31:50 Bean: The LM exterior looks beautiful the whole way around. Real good shape. Not a lot that doesn't look the way it did the day we launched it.		<i>Watch Opening SEQ Bay Door</i>
<i>Walk Back To Pull Ribbons Tight</i>			<i>Watch Opening SEQ Bay Door</i>
<i>Pull Lanyard Ribbons</i>		116:32:02 Conrad: (Possibly pulling a lanyard to open the SEQ bay doors) Light one. (Pause)	<i>Talk</i>
<i>Talk</i>	116:32:12 Bean: Okay. Here we go, Pete. Ohhhhh, up they go, babes. One ALSEP. (Pause)		<i>Watch Opening SEQ Bay Door</i>
<i>Pull Lanyard Ribbons</i>		116:32:22 Conrad: There it is.	<i>Talk</i>

Table 6-10 shows the subactivities of the OpenSEQBayDoor activity. The objective of this black-box validation is to have the simulation generate the exact communication utterance for each agent, at the exact time specified in. Of course, the same issue exists regarding the measurement of the time in GET and the communication delay to/from the moon. It should again be realized that the times in Table 6-10 are times measured by MSC, and are therefore the times that the CapCom agent heard the utterance over the voice loop, thus one and a quarter second later than the time the lunar surface astronauts uttered the words.

After having validated the activity times from the previous section, I changed the model to include the communication utterances specifically for this validation. To generate the exact utterance, the agent creates the utterance as a belief right before it communicates the belief in the Talk activity.

```

workframe CommunicateReadyToOffload {
    /detectable deleted/
    when (knownval(the groupMembership of current = "LunarModulePilot"))
        do {
            //communication from transcription
            conclude((the speech of current = "Okay. And we'll off-load the ALSEP. (Garbled)."), bc:100, fc:0);
            conclude((the speechAct of current = ReadyToOffloadAlsep), bc:100, fc:0);
            Talk(vlcoms, start, OpenSEQBayDoorActivity, 10, 8);
        //end validation

        //communication from transcription
        conclude((the speech of current = "We ought to be able to move out with this thing."), bc:100, fc:0);
        Talk(vlcoms, start, OpenSEQBayDoorActivity, 0, 1);
        //end validation

        InspectSeqBay(0, 4);

        //communication from transcription
        conclude((the speech of current = "The experiment bay looks real good."), bc:100, fc:0);
        conclude((the speechAct of current = the exteriorAppearance of SEQBay), bc:100, fc:0);
        Talk(vlcoms, end, OpenSEQBayDoorActivity, 0, 1);
    //end validation

    conclude((the nextActivity of current = sRaiseSEQBayDoorActivity), bc: 100, fc: 0);
}
}

```

Figure 6-54. Workframe with communication utterance from Apollo LSJ

Figure 6-54 shows the rewritten *CommunicateReadyToOffload* activity including the communication utterances. For each utterance there is a belief created for the *speech* attribute for the LMP agent (i.e. AlBean). This speech attribute signifies the actual speech-utterance of the agent.

```

communicate Talk(Communicator agt, symbol whn, Activity act, int pri, int maxd) {
    priority: pri;
    max_duration: maxd;
    resources: act;
    with: agt;
    about: send(the speech of current = value),
            send(the speechAct of current = value);
    when: whn;
}

```

Figure 6-55. Talk activity to validate communication

Next, the *Talk* communicate-activity actually communicates the *speech* belief to appropriate agents (Figure 6-55). Running the simulation again with this added communication, first and foremost, does not change the behavior of the agents. The end-result of the simulation is the same, as can be seen in Figure 6-56, but Table 6-11 shows that the simulation generates the actual voice loop communication transcription consistent with that in the Apollo LSJ.

The data in Table 6-11 is compiled from the history database. The data shows the communication of the *speech* attribute from the *LmComCircuit* agent. This is the agent that simulates the communication delay from/to the moon (see section 6.7, explaining the voice loop model). Therefore, the time this agent relays the communication should be equal to the GET from the Apollo LSJ. This is shown in the last two columns. The second to last column shows the simulated elapsed time (SET), which is the time from the simulation. The last column shows the ground-elapsed time (GET) as it is recorded at MSC, and is shown in the Apollo LSJ.

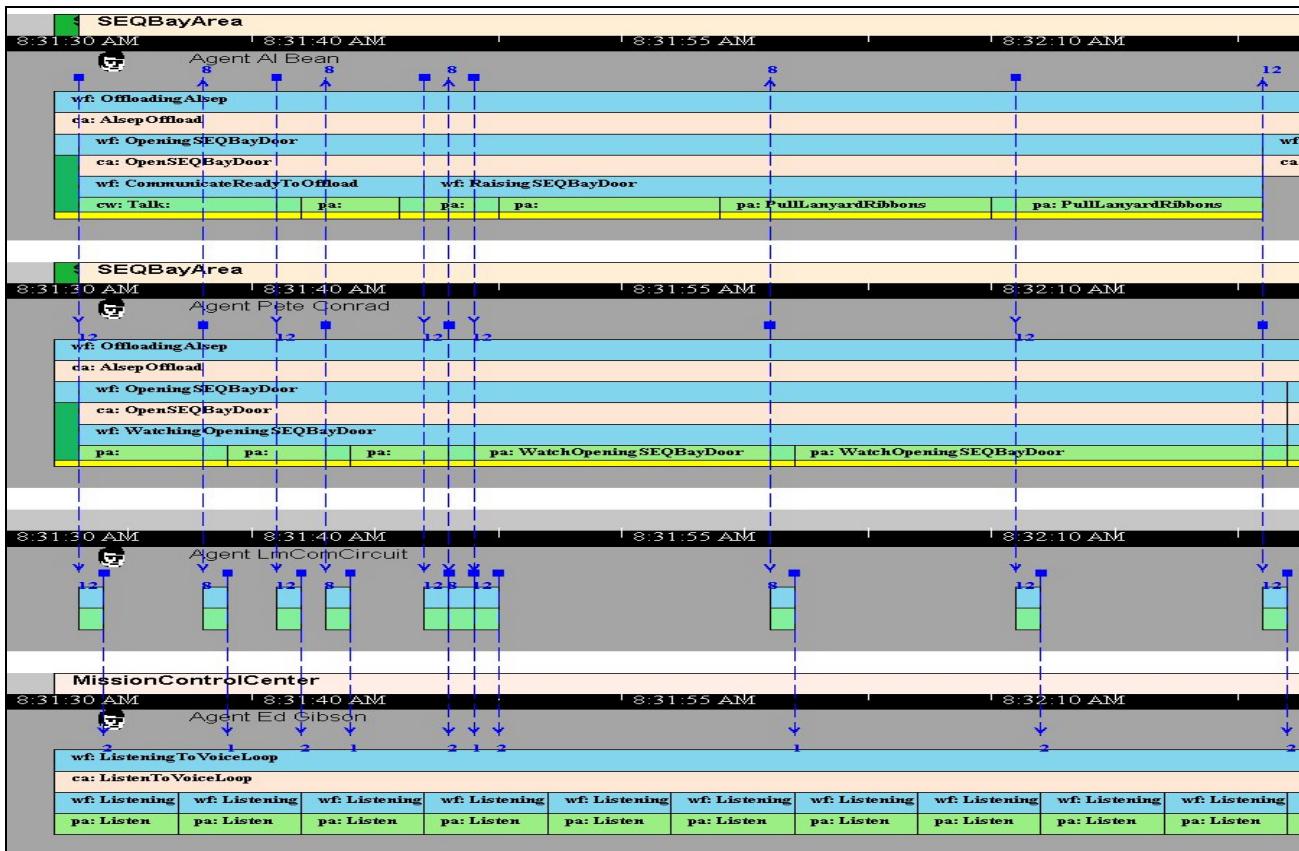


Figure 6-56. Voice loop communication for OpenSEQBayDoor activity

Table 6-11. Agent speech communication validation

Agent	FrameName	Attribute Name	Value (from Apollo 12 LSJ)	Start Time	Speech at SET	Speech at GET (from Apollo 12 LSJ)
LMCOMCIRCUIT	SendingAlBean ComToEarth	speech	"Okay. And we'll off-load the ALSEP. (Garbled)."	0:00:02	8:31:34	116:31:34
LMCOMCIRCUIT	SendingPeteConrad ComToEarth	speech	"Nope."	0:00:07	8:31:39	116:31:39
LMCOMCIRCUIT	SendingAlBean ComToEarth	speech	"We ought to be able to move out with this thing."	0:00:10	8:31:42	116:31:42
LMCOMCIRCUIT	SendingPeteConrad ComToEarth	speech	"Okay."	0:00:12	8:31:44	116:31:44
LMCOMCIRCUIT	SendingAlBean ComToEarth	speech	"The experiment bay looks real good."	0:00:16	8:31:48	116:31:48
LMCOMCIRCUIT	SendingPeteConrad ComToEarth	speech	Yup.	0:00:17	8:31:49	116:31:49
LMCOMCIRCUIT	SendingAlBean ComToEarth	speech	"The LM exterior looks beautiful the whole way around. Real good shape. Not a lot that doesn't look the way it did the day we launched it."	0:00:18	8:31:50	116:31:50
LMCOMCIRCUIT	SendingPeteConrad ComToEarth	speech	"Light one."	0:00:30	8:32:02	116:32:02
LMCOMCIRCUIT	SendingAlBeanCom ToEarth	speech	"Okay. Here we go, Pete. Ohhhhh, up they go, babes. One ALSEP."	0:00:40	8:32:12	116:32:12
LMCOMCIRCUIT	SendingPeteConrad ComToEarth	speech	"There it is."	0:00:50	8:32:22	116:32:22

The data from Table 6-11 shows that the simulation, indeed, generates the communication transcription from the Apollo LSJ, herewith validating the output of the simulation model. This concludes the validation of the model. In the next section some conclusions will be discussed.

6.10 CONCLUSION

In conclusion, I restate the research questions that needed to be answered, and show that indeed these questions are answered in this experiment. These questions are operationalized in the Apollo 12 ALSEP domain, and this operationalization is implemented in a Brahms model of the domain. The goal of this experiment was to investigate the use of the Brahms-language in order to *describe an existing work practice*. The challenge was to investigate if our theory of modeling work practice, implemented in the Brahms language, would be sufficient to describe the work practice in the chosen domain. The research questions were:

1. How can we represent the people, things, and places relevant to the domain?
2. How can we represent the actual behavior of the people, second by second, over time?
3. How can we show which of the tools and artifacts are used when, and by whom to perform certain activities?
4. How can we include the communication between co-located and distributed people, as well as the communication tools used, and the effects of these communication tools on the practice?

Table 6-12 shows how these questions were implemented in the Brahms model. The first column shows a more detailed instantiation of the research questions. The second column shows the operationalization based on the Apollo 12 mission. The third column shows how this is implemented in the Brahms model, thus answering the question in the first column.

Table 6-12. Answering the research questions

Research Question	Operationalization in Apollo 12 ALSEP Offload	Implementation in Brahms Model
How to represent people?	The astronauts Al Bean, Pete Conrad on the moon, CapCom Ed Gibson, and CMP Dick Gordon	Agents AlBean, PeteConrad, EdGibson, and DickGordon
How to represent Communities of Practice?	The different organizational roles of Commander, Lunar Module Pilot, Capsule Communicator, and Command Module Pilot. Also, the functional roles of “being an astronaut on the moon” and “offloading the ALSEP.”	Hierarchy of different roles as groups of agents; ApolloAstronaut, CDR, LMP, CMP, CapCom, LunarSurfaceAstronaut, AlsepOffloadGroup
How to represent artifacts?	The artifacts that are used and are important during the lunar surface activity of the two astronauts on the Moon; the LM, SEQ Bay, ALSEP packages, Lanyard Ribbons, Booms, Photo cameras, Space Suits, etc.	Class hierarchy representing types of objects, and objects being instances of classes to represent specific artifacts in the world; LM, SEQBay, AlsepPkg1, AlsepPkg2, Pkg1LanyardRibbons, Pkg2LanyardRibbons, etc.
How to represent places?	The areas where the astronauts are located, Mission Control, the	Type of areas as area definitions. Representing the Apollo 12

	Command Module, and the areas on the moon where the astronauts are working to offload the ALSEP, such as the area in front of the SEQ Bay, etc.	Geography model as the World area, containing the areas Moon, PlanetEarth, and LunarOrbit. Next, the separate areas part of the Moon. Mission Control is part of PlanetEarth, and the CommandModule area is part of the LunarOrbit area.
How to represent location of people and artifacts?	The lunar surface astronauts are located on the Moon, the CapCom is located in Mission Control, and the CMP is located in the Command Module.	Using the <i>initial_location</i> attribute in agents and objects. Each agent and object is given an initial location at the beginning of the simulation. From that moment on agents and objects have locations, which means they are located in an area and can move to other areas when needed.
How to represent actual behavior over time?	During the mission the astronauts are always performing activities. While the CDR and LMP are offloading the ALSEP packages the CapCom is listening on the voiceloop, etc.	The agent's real-life activities are represented as different types of Brahms activities that take time. Composite activities decomposed into primitive activities, communicate activities, and move activities. Behavior of objects, such as the astronaut's space suit and photo camera is also represented as activities. Next, the activities are executed as part of workframes, constrained by the agent's beliefs acquired or changed over time.
How to represent the use of tools and artifacts?	The lunar surface astronauts use tools to perform activities, such as the use of the lanyard ribbons to lower the ALSEP packages from the SEQ Bay.	The use of tools and artifacts in activities is represented using the <i>resources</i> attribute. Also, the generation and detection of facts represent the interaction or use of an artifact in an activity by an agent. The generation of facts is a representation of the actual physical interaction with the artifact being used in the activity. For example, in the <i>taking a photograph</i> activity the agent is using the PhotoCamera object.
How to represent communication?	The communication between the lunar surface astronauts on the moon, the CapCom in Mission Control, and the communication between CapCom and CMP.	Communication is represented as an activity. During this activity beliefs are communicated to/from agents. All the communication between the astronauts is represented as timed activities communicating speechacts, i.e. the speechact is represented as the value of the attribute

		<i>speechact</i> communicated as a belief from one agent to another.
How to represent communication tools?	The Apollo astronauts were on a communication voiceloop circuit with each other and CapCom at Mission Control. The communication time delay between Earth and the Moon was 1.25 seconds.	Voiceloop communication is represented as a communicate-activity, communicating with agents of the group VoiceLoopCommunicator. To represent the time delay in the communication a LmComCircuit agent represents the voiceloop circuit through which the communication is sent to/from Earth to the Moon. Both the agents on Earth and on the Moon communicate through this LmComCircuit agent.

In this Apollo 12 ALSEP Offload experiment I was able to represent the intricate detail of the human activities and collaboration using the Brahms language. The fact that the model generates all the communication between the astronauts, including the timing of the communication, shows that the Brahms Language is powerful enough to model and simulate the work practice of the astronauts on the Moon and on Earth. Of course the level of collaboration is shown in terms of the activities each agent is performing, as well as the location of the agents, the artifacts the agent is using at that moment, and the use of artifacts in the activity. It has been shown that the research questions posted are answered satisfactorily. Therefore we can say that the hypothesis is proven, and that with Brahms we are able to *describe an existing work practice*.

This concludes the first of three experiments to show that Brahms is a sufficient language for modeling and simulating work practice. In the next experiment I will show that with Brahms we can predict the future activity behavior of agents, based on a model of the work practice.

7. CASE STUDY 2: PREDICTING SITUATED ERRORS

A theory is only a good theory if it can predict future events. In this chapter, I show that a Brahms model can be developed that predicts plausible future work practice scenarios (Sierhuis et al. 2000b). The model is of the Heath Flow Experiment (HFE) deployment during the ALSEP deployment EVA. This model is a theory of the work practice of the HFE deployment. In order to prove that this model is a valid implementation of the theory of the HFE work practice, I show that the model predicts the future behavior of the agents based on the behavioral aspects of the agents in the model. If it can be shown that this model can predict future agent behavior, the theory underlying the model (i.e. the theory of modeling work practice) is a valid theory for modeling such behavior. This is the next step in the quest for evidence that Brahms is a tool for modeling and simulating work practice.

Goals and objectives

The objective is to abstract the work practice of the HFE deployment in such a way that we can simulate any HFE deployment plan for situation-specific scenarios. To do this, I focus on three important features that are needed. First, the model needs a general situated activity-plan execution approach. Second, the model needs a general communication policy to handle the necessary planned voice-data communications from the lunar surface astronauts. Third, during error situations in the HFE deployment activity the model has to show the error-recovery behavior of the agents in situations that have *not* been previously described.

If it turns out that we can only model communication and error-recovery by prescribing every situation, it can be said that a general model can never be constructed, and that we cannot use the model to predict plausible behavior of the agents during new and not previously prescribed situations. If, on the other hand, we can model the communication and error-recovery practice of the agents in such a way that we do *not* simply “hardwire” the behavior of the astronauts in previously observed situations, we can use the model to predict what will happen in future situations that have not occurred previously.

The research question in this experiment is:

Can a work practice model of the Heath Flow Experiment deployment procedures on the Moon predict plausible changes in the activity-behavior of the agents in the model, when previously unmodeled events occur?

Proving that we can predict the work practice behavior of agents during new situations is the first step in showing that we can use Brahms to design new work processes. Designing a new work process is the objective of the third and last experiment, in which we will design the work practices of a semi-autonomous robot on the Moon.

It is important to stress that what is predicted in this experiment is the agent behavior performing situated activities from a predefined set, given the occurrence of certain events. It could be easily misunderstood that the model is predicting events or new, not previously defined activities. It should be clear that this is not the case. By predicting the activities the agents perform from a pre-specified set in given events, we are validating the model being a theory of work practice for the HFE deployment on the Moon.

Approach

I develop a model of work practice for the HFE deployment based on the Final Apollo 16 Lunar Surface Procedures document, in which the detailed nominal timed activities during the HFE deployment are specified (Kain et al. 1972). Using the actual Apollo 16 voice-loop data and mission video I am able to analyze how the procedures are performed in practice. Unlike the Apollo 12 ALSEP Offload model from the previous experiment, this model is not a descriptive model of what happened during the Apollo 16 mission. Instead, this model is a more abstract and general model of the HFE deployment activity, based on an analysis of how the planned nominal procedures are executed in practice. After the general HFE deployment activity is implemented, I will include the error-recovery activities of the astronauts based on observed and analyzed errors during the Apollo 15 and 16 missions. During both these missions, events

occurred that made the astronaut deviate from the nominal planned procedures. Using these situated-specific events, a generic activity model with which the agent can react appropriately and plausibly in “future” error events will be shown.

As a validation step of this generic error-recovery activity, previously observed error events from Apollo 15 and 16 will be generated, and the behavior of the agents during the simulation will be compared to those observed. In the next sections I describe the important aspects of the model, and the results of the validation. Last, there will be some conclusions about the results from this experiment. The description of the model will not be in as much detail as that of the first experiment in Chapter 6. Many of Brahms specific features are detailed in that chapter, as well as in chapter 4.

7.1 HFE DEPLOYMENT

The purpose of the Heat Flow Experiment (HFE) was to measure the thermal conductivity and temperature gradient of the upper 2.44 meters of the lunar surface. They predicted two heat sources active in the Moon’s interior: (1) original heat from the time of the Moon’s formation, and (2) radioactivity. The objective of the experiment was to gather data on the Moon’s internal heating process and use it as a basis for comparing the radioactive content of the Moon’s interior with the Earth’s mantle. In addition to the Moon’s internal temperature, the experiment was capable of measuring the thermal conductivity of the lunar rock material. These combined measurements would give a net heat flux from the lunar interior to the lunar surface. Similar measurements on Earth had contributed to the understanding of volcanoes, earthquakes, and mountain building processes. Together with seismic and magnetic data from the other experiments, the HFE data would give scientist the ability to develop better models of the Moon, and therefore develop a better understanding of its history and origin.

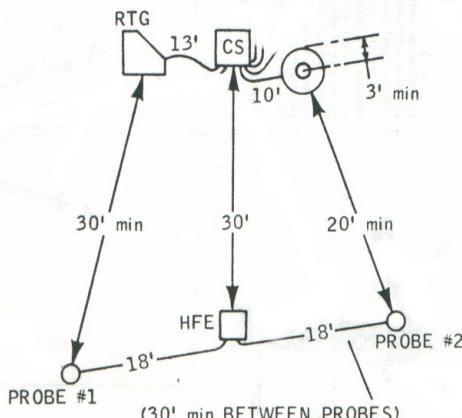


Figure 7-1. HFE deployment configuration

The HFE (S-037) was part of the Apollo Lunar Surface Experiments Package (ALSEP), and was part of the mission plans for Apollo 13, 15, 16, and 17. As many people know, Apollo 13 never made it to the Moon. The first deployment was during EVA-1 of Apollo 15. Although problematic, the experiment was deployed and data from the lunar internal at Hadley-Apennine was sent to Earth. Next was the deployment planned for Apollo 16, also during EVA-1. Unfortunately, the experiment was broken during the deployment and no data was ever received from Descartes-Cayley. The last HFE deployment was at Taurus-Littrow during the first EVA of Apollo 17. There, things finally went according to plan.

The HFE consisted of two heat-flow probes, an electronics box, a probe emplacement tool, and the Apollo lunar surface drill (ALSD). Figure 7-1 shows that in the deployment configuration each of the two probes was connected by a cable to the HFE electronics box that rested on the surface, at the center of the two probes. The two probes were positioned on either side of the electronics box in a straight line about 18 feet away. The HFE electronics box provided control, as well as monitor and data processing capability for the experiment. The box was connected with a cable to the ALSEP central station 30 feet away. The astronaut responsible for deploying the HFE used the ALSD to drill two lined boreholes in the lunar surface to insert the probes. By drilling three hollow drill-rod sections into the surface a lined borehole was created. A closed drill-bit on the first drill-rod allowed for penetration into the lunar surface. After the lined borehole was created an emplacement tool was used to insert the probe to full depth.

7.1.1 HFE time line

One astronaut performed the HFE deployment solo, while the other astronaut was deploying other ALSEP instruments. During the Apollo missions different astronaut-roles deployed the HFE. During the Apollo 15 and 17 missions the CDR deployed the HFE, however during the Apollo 16 mission it was the LMP. I have not been able to uncover the reason for this switch, but it might have been as simple as personal preference

of the individual astronauts. Regardless of this switch, the plan for deploying the HFE was the same on all missions. Figure 7-2 shows the summary time line of the ALSEP offload and deployment for the Apollo 16 mission.

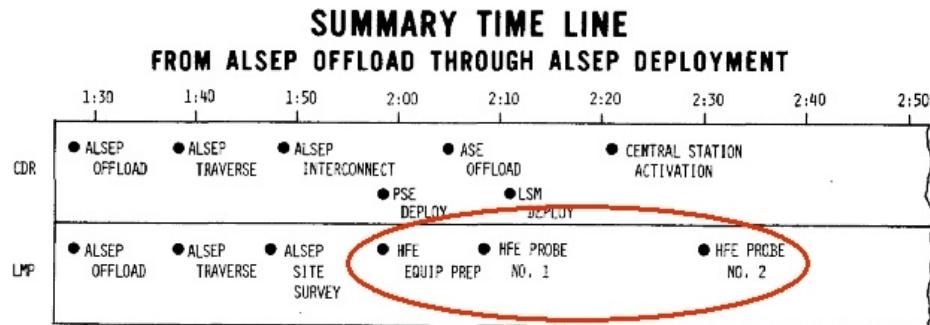


Figure 7-2. Apollo 16 summary time line

It shows the high-level ALSEP Offload and Deployment activities of the CDR and LMP over time. The ellipse in Figure 7-2 shows that the HFE deployment is performed by the LMP and consists of three high-level activities: HFE equipment preparation, deploying HFE Probe 1 and 2.

7.1.2 HFE Deployment procedures

The summary time line from Figure 7-2 was decomposed for planning and training purposes into a very detailed step-by-step EVA procedure for each astronaut. In this procedure every step or activity is timed and planned. Figure 7-3 shows this detailed procedure for the LMP from the Apollo 16 Final Lunar Surface Procedures document (Kain et al. 1972). This is the data used to develop the work practice model.

1		2		3	
MISSION: APOLLO 16 EVA: 1		MISSION: APOLLO 16 EVA: 1		MISSION: APOLLO 16 EVA: 1	
LMP ACTIVITIES	EVA TIME	LMP ACTIVITIES	EVA TIME	LMP ACTIVITIES	EVA TIME
REMOVE HFE SUBPALLET Release pull rings (2); pull pip pins (3)	1+50	<u>BORE HOLE 1 DRILLING</u> Set drill on surface Lean bore/core stem bag against rack; open bag Insert 54" bore stem into drill Pick up drill & push bit into surface as far as possible Remove batt. thermal shield Energize drill until stem top approx. 16" above surface	2+10	Place sunshield over top of stem <u>DRILL BORE HOLE 2</u> Carry drill, rack, rammer and bore/core stems to hole 2 Set drill on surface Lean bore/core stem bag and rammer against rack Insert 54" bore stem into drill Pick up drill & push bit into surface as far as possible Energize drill until stem top is approx. 16" above surface	2+30
MISSION: APOLLO 16 EVA: 1		ATTACH WRENCH TO BORE STEM Attach wrench to bore stem Remove drill from stem Screw 28" bore section to section in surface Screw drill onto bore section Energize drill until stem top approx. 16" above surface	2+20	ATTACH WRENCH TO BORE STEM Remove drill from stem Screw 28" bore section to section in surface Screw drill onto bore section Energize drill until stem top is approx. 16" above surface	2+40
LMP ACTIVITIES	EVA TIME	ATTACH WRENCH TO BORE STEM Attach wrench to bore stem Remove drill from stem Screw 28" bore section to section in surface Screw drill onto bore section Energize drill until stem top approx. 11" above surface	2+30	ATTACH WRENCH TO BORE STEM Remove drill from stem Screw 28" bore section to section in surface Screw drill onto bore section Energize drill until stem top is approx. 11" above surface	2+50
Pull M/P base pin #1; unwrap tape; remove cover; pull pin #2 Remove M/P; set on surface Pull subpallet pip pin Rotate Pkg 2 to surface, align Remove HFE - 2 BB & connector Place HFE on surface Connect HFE to C/S - lock <u>DEPLOY HFE</u> Carry HFE pallet 30' S of C/S, place on surface Remove probe box from pallet - 4 BB Split probe box (2 velcro straps) Carry half with rammer to HFE Hole #1 (~ 18' W) Place box half on surface Carry other box half to HFE Hole #2 (~ 18'E), place box on surface Go to LRV	1+50	EMPLACE HFE PROBE 1 Remove HFE probe from box Deploy rammer, lean on rack Insert probe and first thermal shield into bore hole, using rammer Report probe depth & stem height above surface Emplace second thermal shield to 21" depth	2+30	EMPLACE HFE PROBE 2 Remove HFE probe from box Insert probe and first thermal shield into bore hole, using rammer Report probe depth & stem height above surface Emplace second thermal shield to 21" depth	2+50
Set bore/core stems on surface Configure ALSO hardware: • Verify motor operates • Pull pin #2 • Rotate rack camloc 90° • Rotate batt camloc 90° pull pin lanyard • Remove handle and install on battery • Rotate rack bracket up • Lift rack off treadle, extend legs and set on surface • Pull pin #5, move bracket & lift drill Carry drill, rack and bore/core stem bag to HFE Hole #1 site	2+00	MISSION: APOLLO 16 EVA: 1		LMP ACTIVITIES	EVA TIME
	2+10			Place sunshield over top of stem Remove HFE elec. box from pallet Remove dust cover (4BB) Level & align HFE elec box Remove all debris at least 16' away from HFE area Place UHT on LRV left floorboard	2+50

Figure 7-3. Apollo 16 HFE deployment timeline procedures for LMP

The high-level activity *HFE Equipment Preparation* in Figure 7-2 appears in Figure 7-3 as the detailed procedures *Remove HFE SubPallet* and *Deploy HFE*. The *HFE Probe 1* activity is decomposed into the procedures *Bore Hole 1 Drilling* and *Emplace HFE Probe 1*. Similarly, *HFE Probe 2* is decomposed into *Bore Hole 2 Drilling* and *Emplace HFE Probe 2*.

7.1.3 HFE Deployment activity model

Figure 7-4 shows the activity decomposition for the HFE deployment procedure from Figure 7-3. The HFE deployment procedure consists of the *HfeEquipmentPreparation* and the *DeployingHfeProbe* composite activities. Every astronaut that has trained to deploy the HFE knew how to perform these activities in the HFE deployment procedure. The HFE deployment activities from Figure 7-3 are therefore represented in a Brahms activity model that is implemented into the group *AstronautsThatCanDeployTheHFE*. All members of the *AstronautsThatCanDeployTheHFE* group are able to perform these activities.

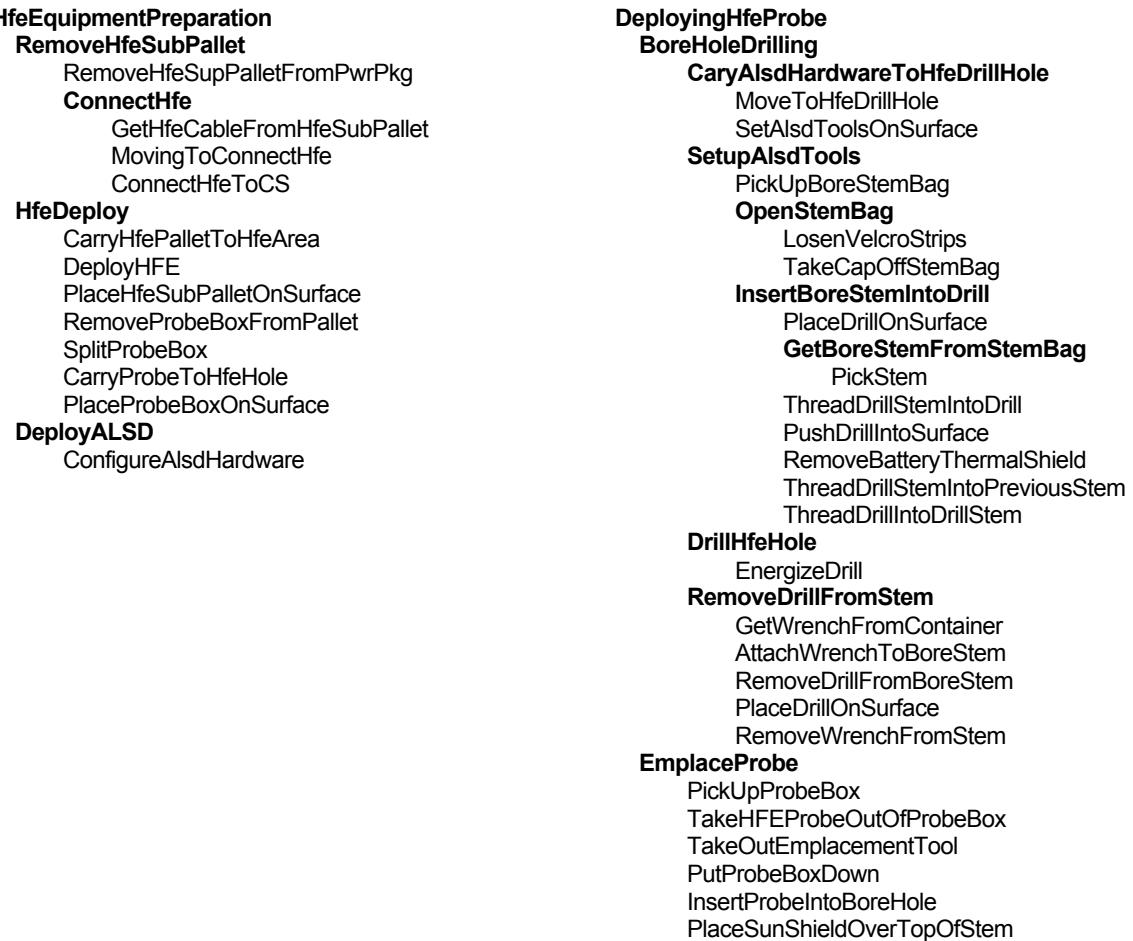


Figure 7-4. HFE Deployment Activity ModelHfeEquipmentPreparation Activity

Before the astronaut can drill the HFE boreholes and insert the HFE probes, he has to prepare the equipment for deployment. The HFE subpallet is stored on the Power Package (PwrPkg; this is the 2nd ALSEP package that also contains the RTG power supply). So, first the astronaut takes the HFE pallet off the PwrPkg and lays it on the surface next to the PwrPkg. Now the HFE instrument on the pallet has to be connected to the Central Station (C/S). The astronaut grabs the flat cable with the C/S connector and walks over with it to the C/S (see Figure 7-5, movement 1). There he connects the HFE instrument to the C/S. He then goes back to pick up the HFE instrument and carries it a minimum of 30 feet away from the C/S. In case of the Apollo 16, the HFE deployment area is 30 feet South of the C/S (see Figure 7-5, movement 2). After he positions the HFE pallet on the surface, he takes off the HFE probe box in which the two HFE probes are held. The box is in two halves held together with Velcro strips. The astronaut separates the two halves of the probe box. Next, he carries the probe box halves 18 feet to the left and right respectively, to the HFE borehole areas (see Figure 7-5, movements 3 & 4). He puts the instruments where he will return to drill the holes. As the last activity in the equipment preparation, the astronaut needs to get the ALSD, the HFE bore, core stems and the bore stem rack together. These are located on the Lunar Rover (LRV), and thus the astronaut has to go to the LRV, get the drill, bore stems and rack from the rover and configure them (see Figure 7-5, movement 5).

7.1.3.1 DeployingHfeProbe Activity

Next, the most complex activity is the actual deployment of the two HFE probes. As can be seen from Figure 7-3, these two activities are identical. Therefore, the model requires only one *DeployHfeProbe* activity. This activity is performed twice, once for the first probe and once for the second, at a different location with the same drill, but using different bore stems and a different probe. The actual task is pretty simple, and if performed on Earth should not create a lot of problems. However, on the Moon in 1/6 G, this

task is quite an effort. First, the astronaut has to carry the ALSD, the bore stems and bore rack to the deployment site, i.e. the drill hole (see Figure 7-5, movement 6).

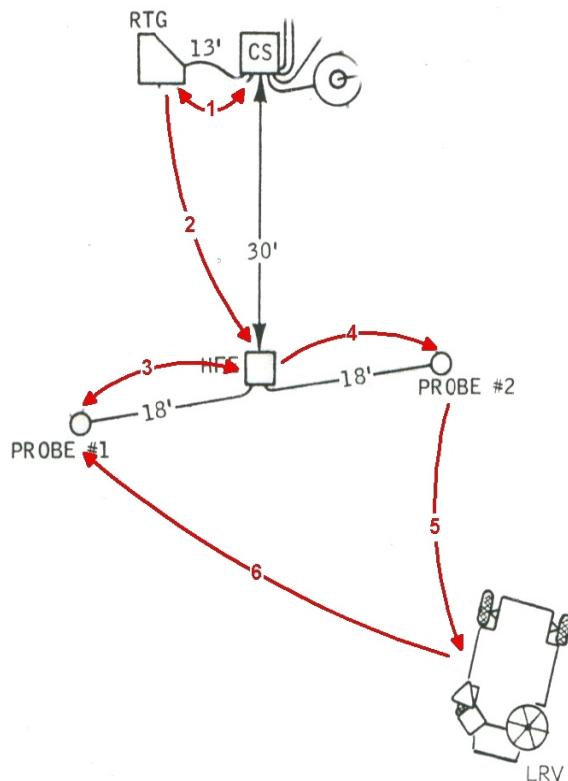


Figure 7-5. Astronaut movement during HFE Deployment activity

The astronaut picks up the probe box to take out the HFE probe and the probe emplacement tool. He then inserts the HFE probe into the borehole, using the emplacement tool. After the probe is emplaced a thermal shield is placed into the borehole. As a last activity, the astronaut places a sun shield over the stem to protect it.

7.2 PREDICTIVE MODELING

The Apollo 12 model of the ALSEP Offload from the previous experiment (chapter 6) is a descriptive model. The agents perform their activities and communications in a pre-specified, more or less “hardwired” way. The model can only simulate the ALSEP Offload as it happened during the Apollo 12 mission. The question that is being addressed in this experiment is how we could develop a work practice model that can be used to simulate the work practice during any HFE deployment mission. Such a model would be a predictive model of the work practice.

There are a number of issues in the design of a predictive model that will be worked out in the next sections. First, I will describe how we can go from a “hardwired activity model” to a more flexible model in which the ordering of an agent’s activities is flexible enough that it can change dynamically, rather than having to change the source code of the model. Secondly, I will address the issue of having conversations. A conversation is a dynamic and situated activity. Agents need to be able to act and react to speech acts in a dynamic fashion, such as asking a question when there is a need for information, and answering the question if it is being asked.

Next, the ALSD tools have to be set up and made ready for drilling. This entails opening the core stem bag, picking the first core stem from the bag and attaching it to the ALSD. Then the core stem is pushed with the ALSD into the surface so that the drilling can start. By starting the drill, the astronaut drills the first bore stem into the surface. When the bore stem is far enough into the surface, the ALSD has to be removed from the drill. Using the wrench does this. The drill is positioned on the surface and a second bore stem is grabbed from the bag. Instead of screwing the bore stem onto the drill laying on the surface, the second stem is screwed on the previous stem already drilled into surface. Then the ALSD is picked up and screwed on to the new bore stem. After this has been accomplished, the astronaut activates the ALSD again and the second bore stem is drilled further into the surface. Next, it is time for the third and last bore stem. After all three bore stems are drilled into the surface the astronaut has to place the HFE probe into the borehole. The probe is located in the probe box that was positioned near the borehole during the *HfeEquipmentPreparation* activity.

7.2.1 Dynamic plans and schedules

The activity plan for the agents in the Apollo 12 ALSEP Offload model was hardwired into the workframes of the model. After an agent is finished with an activity, the agent knows which next activity to perform, because a consequence in the workframe creates the belief about what activity to perform next.

For example, Figure 7-6 shows a hardwired plan in which, after the agent has moved to the SEQBayArea, he will open the SEQ Bay door as his next activity, regardless of the situation that arises at the SEQ Bay.

```
workframe MovingToSEQBay {  
    repeat: false;  
    when (not(the agentLocation of current = SEQBayArea))  
    do {  
        conclude((current.currentActivity = MoveActivity), bc:100, fc:100);  
        Move(SEQBayArea, 5, 1);  
        conclude((current.nextActivity = sOpenSEQBayDoorActivity), bc:100, fc:0);  
    }  
}
```

Figure 7-6. Hardwired activity plan

If we want to use this plan for another Apollo mission, or another future HFE deployment mission, it is obvious that this approach is not flexible enough to plan new situations. Therefore, a more dynamic approach for the scheduling of activities for the agents needs to be implemented, without having to specify every context in which an activity might occur (Agre 1995).

The approach taken is to view the astronauts' activity of determining what next activity to work on as part of their work practice. As such, determining what activity to work on next is explicitly represented as an activity in the model instead of hardwiring it in workframes, such as the workframe shown in Figure 7-6. The operationalization of this activity in the real world can be found in the use of the *cuff-checklist* the astronauts are wearing on their space suit. In effect, this artifact contains a plan for action. When the astronaut is done with an activity, he consults his cuff-checklist to determine the next activity. This is modeled as the *DetermineNextActivity* activity. In this activity the astronaut simply reads from his cuff-checklist what his next activity is supposed to be⁵¹. The *cuff-checklist* artifact is modeled as located on his space suit (using the containment relation), and contains the astronaut's activity schedule (represented as "beliefs in the object").

Figure 7-7 graphically represents this activity. The astronauts' plans are modeled as objects with relations to the *previous* and *next* activities to be performed. By specifying the sequential ordering relationships between activities, we specify the astronaut's plan to be performed. Even more, because the plan is modeled as *beliefs* about activity objects, changing the beliefs about the next activity can dynamically change the plan. This also allows for the agent to dynamically receive or change his beliefs about the plan. This is a crucial element in the design of this approach, which is used to model the change in the plan during situated errors.

Next, the model represents under what conditions the astronaut will perform the *DetermineNextActivity*. This is done using a very simple workframe, shown in Figure 7-8. Whenever the agent gets the belief

(current.getNextActivity = true),

the *ReadingCuffCheckList* workframe (Figure 7-8) gets executed. This immediately makes the agent perform the *DetermineNextActivity* activity, while interrupting any current activity due to the high priority given to the execution of the *DetermineNextActivity* activity (priority = 100).

⁵¹ It should be noted that the agent is *not* planning in this activity, but reading a pre-specified plan developed during the mission planning activity, developed and trained long before the actual mission.

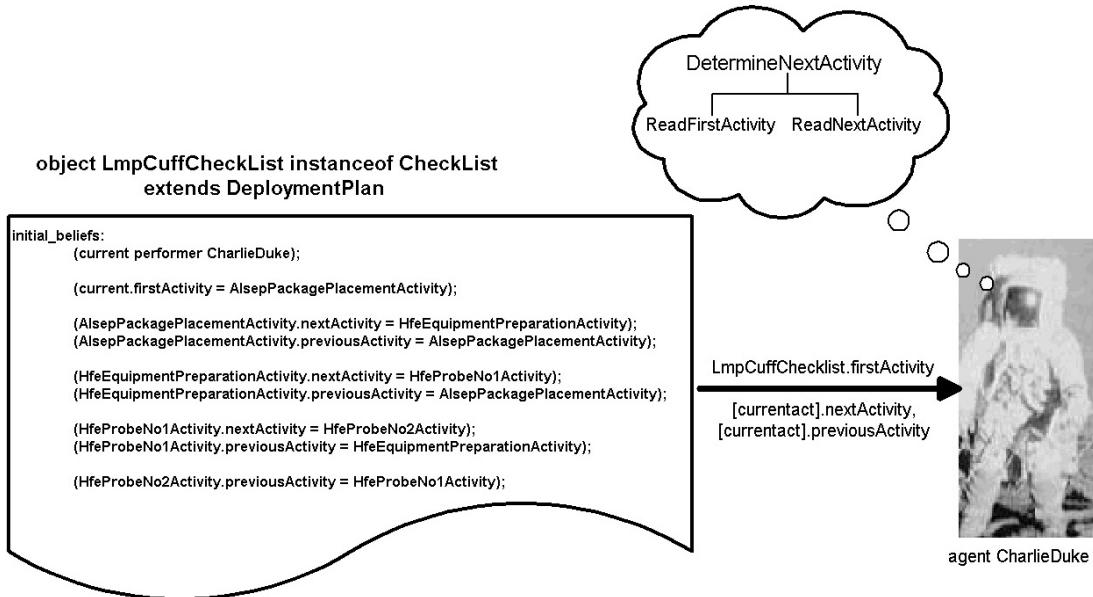


Figure 7-7. Dynamic plan execution activity

```

workframe ReadingCuffCheckList {
  repeat: true;
  when (knownval(current.getNextActivity = true))
  do {
    DetermineNextActivity(start, 100, 1);
    conclude((current.getNextActivity = false), bc:100, fc:0);
  }
}

```

Figure 7-8. ReadCuffChecklist workframe

Figure 7-10 gives the source code of the *DetermineNextActivity* composite activity. This activity is defined within the *ApolloAstronaut* group. Thus all members of the *ApolloAstronaut* group can potentially perform this activity. However, in the HFE deployment model, only members of the *LunarSurfaceAstronaut* subgroup have the *ReadingCuffChecklist* workframe, from Figure 7-8. This means that only the members of that group (i.e. the LMP and CDR agents) will read a cuff-checklist. Within the *DetermineNextActivity* composite activity there is a communicate-activity for reading the first activity from the checklist. There also is a communicate-activity to read all the next activities from the checklist. There are two separate workframes, *ReadingFirstActivity* and *ReadingNextActivity*, that make the agent perform the *DetermineNextActivity* activity. Obviously, when the agent checks his cuff-checklist for his first activity, the *ReadingFirstActivity* workframe is executed. After this, every time the agent needs to determine his next activity, he will execute the *ReadNextActivity* workframe.

Importantly, there are two thoughtframes in the composite activity (see Figure 7-10). One of the thoughtframes is fired every time, immediately after the agent has performed one of the two reading next activity workframes. These thoughtframes create the belief for the agent to start performing the correct next activity,

```
conclude((current.nextActivity = curact.nextActivity), bc:100, fc:0);
```

When the agent gets an updated belief for the attribute *nextActivity* it immediately triggers a workframe to perform the next activity. This workframe needs at a minimum a precondition of the form

```
knownval(current.nextActivity = <activity object name>).
```

Every possible activity needs to be an object-instance of the class *Activity*. For example, the workframe that starts the activity for HFE Probe1 deployment is shown in Figure 7-9. There is an object *HfeProbeNo1Activity* that is an instance of the class *Activity*.

```

workframe DeployingHfeProbeNo1 {
    repeat:false;
    when (knownval(current.nextActivity = HfeProbeNo1Activity))
    do {
        conclude((current.currentActivity = HfeProbeNo1Activity), bc:100, fc:100);
        DeployingHfeProbe(0);
        conclude((current.getNextActivity = true), bc:100, fc:0);
    }
}

```

Figure 7-9. Workframe for DeployingHfeProbeNo1

The workframe from Figure 7-9 has to be read as follows: When the agent knows that his next activity is HfeProbeNo1Activity he starts working on that activity by first concluding that this is his current activity, then he starts performing the DeployHfeProbe activity. After he is done with the DeployHfeProbe activity he concludes that he has to get his next activity from the plan.

7.2.2 Question and answer conversation policy

Similar to the hardwired plan, the Apollo 12 ALSEP Offload model also has the agent communication hardwired in the workframes. Obviously, most *conversations* are not hardwired, meaning, in situated activities we don't have pre-specified speech acts that are always spoken⁵². People don't just utter loosely connected speech-act sequences. They engage in *conversations*, i.e. a sequence of speech-acts between individuals that when interpreted in the context of a larger activity belong together and form an interpersonal dialogue. In order to model conversations, Holmback (1999) argues that we need to predefined the policy of specific types of conversations (Holmback et al. 1999b) (Holmback et al. 1999a).

⁵² There are of course exceptions, such as a standard greeting.

```

composite_activity DetermineNextActivity(symbol whn, int pri, int dur) {
    priority: pri;
    activities:
        communicate ReadFirstActivity(DeploymentPlan plan, symbol whn, int pri, int mdur) {
            priority: pri;
            max_duration: mdur;
            with: plan;
            about:
                receive(plan.firstActivity = value);
            when: whn;
        }
        communicate ReadNextActivity(DeploymentPlan plan, Activity act, symbol whn, int pri, int mdur) {
            priority: pri;
            max_duration: mdur;
            with: plan;
            about:
                receive(act.previousActivity = value),
                receive(act.nextActivity = value);
            when: whn;
        }
    workframes:
        workframe ReadingFirstActivity {
            repeat: false;
            variables:
                forone(Activity) curact;
                forone(CheckList) checklist;
            when (unknown(current.currentActivity = curact) and
                  knownval(current contains checklist))
            do {
                ReadFirstActivity(checklist, whn, pri, dur);
            }
        }
        workframe ReadingNextActivity {
            repeat: false;
            variables:
                forone(Activity) curact;
                forone(CheckList) checklist;
            when (knownval(current.currentActivity = curact) and
                  knownval(current contains checklist))
            do {
                ReadNextActivity(checklist, curact, whn, pri, dur);
            }
        }
    thoughtframes:
        thoughtframe FirstActivity {
            repeat: false;
            variables:
                forone(CheckList) checklist;
            when (knownval(current contains checklist) and
                  known(checklist.firstActivity = value) and
                  unknown(current.nextActivity = checklist.firstActivity) and
                  unknown(current.hasStarted = true))
            do {
                conclude((current.nextActivity = checklist.firstActivity), bc:100, fc:0);
                conclude((current.hasStarted = true), bc:100, fc:0);
            }
        }
        thoughtframe NextActivity {
            repeat: false;
            variables:
                forone(Activity) curact;
            when (knownval(current.hasStarted = true) and
                  knownval(current.nextActivity = curact.name))
            do {
                conclude((current.nextActivity = curact.nextActivity), bc:100, fc:0);
            }
        }
    }
}

```

Figure 7-10. DetermineNextActivity source code

During the Apollo missions, the detailed EVA timeline procedures included the specification of the voice-data that the CDR and the LMP were to communicate during their activities. Figure 7-11 shows a piece of

the voice-data the astronauts need to communicate during the HFE deployment activities, that is part of the lunar surface procedures (Kain et al. 1972). The voice-data schedule includes what data to report, which of the astronauts is to report the data, and also the priority of the data (this is the number one or two between the rounded brackets in Figure 7-11). A priority one (1) means that there is a mandatory requirement for the data at the time or event designated. A priority two (2) means that the data may be deferred until the debriefing.

It is easy for people to forget to make such scheduled utterances in the moment, even if they are well planned and trained. This happened a number of times during the Apollo 16 EVA. When this occurred, the CapCom, keeping track of the schedule, would ask the astronaut for the data later.

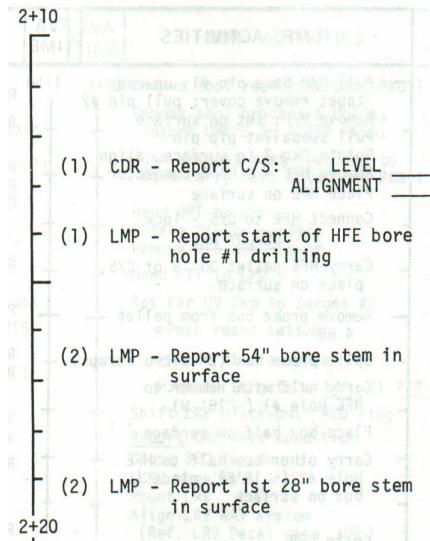


Figure 7-11. Voice-data schedule

To model this kind of interaction, I introduce a type of conversation policy called the *question and answer voice-data policy*. The purpose of this policy is to represent how the CapCom and the lunar surface astronauts handle the voice-data that needs to be reported back to Earth. The policy consists of two parts, one part describing the policy for the *receiver* (i.e. the agent(s) needing the answer to a question) and one part for the *sender* (i.e. the agent answering the question).

The operationalization of this policy for the lunar surface EVA's is as follows. The *question* is the indirect request for the voice-data as is specified in the lunar surface procedures (see Figure 7-11). The *sender policy* (or answer policy) is the utterance of the astronaut at the moment that the voice-data is to be reported.

The *receiver policy* defines when the CapCom is going to ask specifically for the data (i.e. a direct request). For example, if the astronaut forgets to provide the data as was specified in the schedule and the data has a high priority, the CapCom will ask for the data during the EVA. Figure 7-14 shows the conversation policy.

VoiceData Conceptual Object Class

Every piece of voice-data that needs to be communicated according to the plan will be represented as a *VoiceData* conceptual object (Figure 7-12). The voice-data is represented as a conceptual object, because it is not a physical object, but represents a conceptual voice-data element of the schedule the astronaut *remembers* from their extensive training. Since the astronaut's cuff-checklist does not include all the voice-data⁵³, they have to remember to communicate the voice-data at the appropriate time during the performance of their activities.

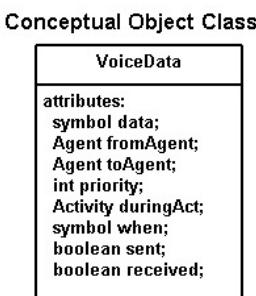


Figure 7-12. VoiceData definition

⁵³ Some astronauts added references to specific voice-data as reminders on their personal cuff-checklist, but this was the exception and they certainly did not contain all voice-data.

Each voice-data object has the following meta-data associated with it:

- *data*: This represent the data that is communicated.
- *fromAgent*: This links the voice-data object to the agent that needs to communicate the data.
- *toAgent*: This links the voice-data object to the agent that needs to receive the data.
- *priority*: This is the priority ("1" or "2") of the voice-data.
- *duringAct*: This links the voice-data object to the activity after which the *fromAgent* needs to communicate the data.
- *when*: This tells when the voice-data needs to be communicated during the activity; at the beginning, or at the end of the activity ("start" or "end").
- *sent*: This tells if the voice-data has already been communicated or not ("true" or "false").
- *received*: This tells if the voice-data has already been received or not ("true" or "false").

For each planned voice-data communication there is a conceptual object. The agents get (initial) beliefs about these objects as a representation of the fact that they remember the scheduled communications from their training, and remember who needs to communicate it and when. For example, Figure 7-13 shows that at the moment the LMP has removed the battery thermal shield from the ALSD, he needs to report the start of the HFE bore hole #1 drilling.

VOICE DATA		MISSION: APOLLO 16		DATE: MARC	
		EVA: 1			
2+10	LMP ACTIVITIES	EVA TIME	CDR ACTIVITIES		
	BORE HOLE 1 DRILLING Set drill on surface		2+10 Deploy 2nd leg, position legs Place M/P NNE C/S; point NW		
	Lean bore/core stem bag against rack; open bag		REMOVE LSM Release LSM from C/S - 2 BB		
	Insert 54" bore stem into drill		Lift LSM off C/S, ck. cable		
	Pick up drill & push bit into surface as far as possible		Set LSM on surface clear of C/S		
	Remove batt. thermal shield		ERECT CENTRAL STATION		
	Energize drill until stem top approx. 16" above surface		Level and align Pkg 1 Release C/S So. side - 5 BB		

Figure 7-13. Voice-data example

This voice-data instance is represented as the conceptual object *ReportStartHfeBoreHoleDrilling*

```
conceptual_object ReportStartHfeBoreHoleDrilling instanceof VoiceData { }
```

The lunar surface astronauts and the CapCom get initial-beliefs about this conceptual object. Each agent that remembers the planned voice data communication has beliefs about the *ReportStartHfeBoreHole1Drilling* conceptual voice-data object, such as:

```
(ReportStartHfeBoreHoleDrilling.duringAct = RemoveBatteryThermalShieldActivity);
(ReportStartHfeBoreHoleDrilling.whn = end);
(ReportStartHfeBoreHoleDrilling.fromAgent = CharlieDuke);
(ReportStartHfeBoreHoleDrilling.toAgent = TonyEngland);
(ReportStartHfeBoreHoleDrilling.pri = 1);
(ReportStartHfeBoreHoleDrilling.sent = false);
(ReportStartHfeBoreHoleDrilling.received = false);
```

Not having beliefs about a voice-data object means that the agent does *not* remember that this data needs to be communicated. This is the way forgetting to communicate the voice data becomes possible. Next, I describe how the Question & Answer Conversation Policy is implemented in the Brahms model.

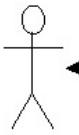
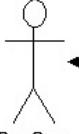
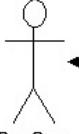
Sender Policies	
LunarSurfaceAstronaut	
Name:	Sender Policy 1
Description:	When the sender needs to communicate voice data at the end of the activity
Data:	VoiceData dataObject
Activity:	CommunicateVoiceData(dataObject.data,dataObject.fromAgent,dataObject.duringActivity)
Precondition:	knownval(dataObject.duringAct.isDone = true) AND knownval(dataObject.priority = 1) AND knownval(dataObject.when = end) AND knownval(dataObject.sent = false)
Consequence:	knownval(dataObject.send = true)
CapCom	
Name:	Sender Policy 2
Description:	When the sender needs to communicate voice data at the beginning of the activity
Data:	VoiceData dataObject
Activity:	CommunicateVoiceData(dataObject.data,dataObject.fromAgent,dataObject.duringActivity);
Precondition:	knownval(VoiceData.duringAct = current.currentActivity) AND knownval(dataObject.priority = 1) AND knownval(VoiceData.when = start) AND knownval(VoiceData.send = false)
Consequence:	knownval(dataObject.send = true)
Receiver Policies	
CapCom	
Name:	Receiver Policy 1
Description:	When the receiver needs to ask for the voice data
Data:	VoiceData dataObject
Activity:	RequestForData(dataObject.data);
Precondition:	knownval(dataObject.priority = 1) AND knownval(dataObject.duringAct.isDone = true) AND knownval(dataObject.received = false)
Consequence:	knownval(dataObject.received = true)

Figure 7-14. The Question & Answer conversation policy

Sender Policies

Sender Policy 1 (SP1) and Sender Policy 2 (SP2) define how the sender of the voice-data communicates the data. This is considered the answer-part of the policy. The condition under which the sender will communicate the data is given in the precondition row of Figure 7-14. For example, the precondition in SP1 states that if the sender is done with the activity during which he needs to communicate the data, and he has to communicate the data at the end of the activity, and the data to be communicated has priority one, then communicate the data using the *CommunicateVoiceData* activity. The precondition in SP2 states that if the sender is working on the activity during which he needs to communicate the data, and he has to communicate the data at the beginning of the activity, then communicate the data using the *CommunicateVoiceData* activity.

Each sender policy is implemented as a separate workframe. For example, SP1 is implemented in the workframe *SenderPolicy1* (see Figure 7-15).

```
workframe SenderPolicy1 {
    repeat: true;
    variables:
        forone(VoiceData) voice_data;
        forone(Activity) act;
        collectall(LmVoiceLoop) vlcoms;
    when (knownval(voice_data.duringAct = act) and
        knownval(current.subActivity = act) and
        knownval(act.isDone = true) and
        knownval(voice_data.whn = end) and
        knownval(voice_data.pri = 1) and
        knownval(voice_data.fromAgent = current) and
        not(voice_data.sent = true) and
        knownval(vlcoms.communicationType = LmVoiceLoop) and
        not(current.name = vlcoms.name))
    do {
        conclude((voice_data.data = current.speechAct), bc:100, fc:0);
        CommunicateVoiceData(100, 1, start, voice_data, act, vlcoms);
        conclude((voice_data.sent = true), bc:100, fc:0);
    }
}
```

Figure 7-15. Workframe *SenderPolicy1*

The actual voice-data speech-act in both SP's is implemented using the *CommunicateVoiceData* communication activity. Figure 7-15 shows how this activity is called from the SP1 workframe, with the correct parameter values. The source code of the communication activity is shown in Figure 7-16. The sender communicates the *data* and *fromAgent* attributes for the voice-data. The sender also communicates the (*sub*)*activity* he is working on when communicating the voice-data. This means that the receiver will get the data that needs to be communicated, as well as who communicated it and during what activity.

```
communicate CommunicateVoiceData(int pri, int dur, symbol whn, VoiceData vd,
                                Activity act, LmVoiceLoop vlcoms)
{
    priority: pri;
    max_duration: dur;
    resources: act;
    with: vlcoms;
    about: send(vd.data = anyvalue),
            send(vd.fromAgent = current),
            send(current.subActivity = act);
    when: whn;
}
```

Figure 7-16. *CommunicateVoiceData* activity

The reason for communicating all this data is that although the voice-data communication schedule—implemented by the voice-data conceptual objects and the agents' beliefs about them—specifies who is to communicate the data, during the mission another agent might actually communicate the voice-data.

Similarly, while the plan specifies during what activity the voice-data has to be communicated, in reality this might be done during another activity.

Receiver Policies

The Receiver Policy1 (RP1) describes under what conditions the receiver (i.e. CapCom) will ask for the data (see the precondition row of Figure 7-14). This is considered the question-part of the policy. In normal circumstance the receiver never has to ask for the data, since the schedule states when the lunar surface astronauts are to communicate. However, this might not always happen, in which case CapCom will ask for it. The condition under which the receiver asks for the data is when the data has a high priority (i.e. "1") and the receiver knows that the sender is already done with the activity after which he is supposed to communicate the data, and the data has not yet been received. The RP1 is implemented by workframe *ReceiverPolicy1* that specifies the policy conditions and executes a request for the voice data as a voice-loop communication activity (Figure 7-17).

```
workframe ReceiverPolicy1 {
    repeat:false;
    variables:
        forone(VoiceData) voice_data;
        forone(Activity) act;
        forone(LunarSurfaceAstronaut) fromagt;

    when (knownval(voice_data.duringAct = act) and
          knownval(act.isDone = true) and
          knownval(voice_data.fromAgent = fromagt) and
          knownval(fromagt.subActivity = act) and
          knownval(voice_data.pri = 1) and
          not(voice_data.received = true))
        do {
            RequestForData(100, 1, start, voice_data, act, fromagt);
        }
}
```

Figure 7-17. Workframe ReceiverPolicy1

When the receiver asks the sender for the data, the sender, if possible, will reply by executing one of the two SP's. This works as follows.

If the sender does not communicate the voice-data when he needs to according to the schedule, it means that the sender has forgotten about it. In the model this happens when the agent does not have the beliefs for the specific voice-data conceptual object. Not having any beliefs about a specific voice-data object implies that neither of the sender policies can be executed. Thus, when the receiver determines that he has not yet received the voice-data that he was supposed to receive, he executes the *RequestForData* communication activity in the *ReceiverPolicy1* workframe. This activity communicates all the necessary beliefs for the voice-data object (see Figure 7-18). When the sender agent receives the beliefs from the request for voice-data, it triggers one of the sender policies, which makes him communicate the voice-data.

```

communicate RequestForData(int pri, int dur, symbol whn, VoiceData vd, Activity act,
                           LunarSurfaceAstronaut fromagt)
{
    priority: pri;
    max_duration: dur;
    with: LmComCircuit;
    about: send(vd.duringAct = act),
           send(vd.fromAgent = fromagt),
           send(vd.toAgent = current),
           send(vd.pri = anyvalue),
           send(vd.received = false),
           send(vd.sent = false),
           send(vd.whn = anyvalue);
    when: whn;
}

```

Figure 7-18. RequestForData Activity

Notice that it will be possible for the agent to have forgotten or simply not to know specific beliefs associated with the voice-data. Such an occurrence would represent partial forgetting. For example, agent CharlieDuke might remember that he needs to communicate the start of the borehole drilling, but has forgotten its priority. In that case, there could be an additional policy that would handle this.

This concludes the description of the Q&A policy and its implementation in the Brahms model. Next, I describe how situated-errors can be handled in a general way, without having to specify every error condition and situation in advance.

7.3 DEVIATIONS FROM NOMINAL PROCEDURES

The procedures for the HFE deployment are nominal, meaning that these only hold as long as everything goes according to plan. In practice this almost never happens. The issue is thus, what will happen when something goes wrong, or not according to plan? The question we need to answer is how we can represent how agents will deal with problem situations, without having to describe every possible situation that can occur.

7.3.1 Problematic situations

The notion of human error is often cited as the cause for mistakes and disasters. What constitutes a human error is often left to the interpretation of the accident investigator. Usually, a deviation from a standard or nominal procedure is identified as an “error.” However, when we consider the situation-specific issues and the work practice in contrast to the procedures, often human error lies in the design and validation process of the procedures. Very often the procedures do not take into account the context and the situation in which the activities take place. Were the right people involved in the planning process? How were interactions between subsystems tested? Did organizational/functional breakdown prevent interactions between activities, materials and the situation?

The following “human error” occurred during the Apollo 16 deployment of the HFE. The LMP was in the process of drilling a hole in the lunar surface to implant the first HFE probe. He had connected the HFE package to the Central Station (CS) with a flatbed cable. At the same time, the CDR was busy deploying the Passive Seismic Experiment (PSE) in close proximity to the C/S. All this was planned and trained and had very detailed procedures (see Figure 7-3). Unfortunately, although known at the time, the procedures and training did not include the fact that the flatbed cables would not lay flat on the lunar surface due to the minimal lunar gravity. For example, the procedures did not include specific instructions on how to avoid getting tangled in one of the cables—it was very difficult for the astronaut to see his feet through the visor of his helmet. Consequently, the cable connecting the HFE to the C/S hooked on one of the CDR’s boots without the CDR noticing it, thus ripping the cable of the C/S and breaking the connection, making the Apollo16 HFE probes unusable.

If we consider the CDR’s specific situation, the procedures and his training displaying itself through the work practice of the astronauts, it is obvious that we should not call this an astronaut error. The CDR’s actions were not an intentional deviation from the nominal procedures. It was the situation specific context on the

moon that showed the error in the designs and procedures, as well as the lack of a developed work practice on the moon. Procedure designers and HFE engineers did not take this into account.

Nominal procedures can never capture the intricacies of work practice. One of the benefits of modeling and simulating not just the nominal procedures—but also the work practice of how the procedures are put into action, including the effects of the environment, communication, tools and artifacts, and error conditions—is that the design of the activities and interaction of the agents with each other and the environment can be more detailed. Thus, specifying contextual procedures and detailing how activities will be performed in reality will lower the chance of unplanned situations causing problems. Note that without considering these issues, Brahms models could incorporate the same kinds of failures. Therefore, a critical engineering framework is required, by which we include systematical failure analysis of past designs (of which the HFE is one example).

The next two sections describe types of activities that I have been able to identify and generalize from data of the Apollo 15 and 16 missions. I then describe an activity design that implements a *general* error-recovery activity that is used by the lunar surface astronaut agents to recover from both types of errors, without having to specify the error situations up front.

7.3.2 Two types of error-recovery

As briefly described in the previous section, the error during the Apollo 16 HFE Deployment broke the experiment. This meant that the HFE drilling activity the LMP was working on was not useful anymore. Once the LMP had determined this, he decided to stop the activity and start another activity by reordering the scheduled plan. This type of behavior is what I refer to as *individual error-recovery*. In other words, in individual error-recovery the individual recovers from a situated error and decides what to do next.

The second type of error-recovery behavior was observed during the Apollo 15 HFE deployment. I refer to this type as *distributed error-recovery*. During the Apollo 15 HFE drilling activity the CDR could not get the ALSD off of the first bore stem he had just drilled into the surface. He was supposed to use his foot to stop the bore stem from turning, while he was turning the ALSD counter clockwise, to get it off the bore stem. Whatever the CDR tried to do, the bore stem would turn with the ALSD, and consequently the drill would not come loose. After awhile the CDR gave up and asked CapCom what he should do, thus letting the people at mission control solve the problem. After a few minutes the CapCom gave the CDR a new procedure using a wrench to keep the bore stem from turning with the drill. The essence of *distributed error-recovery* is that the individual relies on others to help in an error-recovery activity. The problem solving is distributed over a number of individuals.

7.3.3 Error-recovery activity design

How do we model error-recovery activities without describing every potential error situation? The design of the error-recovery activity is divided into two pieces. The first objective of the design is to provide a default error-recovery practice the lunar surface astronauts will follow when they detect an error and they have no specific practice for resolving a particular error. Secondly, the design should allow a modeler to add more specific error-recovery activities as part of the work practice of each planned activities.

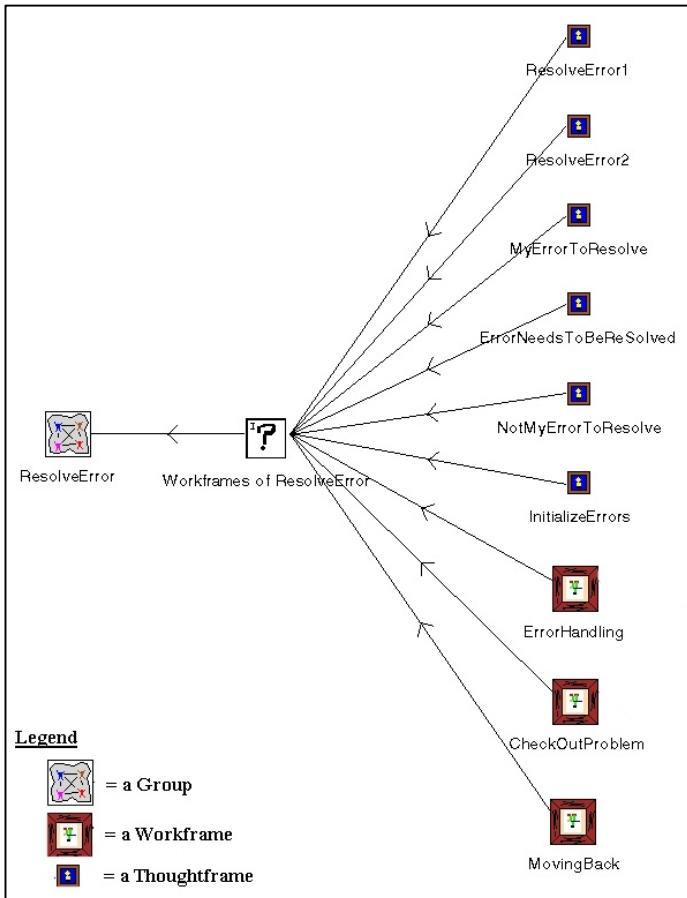


Figure 7-19. Default error-recovery activity workframes and thoughtframes (generated by the old Brahms simulation G2 engine)

After it has been determined that the error needs to be resolved, the agent then needs to determine if the error needs to be resolved by him (*MyErrorToResolve* or *NotMyErrorToResolve* workframes). When the agent has decided that it is responsible for solving the error, it now has to figure out if it can start solving the error at that moment. There are two possibilities; if there is no previous error the agent can go ahead and resolve the one at hand (*ResolveError1* thoughtframe). If there is a previous error that the agent was supposed to solve, but he is at the moment not working on that particular error, then he can go ahead and start solving the one at hand (*ResolveError2* thoughtframe). When the agent decides to resolve the error, he will start the *CheckOutProblem* workframe. Next, the astronaut will perform the *ErrorHandling* workframe. Last, the *MovingBack* workframe allows the agent to move back to the location he was at before he started checking out the problem.

The *CheckOutProblem* workframe has two activities (Figure 7-20). During the performance of this workframe, the thoughtframes in Figure 7-19 can fire. The *Moving* activity lets the agent move to the location where the error occurred. The *CheckOutProblem* composite activity has just one workframe called *InvestigateProblem*. During the investigate activity the agent can detect the facts that represent the problem that occurred; i.e. `detect(error.errorCode = value)`.

Figure 7-19 shows the design of the *ResolveError* group. This group is a subgroup of the *LunarSurfaceAstronauts* group, and implements the default error-recovery activity that every lunar surface astronaut can perform during error situations. The figure is read from left to right. It shows the group with all its workframes and thoughtframes. The design of this default activity is based on how the Apollo 15 and 16 astronaut resolved the errors that occurred during the HFE deployment. First, the astronaut goes over to the location where the error has occurred to check out the problem (*CheckOutProblem* workframe). This is a workframe that the astronauts will always perform in case of an error situation that affects their activities. During the *CheckOutProblem* activity any of the thoughtframes can fire, based on the situation. First, the agent needs to determine if the error needs to be resolved or not (*ErrorNeedsToBeResolved* workframe).

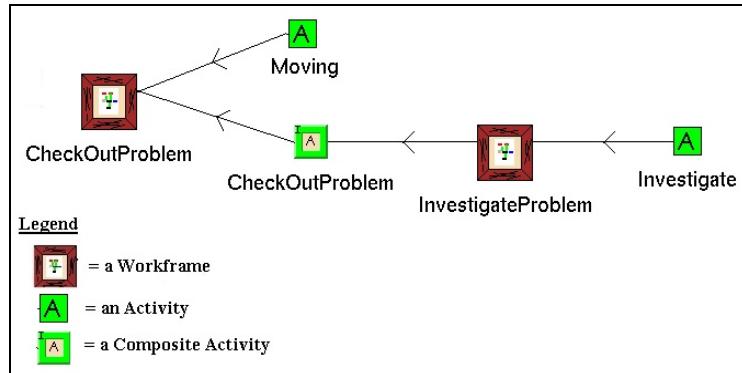


Figure 7-20. CheckOutProblem workframe & activity hierarchy

Now that the agent knows the type of error that has occurred, it can always handle the error using a error handling activity. This is done through the implementation of the *HandleError* activity. Figure 7-21 shows the default *HandleError* activity implementation in the *ResolveError* group. The *ErrorHandling* workframe gets executed when there is no other more specific *HandleError* activity available. The effect of this default activity is that the agent will ask the CapCom (effectively mission control) how to solve the error. Then the agent will wait until the CapCom comes back with a solution. This default behavior is an implementation of the default *distributed error-recovery* approach.

When the CapCom communicates back the solution from mission control, the agent can execute the thoughtframes in Figure 7-21. The two options that have been implemented as default behavior are that a) the solution from mission control is to continue with the activity and ignore the error (*ContinueCurrentActivity* thoughtframe), and b) the solution from mission control is to start working on another activity, effectively abandoning the current activity (*StartNewActivity* thoughtframe). In the second case the CapCom communicates what activity to work on next, thus effectively changing the astronaut's plan.

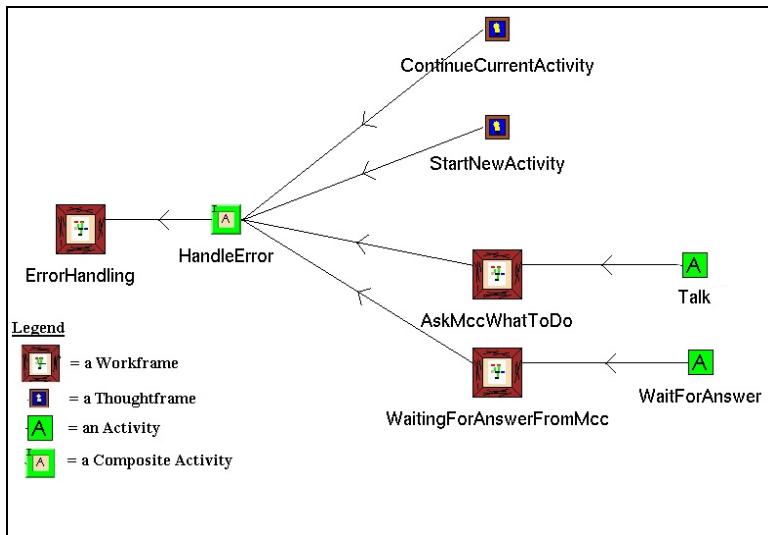


Figure 7-21. ErrorHandling workframe & activity hierarchy

7.4 PURPOSE OF GENERAL REPRESENTATIONS

The objective of this experiment is to determine if with Brahms we can develop a *predictive model* of a work practice. As described in the introduction of this chapter, prediction in this context means that the model can predict what will happen during the Apollo HFE Deployment in not previously specified situations in terms of the activities of the lunar surface agents, as well as the communication of voice-data. The focus of the prediction of future scenarios is on the order in which astronauts perform activities, their communication of the scheduled voice-data, and on the situated error-recovery activities that are performed when a not previously modeled error situation occurs.

The ALSEP Offload model of the Apollo 12 mission (see chapter 6) was not predicting new communications patterns or showing how the agents would recover from error situations. The model was hard-wired in terms of speech acts by the lunar surface astronaut agents, and no error-recovery procedure was part of the model. The purpose of the design and implementation of the question and answer policies and the error-recovery procedure described in the previous two sections is to allow for a more general and predictive model of practice.

By abstracting the default practice of the Apollo astronauts dealing with error situation, I was able to develop a more general applicable model. The model cannot only show the behavior of the Apollo 16 astronauts, but the model is able to predict the behavior of Apollo lunar surface astronauts in situations that were not modeled. More over, we could add more specific error-recovery behavior, based on observable errors in previous missions. Doing this would make the model even more applicable in predicting the behavior in future error events. Such capability would be useful in designing work practice simulations for future missions to the Moon that are based on the Apollo missions from the past. I have shown that we can implement more general models in Brahms, which will allow us to move to the use of Brahms in design. However, in the next section, I first present the V&V of this general model of HFE deployment.

7.5 VERIFICATION AND VALIDATION

This section describes the V&V of the model developed in this experiment. The V&V process used in this experiment is similar to the process described in the first experiment (section 6.9). Again, we need to verify the data used and the conceptual model developed based on this data, as well as the implementation of the model in Brahms source code, and the black-box validation of the experiment.

7.5.1 Data validation

As in the previous experiment, the data used in this experiment are all original NASA records of the actual Apollo missions. Table 7-1 lists all data sources that have been used in this experiment.

Table 7-1. Data sources used during experiment

Data Source	Data Type
Apollo 16 Final Lunar Surface Procedures	Original official Apollo 16 mission procedures
Apollo Lunar Surface Journal	Transcriptions of actual astronaut voice loop recordings + mission photographs.
Apollo 15 & 16 Video Tapes	Video Recordings of the actual Apollo missions from NASA.

The conceptual model is mostly based on the Apollo 16 Final Lunar Surface Procedures (Kain et al. 1972). This is the official mission procedures document, and thus contains an accurate pre-mission plan of the activities, communications, and timelines. Besides the pre-mission data, there are also mission voice-transcriptions and video data. Together these provide a very accurate data set upon which the models can be based.

7.5.2 Conceptual model validation

The validation of the conceptual model is easier in this experiment than in the previous one. This is because I developed the conceptual model based on the HFE procedures as described in the Apollo 16 Final Lunar Surface Procedures. Validation in this respect means that we need to make sure that all the activities from the procedures are indeed specified in the model. The model comprises the hierarchy of activities shown in Figure 7-4. The named activities are based upon and cross-referenced with the detailed HFE activity timeline in the procedures. Besides basing the activity model on the lunar surface procedures, I also verified that the specified activities can be matched with the Apollo 15 and 16 voice transcriptions in the ALSJ (Jones 1997), as well as video data available for both missions.

7.5.3 Computer model verification

The Brahms model is a generic HFE Deployment model. It does not describe the work practice of a single Apollo mission from the past. Instead, it is an idealized model of the lunar surface astronaut activities based on the nominal HFE Deployment procedures. The objective of the model is to represent the procedures in such a way that it can be used to simulate past and possible future HFE deployments on the moon, by representing the nominal procedures at a work practice level.

Before we can validate the predictive nature of the model, we need to verify that the model simulates the nominal procedures correctly. To do this we split the verification process in two steps. In step one, we verify that the model shows the performance of the HFE deployment in the order in which the nominal procedure, given in the Final Lunar Surface Procedures (FLSP), tells us that it should be done. In step two, we verify that the voice-data that needs to be communicated is communicated in the nominal simulation, according to the voice-data schedule given in the FLSP.

7.5.3.1 Nominal procedure verification

The order of the high-level HFE deployment activities (Figure 7-2) is:

0. Place ALSEP Packages on the Lunar Surface (this is the pre-HFE deployment activity)
1. Prepare the HFE equipment for deployment
2. Deploy the first HFE Probe.
3. Deploy the second HFE Probe.

What we need to verify is that a) the order in which the lunar surface astronaut agent performs the HFE deployment is consistent with the FLSP activity order, and b) that each high-level activity is correctly performed, based on the detailed FLSP.

ALSEP Package Placement

Before the astronaut can start with the HFE deployment, he first needs to place the packages on the lunar surface in the correct location, based on the mission specific ALSEP deployment area configuration (see Figure 7-1 for Apollo 16's configuration). This is considered the *first activity*.

We verify that this happens in our simulation. Figure 7-22 shows LMP agent CharlieDuke reading his first activity to perform from the LmpCuffChecklist object contained on his EMU suit. The beliefs the agent gets from this “reading” activity triggers him to start the AlsepPackagePlacement activity, described in Figure 7-23. It shows that the model correctly simulates the start of the first activity, because the name of the first activity is provided as input data to the agent on its cuff checklist, instead of being “hardwired” in the model.

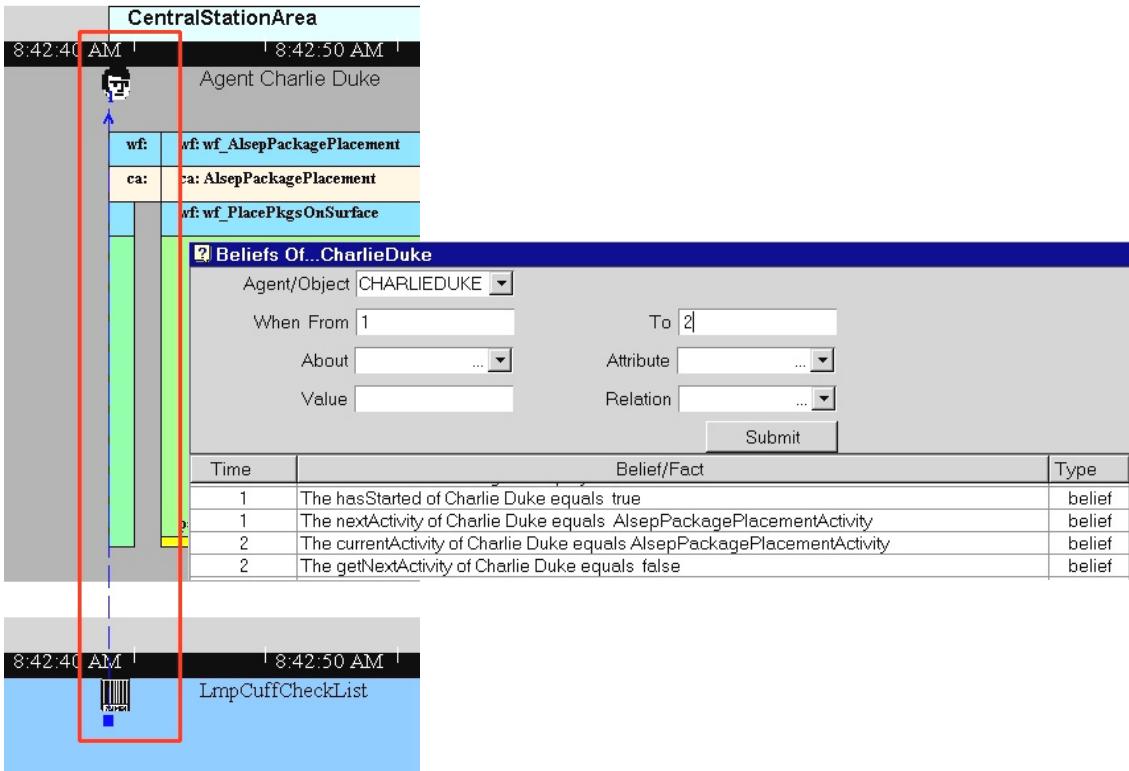


Figure 7-22. Reading the first activity

Figure 7-23 shows the nominal procedure of the ALSEP Package Placement Activity. The astronaut to perform that activity is the LMP. The first sub-activity is to place the ALSEP packages on the surface. Since the LMP carries the packages from the LM to the deployment site, and this is the first activity on arrival at the deployment site, the LMP is carrying the antenna-mast with the two packages attached to it. Placing the ALSEP packages on the surface means that the astronaut has to put the antenna-mast on the surface, and then remove each of the two packages off of the mast. This is done at the C/S site (the site of C/S package). After ALSEP Pkg1 (includes C/S) is on the surface, Pkg2 includes the RTG (RTG package) is removed from the antenna-mast and carried to the RTG area (i.e. 8' West of Pkg1 in the case of Apollo 16).

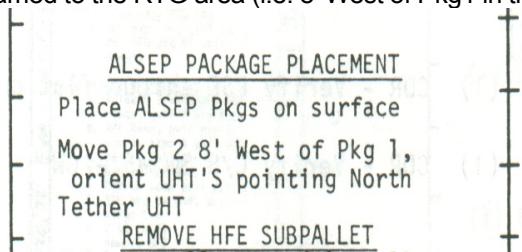


Figure 7-23. ALSEP Package Placement Activity

Figure 7-24 shows the simulation of this activity. After the agent has determined its first activity from the plan, it performs the sub-activities from Figure 7-23 in the correct order. First the agent places the ALSEP packages on the surface (*PlacePkgsOnSurface* activity). You can see that during this activity the agent first moves the AntennaMast object to the surface (in the *CentralStationArea*), and at the end of the activity the *CentralStation* object (ALSEP Pkg1) has also been moved to the surface.

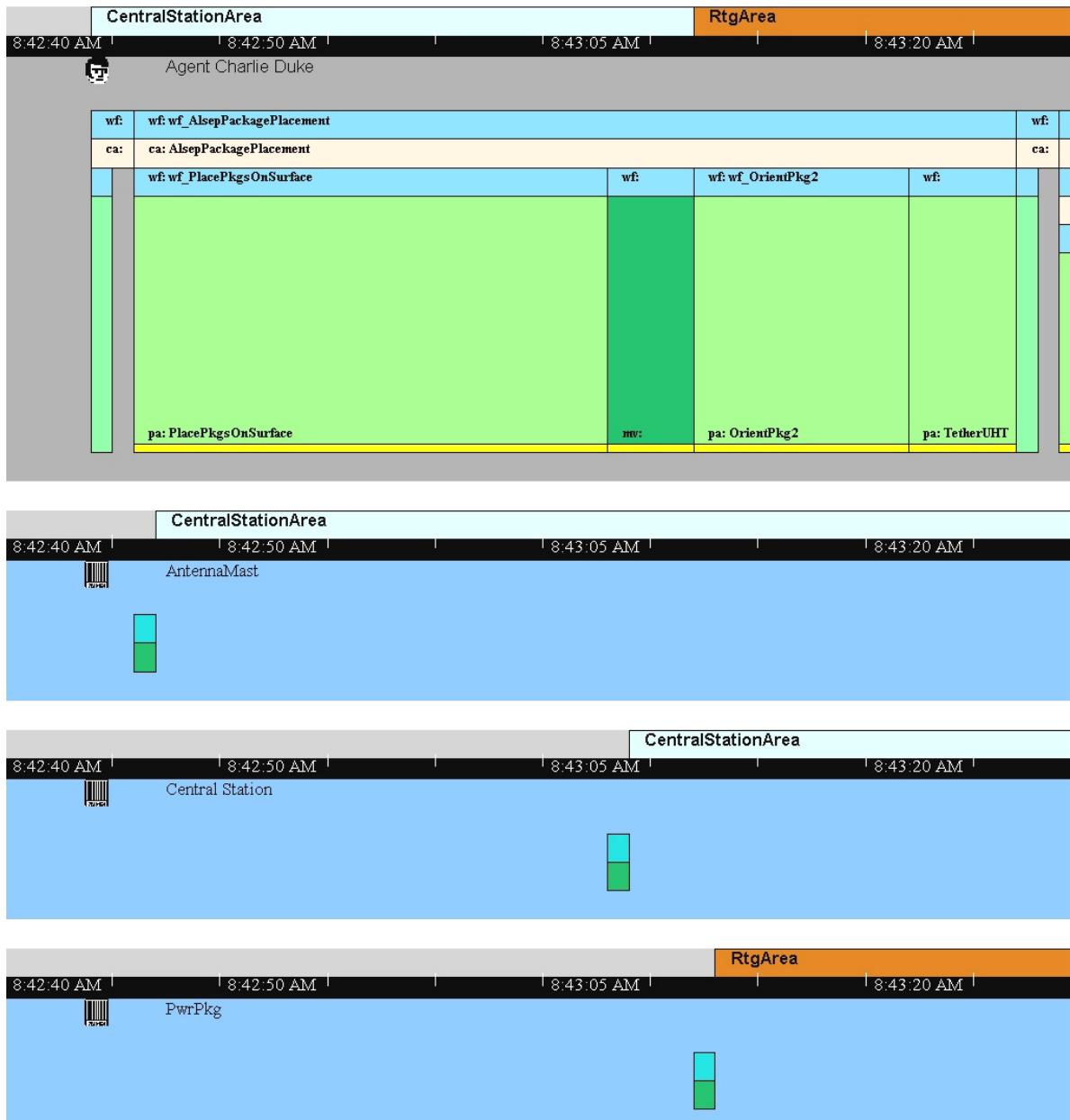


Figure 7-24. ALSEP Package Placement Activity Verification

One might ask how it is represented that the agent would first perform the *PlacePkgsOnSurface* activity. Will the agent always perform this activity? Always performing this activity would obviously not be accurate. The agent only needs to perform this activity if he is in the correct location and is carrying the antenna-mast with the two ALSEP packages attached to it. Figure 7-25 shows how this is accomplished in the preconditions of the workframe. The question is also how the agent moves the antenna-mast and the packages to the lunar surface? This is done in the body of the workframe. First, the *AntennaMast* object is moved to the ground. The consequence

```
conclude((AntennaMast.moveToGround = true), bc:100, fc:100);
```

in Figure 7-25 creates the belief for the agent, but also creates a fact in the world. The *AntennaMast* object reacts to this fact and executes the *MoveToGround* activity, which moves the object to the ground (i.e.

moves it to the same location as the agent currently containing it). The detectable makes sure that when the AntennaMast has moved to the ground, the agent detects that the object is not contained anymore.⁵⁴

```

workframe wf_PlacePkgsOnSurface {
    detectables:
        detectable DetectAntennaMastRelease {
            when (whenever) detect((current contains AntennaMast is false));
        }
    when (knownval(current.location = CentralStationArea) and
          knownval(current contains AntennaMast) and
          knownval(AntennaMast contains AlsepPkg1) and
          knownval(AntennaMast contains AlsepPkg2))
        do {
            conclude((AntennaMast.moveToGround = true), bc:100, fc:100);
            PlacePkgsOnSurface( );
            conclude((AlsepPkg1.moveToGround = true), bc:100, fc:100);
            conclude((AntennaMast contains AlsepPkg1 is false), bc:100, fc:0); // don't create fact!
            conclude((AntennaMast contains AlsepPkg2 is false), bc:100, fc:100);
            conclude((current contains AlsepPkg2), bc:100, fc:100);
        }
}

```

Figure 7-25. PlacePkgsOnSurface Workframe

After the AntennaMast object is on the ground, the agent takes the *AlsepPkg1* off of the AntennaMast, and then moves it to the ground. This is represented by the consequence after the *PlacePkgsOnSurface* activity

```
conclude((AlsepPkg1.moveToGround = true), bc:100, fc:100);
```

and moves the *AlsepPkg1* object to the ground at the end of the activity. Both these object movements can be seen in Figure 7-24.

Next, the agent moves to the *RtgArea*, carrying the *PwrPkg* object (*AlsepPkg2*), which is therefore also moved to the *RtgArea*. When the agent arrives at the *RtgArea* it puts the *PwrPkg* object on the ground. After this, the agent performs the orientation activity (*OrientPkg2*) and then tethers the universal handling tool object (*UHT*). At the far right side of Figure 7-24 you can see that the agent is determining its next activity by again reading its next activity from the cuff-checklist object.

Representing work practice means more than the sequential execution of activities according to the procedure. It is at minimum necessary that we represent under what situational condition activities are performed, e.g. people won't put objects down that they are not carrying. Although it would take a lot of space to show all the situational preconditions for each workframe, every workframe contains those situational conditions that would make the agent execute the activities within it. Each activity can and will only be executed if the "correct" situational context exists in the world, and the agent is aware of it (i.e. has beliefs of this context). Effectively, this makes the model able to predict what activities the agent will execute next, based on the situational events (modeled by facts) that happen in the world.

⁵⁴ Modeling putting objects on the ground in this way is needed because the Brahms language does not contain a built-in "put" and "get" activity. Such an activity should be added to the language.

HFE Equipment Preparation

The next high-level activity to be verified is the *HfeEquipmentPreparation* activity (see Figure 7-2). This activity is decomposed into three other high-level activities from the Final Lunar Surface Procedures; *Remove HFE Sub Pallet*, *Deploy HFE*, and *Deploy ALSD* (see Figure 7-3, number 1 on the left-hand side and Figure 7-4).

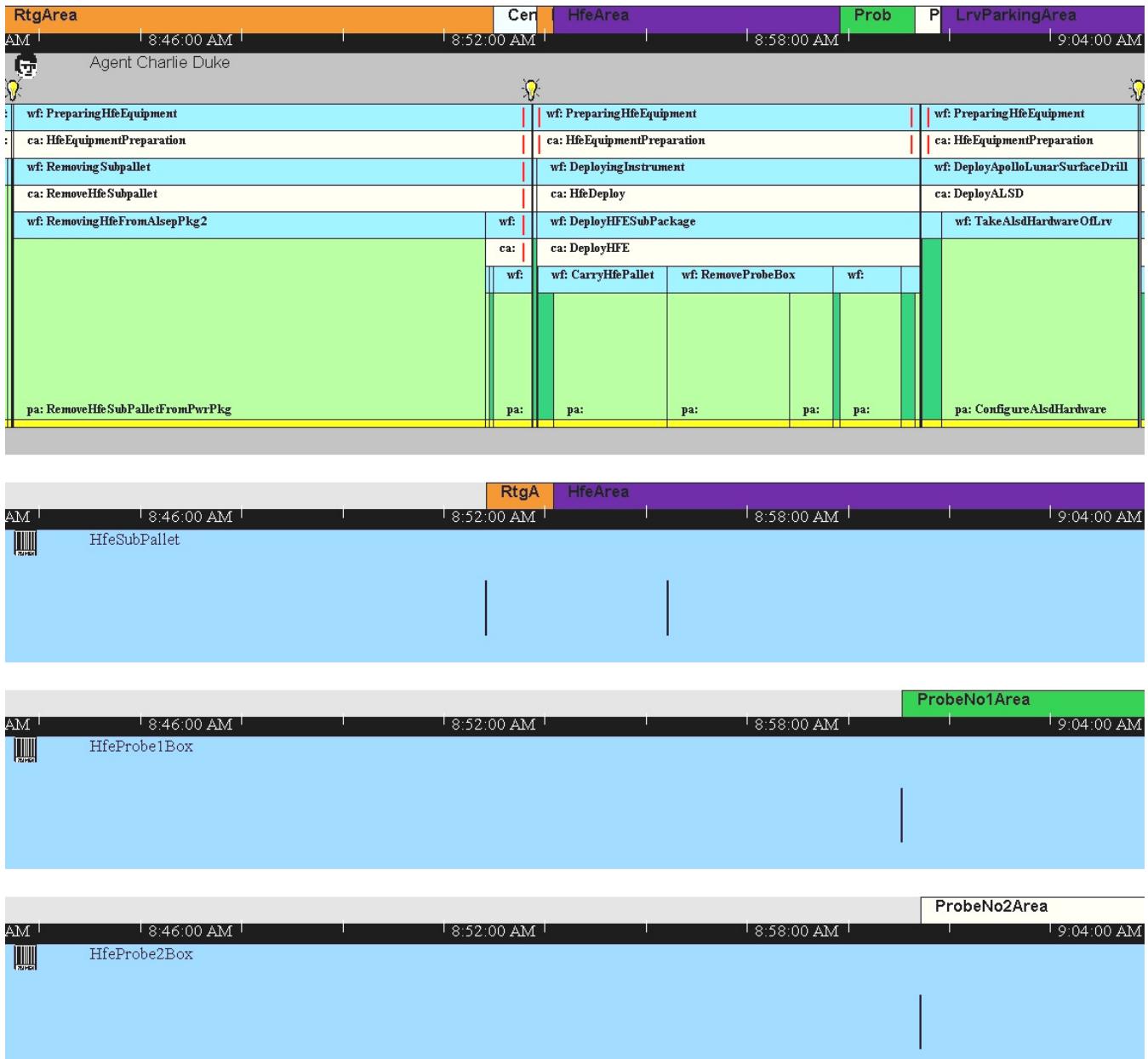


Figure 7-26. HfeEquipmentPreparation Activity Verification

All the way to the left on Figure 7-26 we can see the thoughtframe being fired to read the next activity (shown by the light bulb right underneath the agent icon). Next, the HFE equipment preparation activity is performed. During the HFE equipment preparation activity the agent carries the necessary objects (*HfeSubPallet*, *HfeProbe1Box* and *HfeProbe2Box*) to the appropriate locations. Due to the limited space available, the zoom-level in Figure 7-26 is such that some of the workframe and activity names cannot be displayed, however it has been verified that they are all correctly executed according to the procedure. Figure 7-26 shows again that the agent is not simply executing the activities, but is also interacting with objects in the environment and moving them to the correct locations.

HFE Probe No. 1

By far the longest and most complex activity is the *DeployingHfeProbe* activity. This activity is performed twice, once for deploying HFE Probe No.1 and again for HFE Probe No.2. Figure 7-27 verifies the activity for deploying the first probe. Again, the full procedure for this activity is shown in Figure 7-3 (numbers 2 and 3) and Figure 7-4.

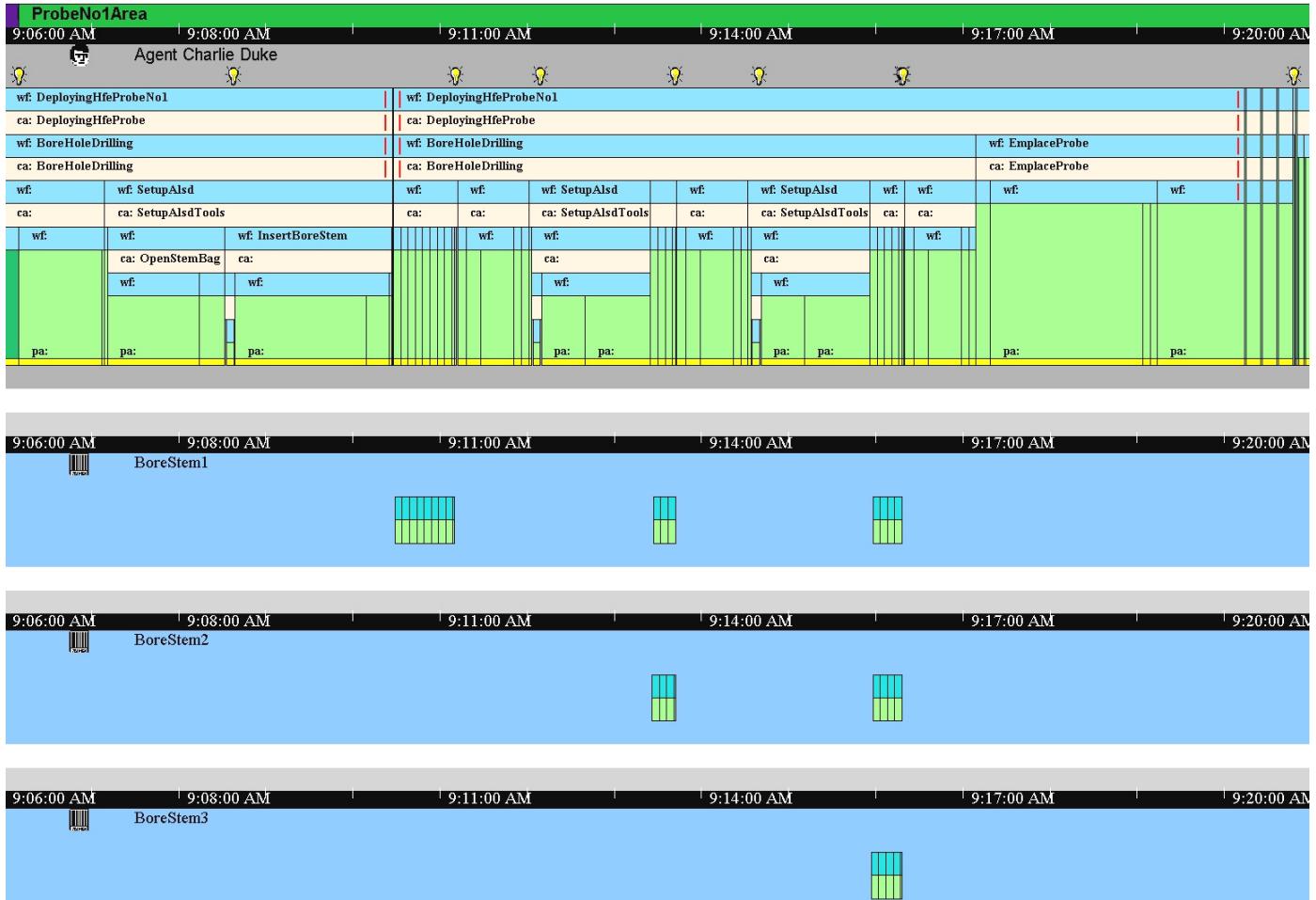


Figure 7-27. DeployingHfe/2Probe Activity Verification

There is a lot that can be said about the implementation and workings of this activity. However, the most interesting aspect in this activity is how the agent actually drills the boreholes into the lunar surface, using the Apollo Lunar Surface Drill (ALSD). This is modeled in the *Bore Hole Drilling* activity. The execution and duration of this activity is determined by the time it takes to drill the bore stems into the surface. For each borehole the agent has to drill three bore stem objects into the surface. A bore stem cannot be drilled into the surface, unless the previous stem is drilled sufficiently deep in the lunar crust. The start of drilling the next bore stem is therefore dependent on the astronaut's ability to drill the previous stem first. The agent will execute the *DrillingBoreStem* workframe and perform the *EnergizeDrill* activity, until the bore stem being drilled into the ground is all the way into the surface. This means that there is an interaction between the agent and the ALSD object, and the ALSD object and the Stem objects that are being drilled.

The agent first picks a bore stem from the bore stem bag, and inserts the drill onto the stem. Then, when the agent energizes the drill (i.e. starts drilling), the ALSD object starts drilling the stem objects connected to drill into the ground. The duration of the drilling activity is dependent on the length of the bore stem being drilled into the ground and the speed at which the stem is going into the ground (the speed of drilling is an average of five inches every five seconds). The stem objects themselves simulate this. Each stem object has a length, plus the workframe *DrillingIntoSurface* that simulates moving the bore stem into the lunar surface, as

long as it is contained by the *ALSD* and the *ALSD* is energized. It does this by calculating how deep the current bore stem is in the surface, using the following consequence

```
conclude((current.inchesIntoSurface = current.inchesIntoSurface + 5), bc:0, fc:100);
```

Every five seconds the stem moves five inches deeper into the ground. The stem object continues moving itself into the surface as long as the *ALSD* is energized. Moreover, the agent energizes the *ALSD*, as long as the bore stem is not all the way into the surface. Consequently, this is an interaction between the agent, the *ALSD* object and the bore stem object, as shown in Figure 7-28.

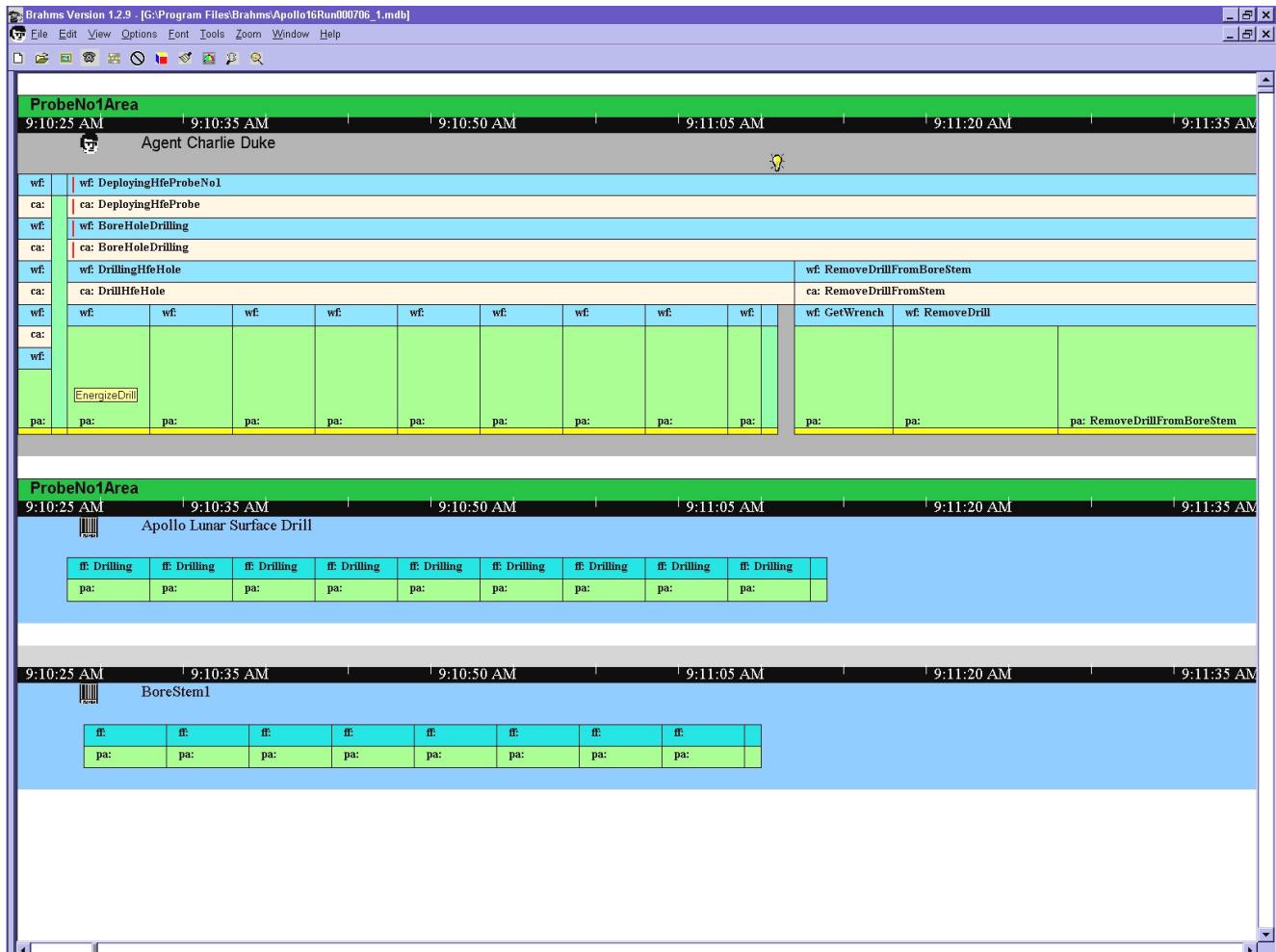


Figure 7-28. Drilling BoreStem1 in the surface

It should be noted that when a bore stem is drilled into the surface and the next bore stem is attached to it, all the bore stems that are attached to each other will be moving deeper into the surface with every drilling cycle of the *ALSD*. This phenomenon is shown in Figure 7-29 when bore stem objects *BoreStem1* and *BoreStem2* are both being moved into the surface at the same time. Since the bore stems have different lengths, the time it takes to drill each one of them into the surface differs.

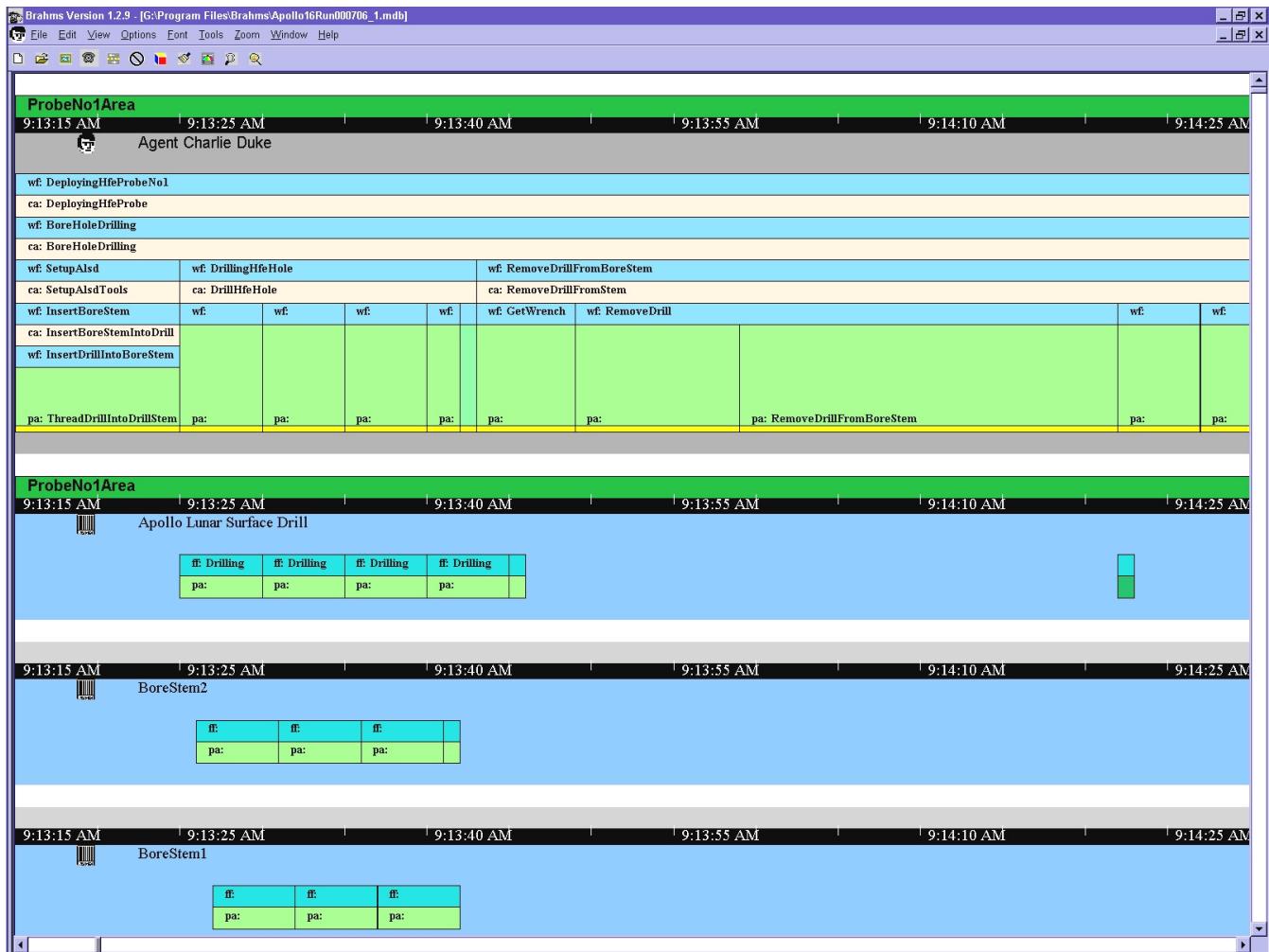


Figure 7-29. Drilling BoreStem2 in the surface

It should be noted that if the agent fails to drill one bore stem all the way into the surface, he will not be able to drill the other ones either, and thus will not finish the drilling activity. This is an important feature of the model for being able to model errors, which is the subject of the black-box validation (section 7.5.4 - 7.5.6).

7.5.3.2 Voice-data communication verification

The second step in the computer model verification is to verify that the voice-data communications happen as they are prescribed in the procedures (Figure 7-11 and Figure 7-13). Figure 7-30 is the verification that the sender conversation policy described in section 7.2.2 works correctly.

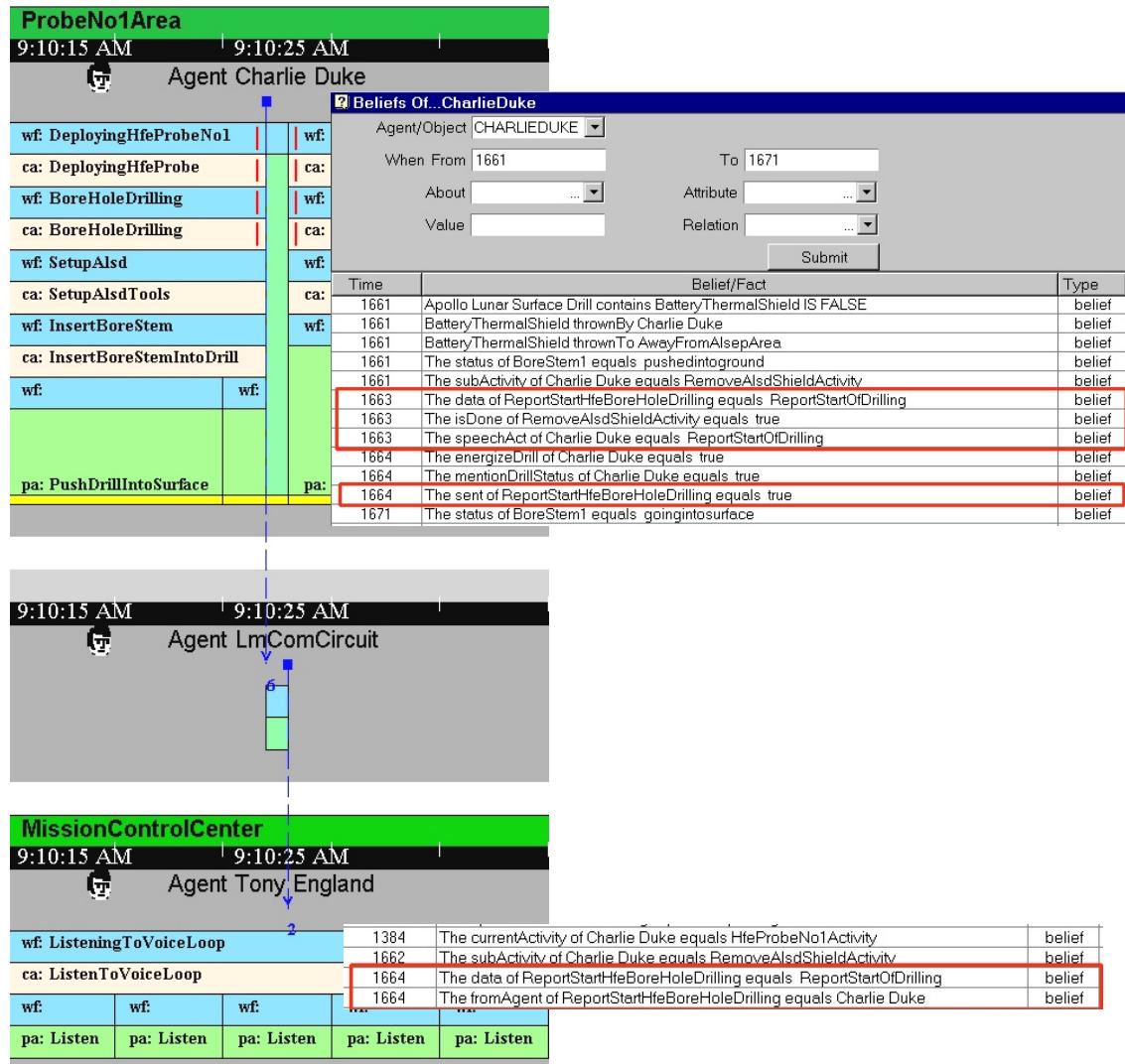


Figure 7-30. Conversation Policy Verification

Figure 7-30 shows that right after the interrupted *DeployingHfeProbeNo1* activity, agent *CharlieDuke* communicates the *ReportStartOfDrilling* data to agent *TonyEngland* specified by the *ReportStartHfeBoreHoleDrilling* voice-data object. A second later, through the voice-loop delay, agent *TonyEngland* has received the voice-data, as well as the belief about which agent communicated the voice-data.

To end the nominal model verification, Figure 7-31 shows the complete simulation of the HFE deployment, including the voice-data communication. The LMP agent *CharlieDuke* first prepares the HFE equipment in the RTG area. He then deploys the HFE and the ALSD in the HFE area and the LRV parking area respectively. Then, the agent deploys HFE probe 1 and 2 by drilling three bore stems for each into the surface. All the while communicating the voice-data to the CapCom agent *TonyEngland*. Figure 7-31 is the emerging activity performance and communication, based on the FLSP.

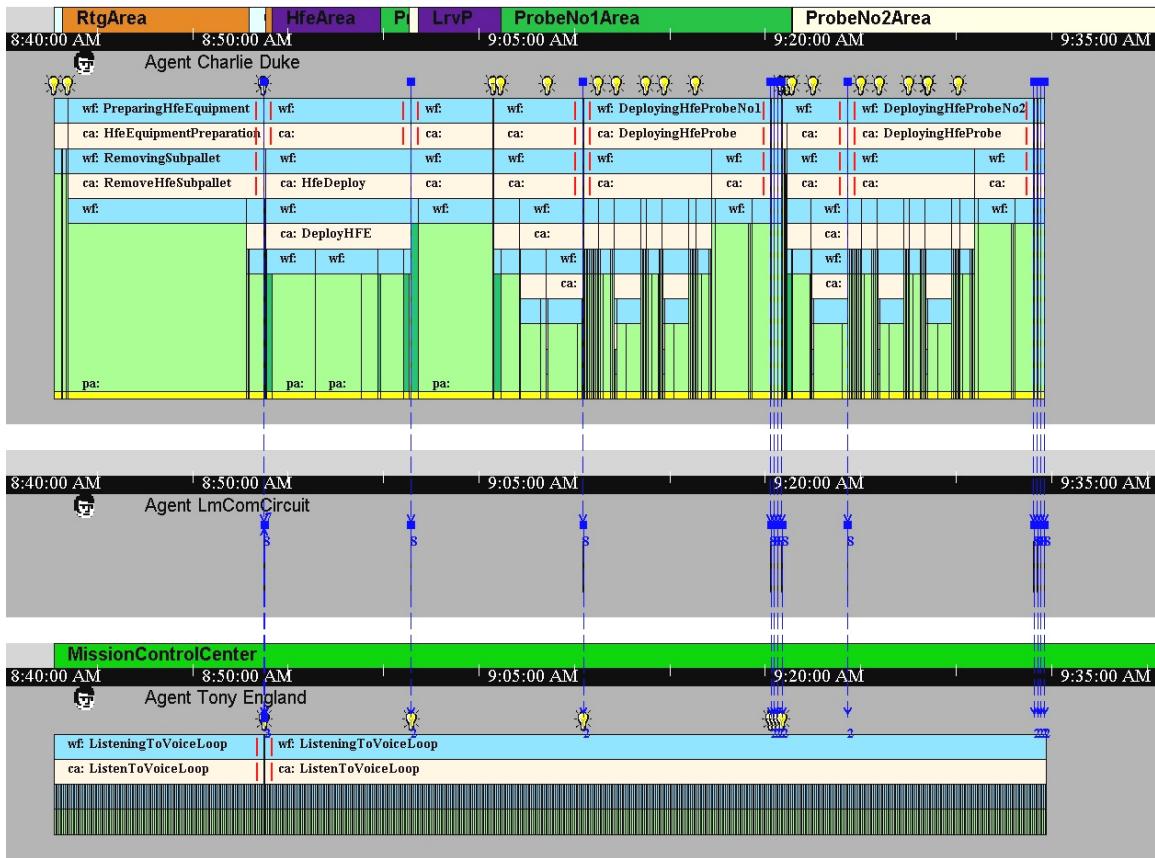


Figure 7-31. Voice-Data Communication Verification

7.5.4 Experiment validation

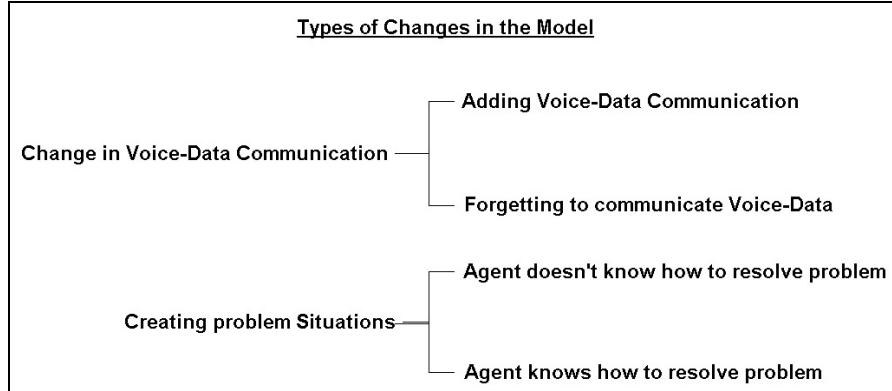
Validation is the determination of the degree of accuracy of a measuring device.⁵⁵ In this case the “measuring device” is the Brahms model of the Apollo HFE deployment procedures. What could this device be measuring? In other words, what are we validating?

In the previous section we have verified that the model simulates the nominal procedures for deploying the HFE experiment on the Moon. Meaning, we have established the truth about the accuracy of the model. Thus, we can say with a certain degree of confidence that the model is a representation of the nominal procedures at the work practice level, of the lunar surface astronauts performing the HFE deployment task. However, showing that this could be done in Brahms was the subject of the first experiment. In this experiment we want to go a step further and ask the question; if we have a model of work practice that simulates how astronauts on the Moon deploy the HFE, can we use this model to *predict* what will happen if the situation is different from the nominal procedures? The ultimate question in this section is thus whether the model is a “measuring device” for the work activities of the astronaut performing the HFE deployment in situations never previously described in the model. In other words, can we use the model to predict how the astronaut will behave in unanticipated situations, i.e. situations beyond the nominal case?

One of the main reasons for using simulation of any system is to predict the future behavior of the system, especially if we want to understand the impact of changes to a system before they happen. What is being validated here is whether the nature of the modeling—the way we built the model—allows us to use Brahms in developing predictive models that are based on representations at the work practice level. We want to understand what the impact will be on the activity performance of the lunar surface astronaut deploying the HFE, if the situation changes from the nominal procedures. I will show this by running different simulations in which I introduce a specific type of change event into the simulation—a scenario—and evaluate the outcome.

⁵⁵ Miriam-Webster's Collegiate Dictionary

We could try to give an exhaustive list of situations that could change the nominal procedures (types of changes). However, besides being impossible, this is overkill in relation to the purpose of this experiment. The purpose is to show that with Brahms we can make predictions about the work practice. Showing the model can handle one type of change would be too limited of a sample to draw conclusions. Being able to show more than one type would increase the validity, and thus two types of changes to the nominal procedures are worked out. The types of changes to the nominal model that can be introduced are:



7.5.5 Validating changes in voice-data communication

7.5.5.1 Adding voice-data communication

Adding voice-data communication to the model allows the simulation to predict *when* agents will communicate the voice-data during the performance of their activities. Adding voice-data requires the modeler to add a voice-data object to the model. There are two ways of changing the model. First, we could add a voice-data object statically, i.e. before the simulation starts. In a sense, this is like changing the voice-data communication procedures for the agent. This is obviously possible, because that is how the nominal voice-data communication is simulated. However, more interestingly we could assume that the agent who is to communicate the voice-data has no pre-specified procedure for this. What will happen if another agent, say the CapCom agent asks the HFE deployment agent for specific information not previously part of the procedures?

Scenario

When the LMP starts drilling the first bore stem into the surface, mission control wants to know when the bore stem is all the way into the surface. Therefore, the CapCom asks the LMP to give an “end mark” for the drilling of the borehole.

We change the model as follows: we add a workframe to the CapCom group that will make the CapCom ask the LMP for this drilling end mark voice-data (Figure 7-32).

```

workframe wf_AskForDrillingEndMark {
    repeat:false;
    when (knownval(ReportStartHfeBoreHoleDrilling.data = ReportStartOfDrilling))
    do {
        conclude((ReportDrillingEndMark.duringAct = EnergizeDrillActivity), bc:100, fc:0);
        conclude((ReportDrillingEndMark.fromAgent = CharlieDuke), bc:100, fc:0);
        conclude((ReportDrillingEndMark.pri = 1), bc:100, fc:0);
        conclude((ReportDrillingEndMark.received = false), bc:10, fc:0);
        conclude((ReportDrillingEndMark.sent = false), bc:100, fc:0);
        conclude((ReportDrillingEndMark.toAgent = current), bc:100, fc:0);
        conclude((ReportDrillingEndMark.whn = end), bc:100, fc:0);
        AskForDrillingEndMark();
    }
}

```

Figure 7-32. AskForDrillingEndMark workframe of CapCom agent

The precondition of the *wf_AskForDrillingEndMark* (Figure 7-32) states that when the agent gets the belief that the drilling of the borehole has started, the agent will ask agent CharlieDuke to report the drilling end mark (i.e., the moment the bore stem is all the way into the lunar surface).

Figure 7-33 shows that immediately after the LMP agent tells the CapCom agent that he will start the drilling, the CapCom agent asks the LMP agent to communicate when he is done with drilling the bore stem. This communication happens at the end of the *DrillHfeHole* activity. Note, that the LMP agent's model did not change from the nominal model of the procedures. The simulation predicts that the LMP agent will answer the question of the CapCom agent for Voice-Data, as well as when the LMP agent will give the answer.

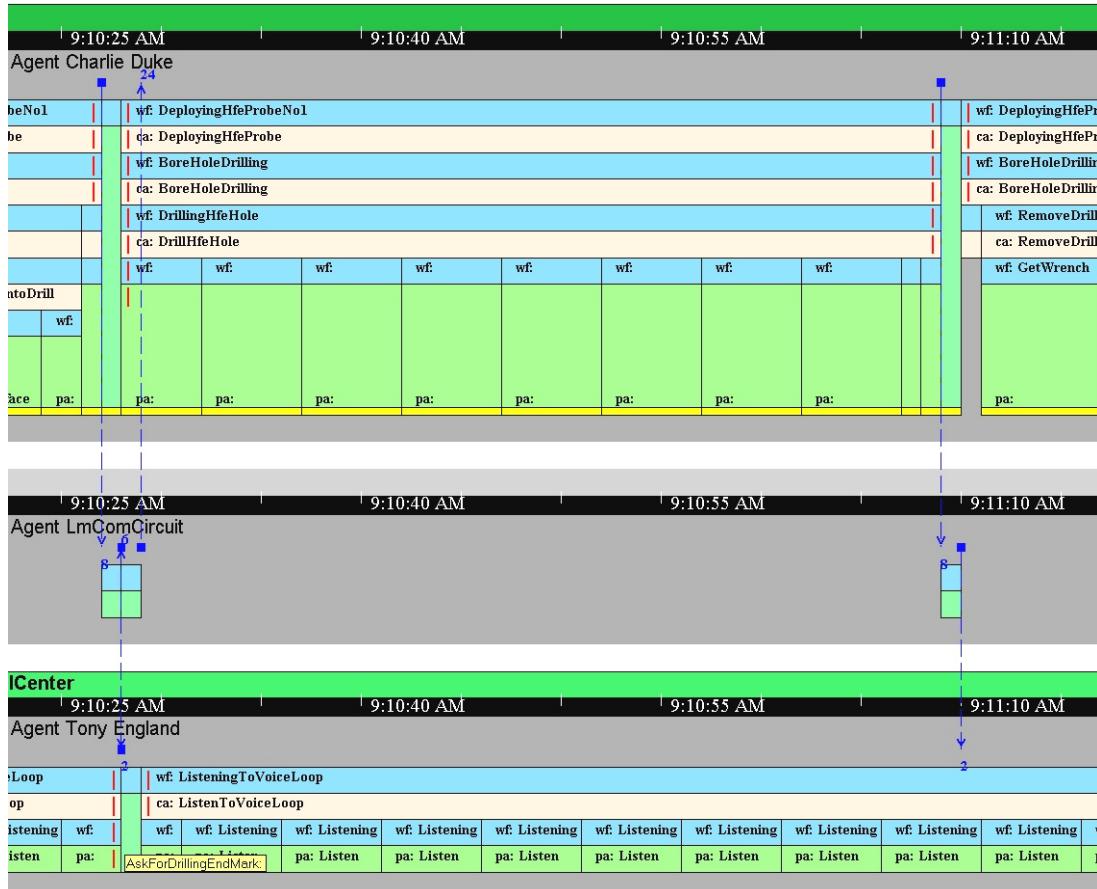


Figure 7-33. Asking For Voice-Data

7.5.5.2 Forgetting to communicate voice-data

The Q&A conversation policy includes a policy for the receiver to ask for voice-data (as shown in the previous scenario). If the sender does not communicate the pre-specified voice-data communication, the receiver is to ask for it. Using this approach the model is able to predict when the CapCom agent will ask for voice-data in situations where the LMP forgets to communicate it.

Scenario

Let's assume the LMP forgets to communicate the first scheduled voice-data communication (i.e. reporting that the RTG cable is connected to C/S).

We change the model as follows: remove the LMP agent's (the sender's) initial-beliefs about the *ReportHfeCableConnectedToCs* voice-data. This creates the situation that the agent *forgets* to communicate the voice-data.

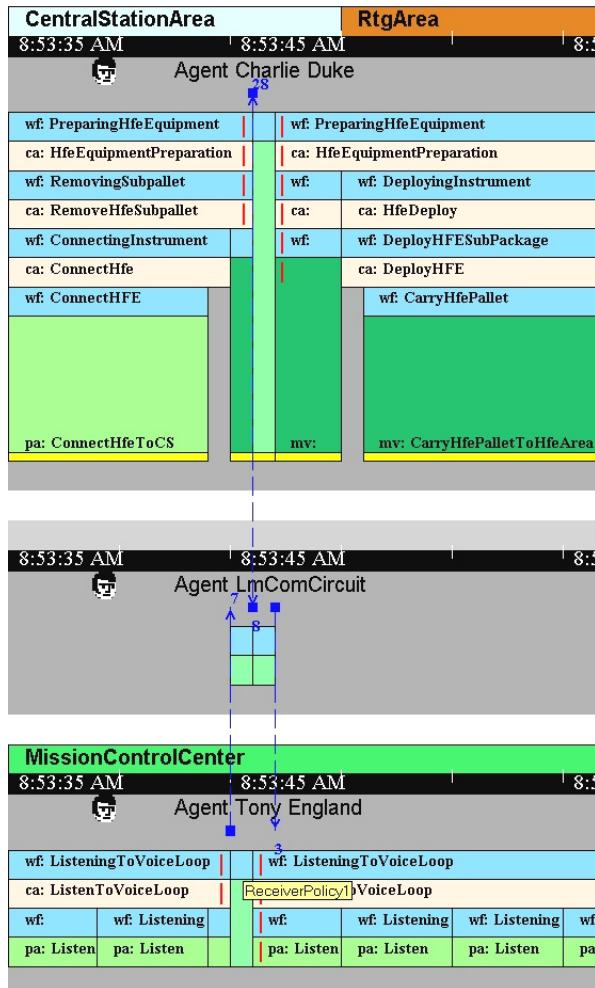


Figure 7-34. Asking for Voice-Data

Figure 7-34 shows that after the LMP agent has finished connecting the HFE cable to the C/S, and is starting to move to the RtgArea to deploy the HFE, the CapCom agent realizes that the LMP agent did not communicate that the HFE cable was connected to the C/S. Therefore, the CapCom agent executes the *ReceivePolicy1* workframe to ask the LMP for confirmation. The LMP agent replies with the confirmation of this voice-data. Thus, the simulation predicts a question and answer conversation about voice-data between the CapCom agent and the LMP agent, in case the lunar surface agent forgets to communicate the voice-data. This was validated against the voice transcription of the Apollo 15 and 16 missions.

7.5.6 Validating creating problem situations

As described in section 7.3, the model includes an error-recovery procedure. The agent has a general procedure for dealing with unknown situations that cannot be handled by nominal procedures. The corrective behavior of people in a work setting is all about their ability to handle situations that occur and are not described by the nominal procedures. This is part of human ability that is difficult to duplicate by machines. If we want to model and simulate human work practices, we need to be able to simulate this human ability in our models, in order to predict what will happen to the activity-behavior of people in unplanned situations. I will describe the result of two types of scenarios that show how the model is able to predict the agent's activity-behavior when unplanned situations occur. The first scenario deals with a situation the agent does *not* know how to handle. Consequently, the agent falls back on the *default* error-recovery procedure described in section 7.3.3. The second scenario deals with a situation that the agent *does* know how to handle.

7.5.6.1 Agent does not know how to solve problem

The default error-recovery procedure (section 7.3.3) describes how a distributed team can handle a situation. This procedure, used by the lunar surface astronauts was very simple: when you're stuck and don't know what to do next, ask mission control what to do. The following scenario is being modeled and validated against the mission data.

Scenario

Let's assume that after the LMP astronaut has carried the ALSD tools from the LRV to the first HFE probe area, and he is setting the tools down on the lunar surface, something happens to the bore core stem bag that creates a problem situation. When the astronaut has no method to solve this problem, he asks CapCom what to do.

We change the model as follows: during the activity *SetAldToolsOnSurface* an error event needs to be generated that simulates the problem occurring. Having the *BoreCoreStemBag* object create an Error object simulates this.



Figure 7-35. Default error-recovery procedure

Figure 7-35 shows both the generation of the error event and the performance of the default error-recovery procedure by the LMP and the CapCom. At time 8:43:07 AM the BoreCoreStemBag object creates the HfeDeploymentError object (Figure 7-35), and transfers (communicates) the error information in the form of beliefs to the object (shown in Figure 7-36). This in effect is the generation of the error-event. The error object contains event-specific information about the error. The information that is generated is that an error has occurred, where the error has occurred, and an error code that specifies the severity of the error.

AGENTorOBJECT	StartTime	DISPLAYTEXT
HfeDeploymentError	24	BELV: HfeDeploymentError hasOccurredIn HfeProbeNo1Activity
HfeDeploymentError	24	BELV: The errorCode of HfeDeploymentError equals notserious
HfeDeploymentError	24	BELV: The hasOccurred of HfeDeploymentError equals true

Figure 7-36. Error Object Data

The LMP detects the error immediately, and after deciding that the error is an error he has to resolve (this is represented by the four thoughtframes that are shown in Figure 7-35) he starts the CheckOutProblem activity described in section 7.3.3, Figure 7-20⁵⁶. After checking out the problem and because the agent does not have a specific error procedure for this type of error, the agent starts the default ErrorHandling activity, described in section 7.3.3, Figure 7-21. In this default *ErrorHandling* activity the LMP asks the CapCom for help in solving the problem. The CapCom decides what activity the LMP needs to perform next. This is what the CapCom communicates back to the LMP over the voice-loop (Figure 7-37).

⁵⁶ Figure 7-35 does not show the name of the *CheckOutProblem* workframe and activity due to the font size and lack of space.

DisplayText	Agent-or-FocusO	StartTime	DisplayText	Agent-or-FocusObject
BELV: The speechAct of Tony England equals HfeProbeNo1Activity	LmComCircuit	42	SendComToLm:	Charlie Duke

Figure 7-37. CapCom's error-recovery decision

Because the ErrorCode for the Error object is “not serious,” the CapCom has decided that the LMP should continue with the deployment of the first HFE probe (i.e. the HfeProbeNo1Acitivity in Figure 7-37). The LMP performs the WaitForAnswer activity until the CapCom has communicated this decision. At that point the LMP determines (via a thoughtframe) that he needs to continue with the HfeProbeNo1Activity. He continues the DeployingHfeProbe composite activity, and keeps setting the ALSD tools on the lunar surface.

This error handling activity is a general activity, and would work for both lunar surface astronauts in any of their activities for any type of error that affects their activity performance. This is because the error object describes (in terms of the beliefs it “encodes”) what activity it impacts (*((error hasOccurredIn activity))*). The downside of this approach is that it is not the astronaut agent determining whether the error impacts his activity, it is the modeler creating the error event. What the model predicts is how the astronaut agent behaves when such an error occurs. To make the determining of the error impact also predictive, we need to represent cognitive error diagnosis abilities for the agent, based on the type of error and its impact on the activity.

7.5.6.2 Agent knows how to solve problem

The previous section verified the model's default error-handling procedure. Asking mission control what to do is a safe default procedure, because mission control has more information, and more people to help in finding the best solution in a given situation. However, this is not the fastest procedure in all cases. EVA time is resource bounded, and time is critical. Therefore, it makes sense to have the astronaut be able to make critical decisions when he can. The question is how we can model such behavior in a generic way such that the astronaut will use his decision procedure in a situation when there is one, and if not, will fall back on the default error-handling procedure. This is the topic of this section.

Example Scenario from Apollo16

While the LMP is emplacing the first HFE probe into the borehole that has been drilled and is communicating the voice-data about the depth of the probe, et cetera., the CDR is working at the C/S taking off a sub-package and bringing it to another site. While the CDR is walking away from the C/S, his foot gets caught behind the HFE cable connected to the C/S, and he breaks the cable at the connector to the C/S. At this point the LMP decides how to solve this unrecoverable error.

```
workframe CreateHfeCableError {
    repeat:false;
    when (knownval(HfeCable.isConnectedTo current) and
          knownval(CharlieDuke.location = ProbeNo1Area) and
          knownval(EmplaceProbeActivity.isDone = true) and
          knownval(CharlieDuke.currentActivity = HfeProbeNo2Activity))
    do {
        conclude((HfeDeploymentError.hasOccurred = true), bc:100, fc:100);
        conclude((HfeDeploymentError hasOccurredIn HfeProbeNo2Activity), bc:100, fc:100);
        conclude((HfeDeploymentError.occurredInLocation = CentralStationArea), bc:100, fc:100);
        conclude((HfeDeploymentError.errorCode = unrecoverable), bc:100, fc:100);
        Error(HfeDeploymentError, CentralStationArea, HfeProbeNo2Activity);
        conclude((current contains HfeCable is false), bc:0, fc:100);
    }
}
```

Figure 7-38. CreateHfeCableError workframe

We change the model as follows:

As in the previous validation, the model needs to generate the appropriate error-event at the appropriate time. In a complete model of both the lunar surface astronauts' work practices, we would include the behavior of the CDR agent as well. In that case, the CDR agent would trigger the error-event because the model would simulate him tripping over the HFE cable. In this limited model we leave out the behavior of the

CDR agent. We use an approach similar to the previous error-recovery validation (section 7.5.6.1). The *A1sepPkg1* object (i.e. the C/S) generates the *HfeDeploymentError* object with the appropriate error-event information, at the appropriate moment in time (Figure 7-38). The error-event is generated when the LMP is done with the *EmplaceProbe* activity, and is just starting the *HfeProbeNo2Activity* activity. Breaking the HFE cable cannot be fixed; thus the error is an unrecoverable error. The location where the error occurred is the location of the C/S (CentralStationArea).

Now that the correct error-event is being generated, the next change in the model is to add a situation specific error-handling activity that allows the agent to handle unrecoverable errors during the *DeployingHfe* activity. When there is an unrecoverable HFE error, the astronaut deploying the HFE should not continue deploying the HFE, because this would be a waste of time. This is the basis upon which we can develop an error-handling activity in which the astronaut can decide himself what to do next.

Polymorphic Activity-Behavior

Polymorphism is a feature of object-oriented programming in which it becomes possible to design and implement systems that are easily extensible (Cardelli and Wegner 1985). By including more specific classes and methods in a generic hierarchical class structure, only those parts of a program need modifications that require direct knowledge of the more specific behavior. In borrowing this concept for designing the ability to add more specific error-handling behavior, I have developed an extensible and easy modifiable *error-handling* activity design, based on the notion of polymorphism.

The idea behind the design is the following: we specify default activity-behavior for agents who need to be able to perform an error-handling activity in a group that represents the activity—in this case I called the group the *ResolveError* group. This group includes all the activities and workframes for being able to resolve errors in a default way. Agents who are members of this group can resolve errors. Then, if we need to make a more specific error-handling behavior for an agent, we add a more specific error-handling activity in a subgroup of the *ResolveError* group, and use a form of *activity overriding* to execute the correct error-handling activity.

Figure 7-39 shows the polymorphic design for the error-handling behavior. The *ResolveError* group represents the default error-handling behavior. The *ErrorHandling* workframe calls the *HandleError* activity (see Figure 7-21 and Figure 7-35) with a high priority of 50. In the *HfeDeployment* group a more specific error-handling procedure is defined in the more specific *HfeDeployment.HandleError()* activity. This activity is called in the *DeployingHfeProbeErrorHandling* workframe with a priority of 55.

When the error-handling activity needs to be performed, both workframes in the two groups can become available for an agent. Because of the higher priority activity in the more specialized lower-level workframe, the lower-level workframe will always have precedence over the more higher-level workframe. Interestingly, if for some reason the more specialized activity cannot find an appropriate solution, the higher-level activity will be executed, which in this case will always result in the default error-recovery behavior, i.e. asking mission control for help.

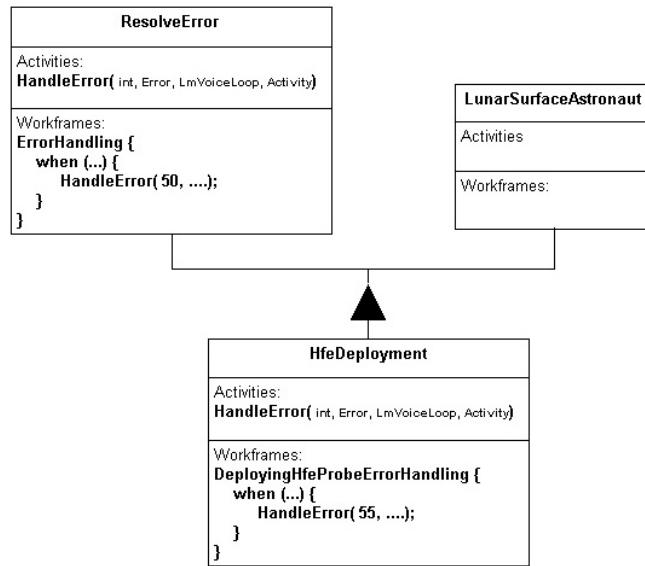


Figure 7-39. Polymorphic Error Handling Behavior

HfeDeployment HandleError Activity

Figure 7-21 shows the workframe-activity hierarchy for the default *HandleError* activity, whereas Figure 7-40 shows it for the more specific *HandleError* activity in the *HfeDeployment* group.

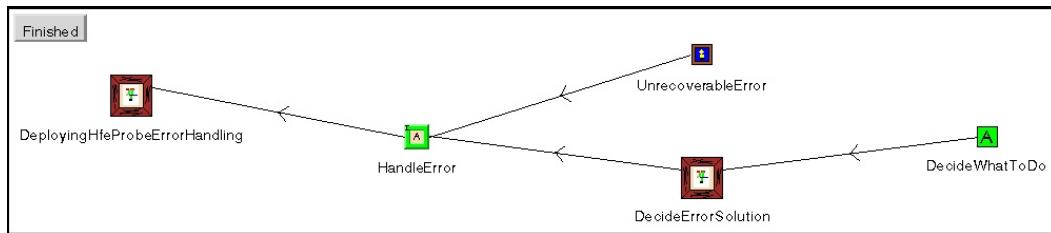


Figure 7-40. HfeDeployment group's HandleError Activity tree, generated by the Brahm's engine

In this more specific activity the agent decides on a solution to the error situation. The only decision he knows to make is when there is an *unrecoverable* error. In that case, while the agent is executing the *DecideWhatToDo* activity, the *UnrecoverableError* thoughtframe fires.

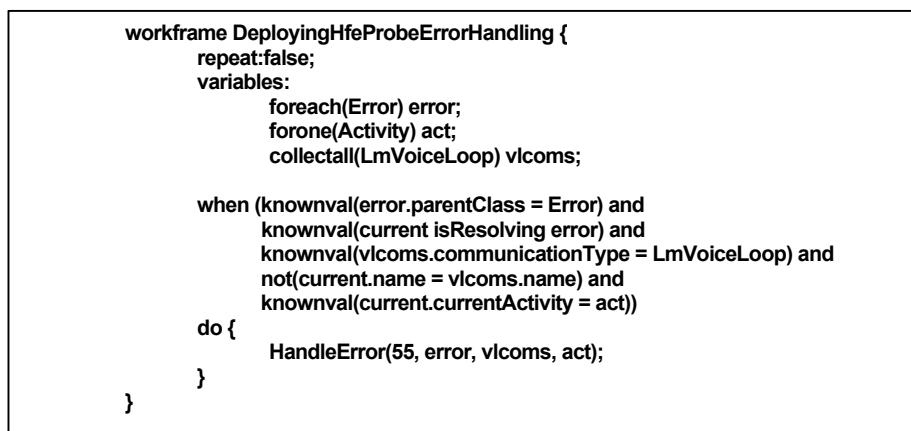


Figure 7-41. Workframe DeployingHfeProbeErrorHandling in the HfeDeployment Group

```

composite_activity HandleError(int pri, Error error, LmVoiceLoop vcoms, Activity act) {
    priority: pri;
    activities:
        primitive_activity DecideWhatToDo(int pri, int dur, Activity act) {
            priority: pri;
            max_duration: dur;
            resources: act;
        }
    workframes:
        workframe DecideErrorSolution {
            repeat: false;
            when (known(error.errorCode = value))
            do {
                DecideWhatToDo(0, 5, act);
            }
        }
    thoughtframes:
        thoughtframe UnrecoverableError {
            repeat: false;
            when (knownval(error.errorCode = unrecoverable))
            do {
                conclude((current.nextActivity = DrillCoreSampleActivity), bc:100, fc:0);
                conclude((current needsToResolve error is false), bc:100, fc:0);
                conclude((current isResolving error is false), bc:100, fc:0);
                conclude((error.status = resolved), bc:100, fc:0);
            }
        }
}

```

Figure 7-42. Activity HandleError in the HfeDeployment Group

Figure 7-41 and Figure 7-42 give the source code for the workframe that calls the *HandleError* activity, and the *HandleError* activity, respectively. As described above, it is the *UnrecoverableError* thoughtframe that allows the agent to decide what to do next. In case of an unrecoverable error situation during the deployment of a HFE probe, the agent will stop with the current activity, and will start the *DrillCoreSampleActivity* as his immediate next activity (see the thoughtframe in Figure 7-42);

```
conclude((current.nextActivity = DrillCoreSampleActivity), bc:100, fc:0);
```

How does the agent switch from deploying the HFE probe to drilling the core sample? This is accomplished using an *impasse* detectable. An impasse means that the workframe being impassed cannot continue until the impasse condition is lifted. In this scenario, the impasse condition should hold as long as the HFE deployment error is not corrected, and the agent cannot continue deploying the HFE probe. This is implemented with the *DetectErrorCondition* detectable in the *DeployingHfeProbeNo1* and *DeployingHfeProbeNo2* workframes. Figure 7-43 shows the detectable in the *DeployingHfeProbeNo2* workframe.

During the deployment of the HFE probes, as soon as the agent detects that an *HfeDeploymentError* error has occurred (see detectable in Figure 7-43):

```
detect((HfeDeploymentError.hasOccurred = true))
```

the current activity of the agent gets impassed, until,

```
(HfeDeploymentError.hasOccurred = false).
```

```

workframe DeployingHfeProbeNo2 {
    repeat:false;
    detectables:
        ....
        detectable DetectErrorCondition {
            when (whenever)
                detect((HfeDeploymentError.hasOccurred = true))
            then impasse;
        }
        ....
        when (knownval(current.nextActivity = HfeProbeNo2Activity))
        do {
            conclude((current.currentActivity = HfeProbeNo2Activity), bc:100, fc:100);
            PickUpAlsdHardware(0);
            DeployingHfeProbe(0);
        }
}

```

Figure 7-43. Workframe DeployingHfeProbeNo2

Simulation Result

Simulating the scenario results in the behavior is shown in Figure 7-44. When the HFE error occurs at 9:20:50 AM, the agent immediately detects the error and starts the default error-recovery procedure, impassing the current *PickUpAlsdHardware* activity (see Figure 7-19). The default error-recovery procedure is started. The first step is to check out the problem. To do this the agent moves to the location where the error occurred, i.e. at the *CentralStationArea*. Then, instead of performing the default *HandleError* activity, the agent performs the more specific *HandleError* activity in the *HfeDeployment* group. You can see the firing of the *UnrecoverableError* thoughtframe above the *HandleError* activity.

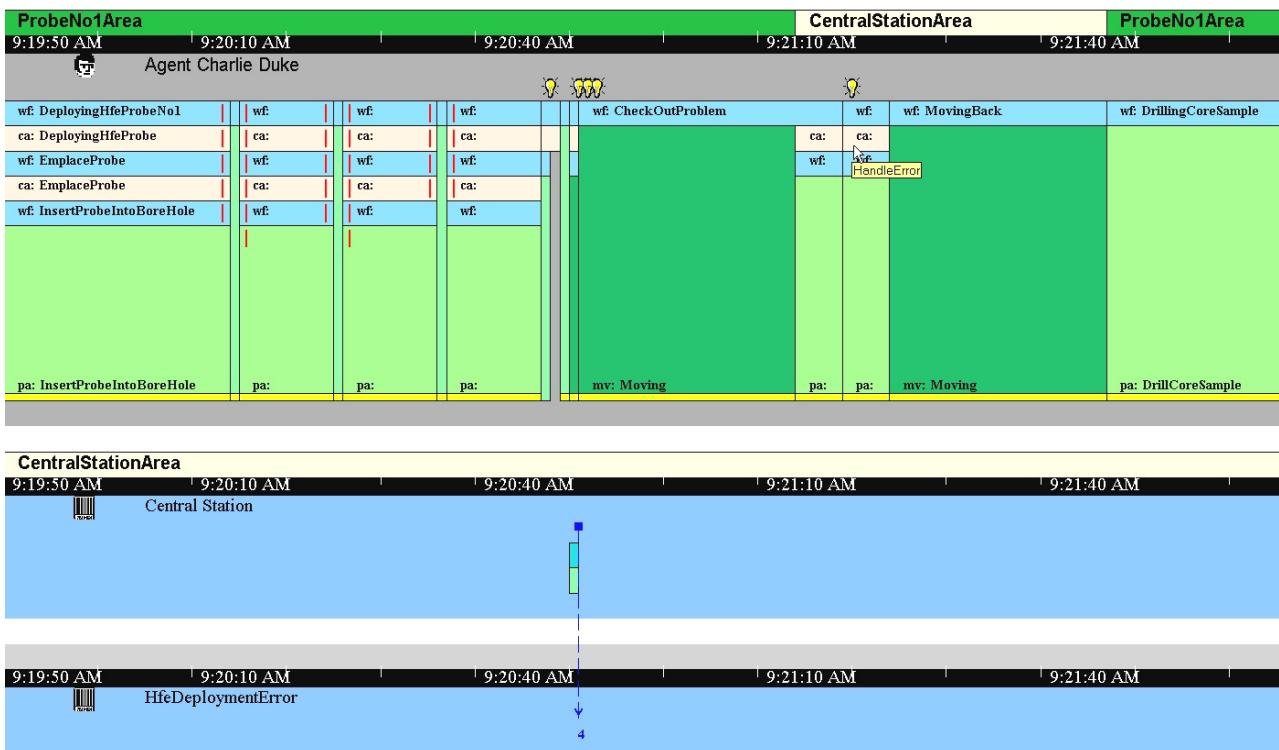


Figure 7-44. Simulating DeployingHfeProbe ErrorHandling Procedure

After the agent has made the decision, he continues with the default error-recovery procedure. The next activity is to move back to the area where the agent came from, i.e. *ProbeNo1Area*. Once he has arrived at that area, he starts the *DrillCoreSample* activity. The *DeployingHfeProbe* activity is impassed, and will stay impassed, because the error will not be resolved.

Validation of the simulation data

The simulation in Figure 7-44 is based on the real Apollo 16 error-scenario described previously. We can therefore validate the simulation result with the data from the Apollo 16 mission. Following is the voice transcription of the Apollo 16 mission from (Jones 1997). Although the specific timing from the real data and the simulation data is not exact, the model is a plausible representation of what transpired during the actual Apollo 16 mission.

121:21:21 Duke: Okay, (on) the second one: the thermal cover is in to the second red mark. And, Tony, the probe is out of the ground up to B-8. Right on the line between B-7 and B-8.

121:21:37 England: Okay; Baker 7 and 8. (Pause)

[While Charlie was talking, John lifted the mortar package off the top of the Central Station and headed around the east side toward the subpallet. As he went between the rock and the Central Station, he lifted his right foot to clear a cable - as he had done previously - but, just off-camera, his trailing left foot was caught in the heat-flow ribbon cable and, because of the tension from that cable, he stumbles slightly and, in the process, bends his right knee enough that his left knee almost touches the ground. As he brings his left leg forward, we see the heat-flow cable draped over the top of his foot. Fendell tilts the TV down as John steps forward with his right foot and then, as he makes a final step with his left, the cable pulls taught and tears loose from the base of the Central Station. John turns to his left and, as he hops backwards away from the Central Station, his left leg off the ground, he surveys the damage.]

121:21:45 Young: Charlie.

121:21:46 Duke: What?

121:21:47 Young: Something happened here.

121:21:48 Duke: What happened?

121:21:49 Young: I don't know. (Brief Pause) Here's a line that pulled loose. (Pause)

[John puts the mortar package down and goes to his right knee to try to get the end of the heat-flow cable. He rises without it.]

➔ **121:21:57** Duke: Uh-oh.

121:21:58 Young: What is that? What line is it?

[John kicks the end of the cable toward the rock and then gets down on both knees, steadying himself with his left hand on the rock, and picks up the severed end of the cable.]

➔ **121:22:02** Duke: That's the heat flow. You've pulled it off.

121:22:05 Young: I don't know how it happened. (Pause)

[John rises easily and merely has to step forward to get his feet under him. All of his movements are amazingly stable.]

121:22:11 Young: (Walking toward the Central Station) Pulled loose from there?

121:22:12 Duke: Yeah.

121:22:14 Young: God almighty. (Pause)

➔ **121:22:17** Duke: Well, I'm wasting my time.

*[John drops to his knees at the back of the Central Station and examines the connector. Photo AS16-[113-18348](#) (***) is a close-up of the end of the cable and the connector.]*

121:22:20 Young: I'm sorry. I didn't even know...I didn't even know it. (Pause) Agh; it's sure gone.

[John leans back to get his center-of-gravity over his feet, rises, and, this time, has to hop back a foot or two to get his balance.]

121:22:34 England: Did the wire or the connector come off?

121:22:36 Young: (lost under Tony) had our first catastrophe. It's broke right at the connector.

[John starts walking toward the mortar package; and Charlie comes into view, going to the Central Station to inspect the damage.]

→ 121:22:42 Duke: The wire came off at the connector.

121:22:45 England: Okay, we copy. (Pause)

[To get past the rock, John actually steps up on it and over. To my knowledge, this is the only record in Apollo of someone stepping up on something. The rock is approximately 20 centimeters tall and is equivalent to the rise of an ordinary household step.]

121:22:45 England: Okay, I guess we can forget the rest of that heat flow.

→ 121:22:55 Duke: Yeah, I'll go do the (deep core). (Pause) Oh, rats!

7.6 CONCLUSION

In conclusion I come back to the research question for this experiment, and discuss the findings. Simulation is an imitation of the operation of a real-world process or system over time. Whether done by hand or by computer, simulation generates an artificial history of the system and the observation of that history to draw conclusions about the operating characteristics of the real-world process or system (Banks et al. 1996). Predicting behavioral changes of a complex system is thus one of the important reasons for using simulation technology.

In this case study I used Brahms to simulate the procedures for the Apollo HFE deployment. The simulation model was used to predict the astronaut's behavior, based on the real Apollo 15 and 16 missions. The Brahms simulation generated a simulation history file that could be inspected using the Brahms AgentViewer program.

7.6.1 Experiment outcome discussion

The goal in this experiment is to determine whether using Brahms allows us to develop models of work practice that can predict the changes in human behavior and the impact on the work system based on contextual changes. To determine this, I implemented a Brahms model of the nominal work process of a lunar surface astronaut (the LMP) deploying the Heat Flow Experiment on the Moon (sections 7.1, 7.2, and 7.3).

First, I verified the Brahms model's simulation of the normative behavior of the astronaut, based on the detailed Apollo 16 Lunar Surface Procedures (sections 7.5.1, 7.5.2 and 7.5.3). Next, I defined a number of types of situational changes we could create as changes to the model (section 7.5.4). Then, I found a two scenarios from the Apollo 15 and 16 missions with which I tested these types of changes in separate simulation runs, to validate the predictive behavior of the model (sections 7.5.5 and 7.5.6). The outcome of this experiment is summarized in Table 7-2.

Table 7-2. Summary of Experiment Outcome

Predict	Operationalization	Model Input	Model Output
Nominal Activity Behavior	Apollo HFE deployment	Apollo 16 Final Lunar Surface Procedures for nominal HFE deployments	Prediction of nominal activity behavior during HFE deployment on the Moon.
Changes in Voice-Data Communication	Adding and forgetting of HFE voice-data communication from lunar surface to mission control	Changes to the Apollo 16 Final Lunar Surface schedule for HFE voice-data communication (additions and omissions)	Prediction of voice-data communication-activity behavior.
Error Recovery Behavior	Situated error events during HFE deployments	Error objects and generated error events	Prediction of individual and distributed error-recovery behavior

The validation shows that with the correct behavioral abstraction and modeling approach we can develop Brahms models that can be used to predict the collective behavior of agents in specific situations.

7.6.2 Modeling approaches discussion

The modeling approaches that have been developed during this experiment are useful for future predictive models. In the first experiment, the Apollo 12 ALSEP Offload, the model was developed with no generic features in mind, even though we were able to reuse part of the voice-loop communication model. The model could not simulate any other scenario than the Apollo 12 specific scenario. In the HFE Deployment model I focussed on making the model generic. This means that by changing some simple parameters in the input, I was able to simulate a different scenario. The following modeling techniques and approaches were developed, and could be used in the future for predictive models.

1. Parameterize the activity plan for agents—Section 7.2.1.

The agents do not have their activity plan hardwired in the preconditions, but instead have an external plan represented as beliefs in an artifact. This way the agent can dynamically determine the order of the activities, either by communicating with the artifact (such as reading a plan), inferring a new plan, or by detecting changes in the environment. Other agents are also able to communicate any changes in the plan.

2. Conversation policies for predicting agent communication—Section 7.2.2

I abstracted a standard process of asking and answering questions into conversation policies. Using this pragmatic modeling approach for defining how agents converse, based on transcriptions of the Apollo 15 and 16 mission, I was able to predict when and how question and answer conversations between agents take place. Again, in the Apollo 12 model the communications were hardwired in the body of workframes, and were very limited, because it does not allow for variations in the communication between agents.

3. Polymorphic activity design for specializing activity behavior—Section 7.5.6.2

Another interesting finding in this experiment is the use of polymorphism, supported by the Brahms language in designing default and more specialized activity behavior. The use of group membership and the inheritance of attributes, relations, initial beliefs and facts, as well as activities and workframes, allow for polymorphism. This is similar to polymorphism in object-oriented languages. In object-oriented languages polymorphism is used to specialize object type behavior. However, in modeling agent behavior,

polymorphism is used for a different purpose. The purpose here is to create default *and* specialized behavior. The lower in the group hierarchy a polymorphic activity is defined, the more specialized this behavior is. However, the specialized behavior is not overriding the default behavior, but is in addition to it. Therefore, the agent can perform both the default behavior, as well as the more specialized behavior. Using this approach we can make sure the agent is always able to act in any situation, by providing at least one default activity and more specialized activities for different situations.

This ends the second case study. Next, I describe the third and last case study, that of prescribing a new work system.

8. CASE STUDY 3: DESIGNING HUMAN-ROBOT COLLABORATION

In this chapter, I describe the third and last case study. This study is showing the use of Brahms as a tool for the design of a work system for a robotic mission to the Moon. Design is a prescriptive activity, and using Brahms as a design tool means that we are creating a prescriptive model, that is, a model of a work system that does not yet exist. At NASA this type of design activity falls under the rubric of Mission Operations Design (Wall and Ledbetter 1991). Currently, the people involved in designing mission operations for robotic missions use relatively impoverished tools for the task at hand. Consequently, the confidence in the prescriptive models is not high, with as a result that the designers include "slack" in their designs. For example, the complexity of spreadsheet models for the design of mission timelines and activities does not allow them to include all relevant variables into the analysis. Also, the lack of powerful modeling tools does not allow mission operation designers to investigate many scenarios that could be relevant. In a personal conversation with a Co-Investigator for the 2003 Mars Exploration Rovers ('03 MER) mission⁵⁷, I was told that due to a lack of powerful tools they were able to investigate only two of the original five mission scenarios. The reason for this was the complexity of the model and the time it took to create a scenario.

In this case study I investigate the use of Brahms for the design of mission timelines and activities, based on a richer model of the work system than is currently possible with the available tools. This case study shows that providing mission operation designers with more powerful prescriptive modeling tools will allow them to be more efficient and effective in the design of missions. This helps NASA with doing missions better, faster, and ultimately cheaper.

Robots and Humans as Partners

In the coming decades, the moon will also prove useful as a laboratory and test bed. Astronauts at a lunar base could operate observatories and study the local geology for clues to the history of the solar system. They could also use telepresence to explore the moon's inhospitable environment and learn how to mix human and robotic activities to meet their scientific goals.

The motives for exploration are both emotional and logical. The desire to probe new territory, to see what's over the hill, is a natural human impulse. This impulse also has a rational basis: by broadening the imagination and skills of the human species, exploration improves the chances of our long-term survival. Judicious use of robots and unmanned spacecraft can reduce the risk and increase the effectiveness of planetary exploration. But robots will never be replacements for people. Some scientists believe that artificial-intelligence software may enhance the capabilities of unmanned probes, but so far those capabilities fall far short of what is required for even the most rudimentary forms of field study.

To answer the question "Humans or robots?" one must first define the task. If space exploration is about going to new worlds and understanding the universe in ever increasing detail, then both robots and humans will be needed. The strengths of each partner make up for the other's weaknesses. To use only one technique is to deprive ourselves of the best of both worlds: the intelligence and flexibility of human participation and the beneficial use of robotic assistance. (Spudis 1999)

Mission operation design and planning for robotic and mixed human-robotic tasks is currently done quite informally with the design team's heuristic intuitions about tasks the agents (either human or robotic) need to do, and the likelihood of that capability being available in the future state of the art. This creates the fundamental problem where the analysis of the human and robotic collaborative elements of a mission is being carried out at a very high-level of abstraction, until well into the commitment for a design of the robot. In part this is a consequence of the inadequacy of current systems in allowing easy modeling of the intricacies of a rich and dynamic set of tasks being carried out by robots in conjunction with humans (Sims et al. 2000).

⁵⁷ <http://mars.jpl.nasa.gov/missions/future/2003.html>

The Melding of Mind and Machine

Human dexterity and intelligence are the prime requirements of field study. But is the physical presence of people really required? Telepresence—the remote projection of human abilities into a machine—may permit field study on other planets without the danger and logistical problems associated with human spaceflight. In telepresence the movements of a human operator on Earth are electronically transmitted to a robot that can reproduce the movements on another planet's surface. Visual and tactile information from the robot's sensors give the human operator the sensation of being present on the planet's surface, “inside” the robot. As a bonus, the robot surrogate can be given enhanced strength, endurance and sensory capabilities.

If telepresence is such a great idea, why do we need humans in space? For one, the technology is not yet available. Vision is the most important sense used in field study, and no real-time imaging system developed to date can match human vision, which provides 20 times more resolution than a video screen. But the most serious obstacle for telepresent systems is not technological but psychological. The process that scientists use to conduct exploration in the field is poorly understood, and one cannot simulate what is not understood. Finally, there is the critical problem of time delay. Ideally, telepresence requires minimal delays between the operator's command to the robot, the execution of the command and the observation of the effect. The distances in space are so vast that instantaneous response is impossible. A signal would take 2.6 seconds to make a round-trip between Earth and its moon. The round-trip delay between Earth and Mars can be as long as 40 minutes, making true telepresence impossible. Robotic Mars probes must rely on a cumbersome interface, which forces the operator to be more preoccupied with physical manipulation than with exploration. (Spudis 1999)

There are two major problems Spudis writes about: 1) poorly understood work practice of field science, both on Earth and in space, and 2) the issue of time delay in teleoperated robots and its impact on the interface between humans and machines. Both these issues have been the partial focus of the previous case studies in this thesis. Unlike Spudis, who states that “one cannot simulate what is not understood,” in the previous two case studies I used modeling and simulation to understand the work practices of field scientists on the Moon, *while* in the process of developing the simulations. In other words, in the previous case studies I have developed a way of using multiagent simulation technology as a technology solution for *the process of understanding*. In this case study I go one step further, and use modeling and simulation as *a process for designing solutions*.

In this study, I use Brahms to design the human-robotic collaboration⁵⁸ for the Victoria mission. The Victoria mission proposal is a recently submitted NASA proposal for a semi-autonomous robot to search for water on the South Pole of the Moon. In this proposal an Earth-based science team will conduct a long-term robotic mission investigating permanent dark areas on the Moon's South Polar region. A semi-autonomous robot will be the agent doing the actual field science, while in constant communication with the science team. The issue I address in this case study is the use of work practice modeling and simulation for design of the human-robotic collaboration.

Goals and objectives

The goal of this case study is to show that with Brahms we can design how humans and robots can work together. In the process of developing a computational model of a new human-robotic work system we acquire requirements for the systems (such as for the robot and data systems), communication, team interactions, and the distribution of work activities.

The research question in this case study is:

⁵⁸ Some people might question the concept of collaboration with a teleoperated robot, and I am aware of this seemingly strange notion. However, in this thesis the notion of collaboration is used specifically in situations where one or more agents (human or machine) are aware of each other's activities in pursue of a common goal. The question is thus not *if* the humans and the robot are collaborating, but instead we accept the fact that the humans in this endeavor *feel* that they are in a collaborative activity with the robot and other human participants. Therefore, the question becomes, *how can we make the robot aware of the fact that such collaborations exists?* Answering this question goes beyond the scope of this study, but a start will be made with getting closer to answering this question, by creating a model of this collaboration.

Can the Brahms modeling and simulation environment be used to prescribe a realistic work practice in the design of the human-robotic collaboration during a robotic lunar mission?

The objective of this case study is to design a working Brahms simulation that models a relatively simple, but realistic human-robot collaborative activity of the Victoria rover described in the Victoria mission proposal. The model should show in relative detail the collaboration and distribution of activities between humans on Earth and the robot on the Moon. In the next sections I describe the result of this study. First, the next two sections describe the Victoria mission in more detail. Then, I describe how the use of Brahms in this case study will be evaluated. I will then briefly describe the V&V of the model. After this the model details are described, as well as the outcome of the simulation. Last, I will conclude with some observations.

8.1 VICTORIA MISSION

Victoria is the name of a proposed long-term semi-autonomous robotic mission to the South Pole region of the Moon at the end of 2005. The name Victoria was chosen after the only ship of Ferdinand Magellan's voyage that circumnavigated the world⁵⁹.

At the start of this case study the Victoria team was in the middle of writing the proposal. Team members (so called Principal Investigator and Co-Investigators) of the Victoria mission are world-renowned scientists from different scientific disciplines (planetary scientists geologists, robotists, and AI-specialists). The description and all data mentioned in this thesis come from the drafts of the actual proposal, which can be seen as functional requirement specifications for the mission. Scientific space mission proposals are necessarily very detailed descriptions of every aspect of the mission. One could view a robotic science mission proposal as a detailed requirement specification for the proposed science investigation, the science instrumentation, the launch vehicle, spacecraft and robot, and the control & data communication.

For a space science mission to be funded, a proposal needs to withstand many reviews and severe competition from other proposals. It is therefore not surprising that a proposal needs to provide the reviewers with much detailed information showing that, when funded, the mission is doable and the proposed goals and objectives are likely to be met within the proposed schedule, with the proposed mission technology and within the proposed budget. The Victoria mission proposal is a multi-million dollar project description and plan from which scientists and engineers are able to start the design and implementation of the needed mission elements.

8.1.1 Mission overview

Victoria's fundamental goal is to gain a better understanding of the history of volatiles in the solar system, by first getting a deeper understanding of how these volatiles might be preserved at an ideal site in space. It is determined from the data returned by the previous Lunar Prospector⁶⁰ mission that the Moon's permanently shadowed areas at the poles might be an ideal site for the preservation of volatiles, in particular in the form of water ice (Hubbard et al. 1998). There is a secondary goal of gaining a better understanding of the evolution of the Moon. Even though the Apollo missions significantly increased our knowledge about the Moon's evolution, the ultimate question of how the Moon came about is still not positively answered today. Given this, the mission's objectives are to:

1. Verify the presence of water ice and other volatiles within permanently shadowed regions on the Moon. This will be accomplished by gathering the necessary in-situ data for analyzing the history of water and other volatiles on the Moon, and by implication in the inner solar system. This is the primary mission science objective.
2. Perform a geological survey of the southern lunar polar region, in extreme high resolution.
3. Determine the composition of recent pyroclastic type deposits around mare-type volcanos that are between one to two billion years old, as well as nearby mare basalts that are about 3.2 billion years old.

⁵⁹ Ferdinand Magellan, (1480?-1521), Portuguese-born Spanish explorer and navigator, leader of the first expedition to circumnavigate, or sail completely around, the world.

⁶⁰ <http://lunar.arc.nasa.gov/>

4. Transect the South Pole-Aitken (SPA) basin creating a fine scale geological survey to understand the interaction between the SPA impact⁶¹ event and the underlying mantle materials.

These are the high-level scientific purposes of the Victoria mission. From these scientifically important objectives, the Victoria team argues that the most efficient way to meet these science objectives is to use a high-speed semi-autonomous rover that can traverse over long distances (several hundreds of kilometers), for a long time period (three months to a year), to gather the necessary geological and physics data.

8.2 PROBLEMS WITH AUTOMATED PLANETARY SURFACE EXPLORATION

8.2.1 Data overload

Scientific field exploration on Earth is difficult work. Not only does the scientist have to endure tough environmental conditions, such as being in cold or warm climates for a long period of time, isolated from the rest of the world, but also because of its data intensive nature and the fact that good scientific principles need to be adhered to. Automated planetary surface exploration will for the most part eliminate these environmental and isolation problems. It will not, however, change the data intensive nature of the work. As a matter of fact, recent studies of remote science team activities in automated field exploration on Earth has shown that the amount of data that is being collected automatically by the robot in the field is sometimes overwhelming for the science team. Thomas et al, note the following problem with data archiving in their study of a remote science team at NASA Ames Research Center, during an experiment for automated planetary surface exploration (Thomas et al. 1999, p.24);

The greatest difficulties were noticing the arrival of new information and finding relevant information entered by other researchers. Some data products were only available after the initial, raw information was reprocessed. Consequently information became available at different times, but no global mechanism was available to alert the scientists.

Based on the problem of data archiving observed by Thomas et al, I will address the following questions in the model:

1. How will science data *be gathered*, in collaboration with the Earth-based science team, rover teleoperator, and the rover on the lunar surface?
2. How will the gathered science data *be made available* to the science team on a continuous basis during the long mission?

To answer these questions, I will develop a model of the activities of the above-mentioned teams, based on the description of a planned mission traverse described in the Victoria proposal.

8.2.2 Power constraint

One of the biggest constraints in any robotic mission is power consumption of the robot. A robot gets its energy from onboard batteries. These batteries are charged by solar energy, using large solar arrays on the robot (Figure 8-1). In every activity the rover uses energy, therefore the sequence of activities for the rover is constraint by the amount of power available to complete the sequence. When the robot's batteries are low, it needs to return to a sun-exposed spot in order to recharge its batteries. Batteries are heavy artifacts that need to be brought up in space, and are therefore limited in size and power. This makes the robot power consumption issue a very important constraint in the design of the robot, but also a very important constraint in the ability of the robot to perform certain activities during the mission.

⁶¹ SPA is the largest recognizable impact basin in the solar system.

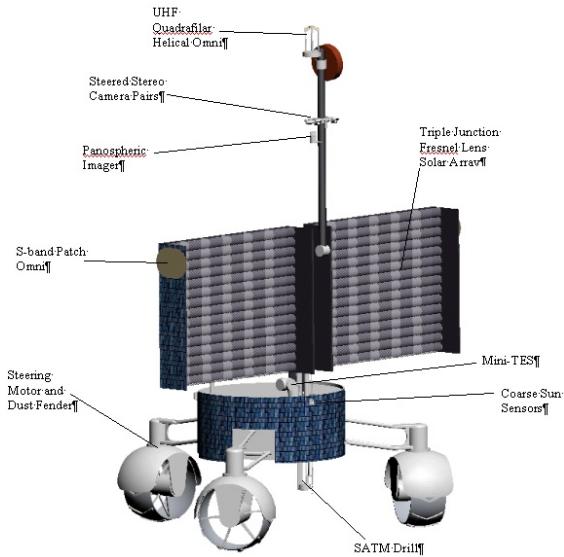


Figure 8-1. Victoria Rover

In order to calculate the power consumption for the Victoria rover at any moment in time, I developed the following equation:

$$\begin{aligned}
 \text{Power consumption for rover at time } t (\text{Prover}(t)) = & \\
 \text{power for driving } (P_d(t)) + \text{ power for command \& data handling } (P_h(t)) + & \\
 \text{power for science instrumentation } (P_i(t)) + \text{ power for communications } (P_c(t)) + & \\
 \text{power used for thermal protection } (P_t(t)) + \text{other (not measurable power)} &
 \end{aligned} \tag{1.0}$$

Given (1.0), we can now define the *energy usage during an activity* for the rover to be:

$$E_{acti} = \int_{start\ of\ acti}^{end\ of\ acti} \text{Prover}(t) dt \tag{2.0}$$

The *energy constraint* that exists during a mission traverse into a permanent dark area is:

$$\int_{start\ of\ traverse}^{halfway\ point\ of\ traverse} \text{Prover}(t) dt \leq \text{Pbattery}(start\ of\ traverse) - 15\% \text{ full battery reserve} - 50\% \text{ of Pbattery}(start\ of\ traverse) \tag{3.0}$$

This constraint says that the total energy usage of the rover during the traverse into a permanent dark area cannot be bigger than the available battery power minus a 15% reserve battery capacity, and minus a 100% margin (i.e. 50% of Pbattery(start of traverse)). The 15% reserve is a standard reserve that has to do with the battery operation specifications. The 100% margin is used only in traverses into permanent dark areas. The idea is based on the fact that when the rover does a mission in a permanent dark area, it will use the full battery capacity to make the mission as effective as possible. With a full battery you can safely assume that you can use half of the battery life to drive *into* the permanent dark area, and you need half of the battery life to *get out* of the permanent dark area. Therefore, the total energy you can use to get into the permanent dark area is given by the energy constraint (3.0). In other words, the time available to get to the halfway point of a traverse within a permanent dark area is constraint by the time it takes to use the maximum power available that still does not violate constraint (3.0).

8.3 EVALUATION CRITERIA FOR USE OF BRAHMS IN DESIGN

The objective in this study is to show that Brahms as a modeling language can help in specifying work practices, while the simulation engine permits execution and evaluation of a newly designed work system.

During the Victoria mission the rover will traverse into permanently dark regions on the Moon. These are interesting areas, because of their potential to keep water ice from evaporating. These areas exist particularly on the South Pole of the Moon, because a) the angle of the Sun at the South Pole and b) the terrain relief due to large impact craters. A problem with traversing into permanent dark regions is the fact that the robot needs to have enough power to make it safely out of the permanent dark region before its battery is empty. This might not be immediately obvious, but it should be clear when one realizes that when the robot runs out of power in a permanent dark region the mission is over and done with. Permanently dark means no sun, ever, and if sun is the only way to recharge the robot's batteries we can easily see why this is a very important constraint.

Remembering that the previous two case studies proved that modeling and simulation with Brahms has a “reality value,” the quality criteria to be applied to this case study are radically different from the ones in the first two. In this case study the Brahms language and the simulation engine are used as *design aids*, which implies that the value should be judged on the aid provided, i.e. an evaluation of this aid on relevant criteria. These criteria must be elaborated, defined and judged. In other words, the quality of the model and the aid the model simulation provides in design is more or less subjective.

The work system design method proposed in this case study involves modeling, simulation, and analysis of a candidate work system for the Victoria mission. There are two quite different uses of modeling and simulation in this area:

1. A Brahms model may be developed and simulated to *verify* the operational correctness in terms of the work system, as well as to evaluate the performance of the designed system. During such an effort, more than one design of the work system may be modeled and evaluated to select the best one for implementation.
2. In complex systems, plans are often used for short time horizons. The relative unpredictable nature of reality will make a plan useless at the moment situations occur that were not previously considered during the planning activity. During the actual Victoria mission, continuous changes to the planned activities of the rover and science team will be necessary. If we have a validated model of the implemented work system, we can use this model for *continuous mission planning* during the mission. Necessary changes to the plan can be modeled, simulated and evaluated before the actual changes to the active plan are being carried out. What-if scenarios can be simulated and different plans compared.

Formulated as *objectives* for modeling, these two purposes lead to two quite different experimental frames. An *experimental frame* is a specification of the conditions under which the system is observed, or experimented with. As such, an experimental frame is the operationalization of the objectives that motivate the modeling and simulation project (Zeigler et al. 2000).

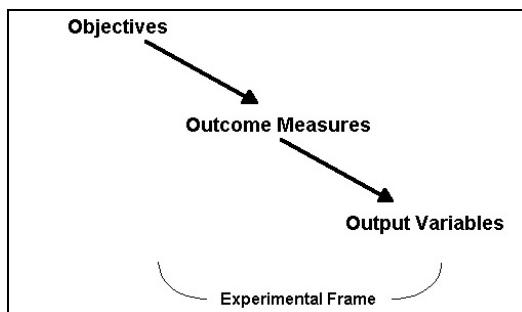


Figure 8-2. Transforming objectives into experimental frames (borrowed from (Zeigler et al. 2000))

Figure 8-2 shows the process of transforming objectives into an experimental frame. In order to evaluate design alternatives, modeling objectives require measures of *effectiveness* for the system to accomplish its goals. Such measures are called *outcome measures*. To compute such measures the model needs to include *output variables* that are computed during model execution runs. Therefore, for the above-mentioned objectives for the use of Brahms in work system design we will need to identify output variables that represent system performance.

8.3.1 Outcome measures

For this thesis, I am focusing on the use of Brahms for *verifying operational correctness* of a specific work system design (i.e. #1 use). Possible measures that the model could provide are⁶²:

- human activities performed
- robot activities performed

⁶² One could define a number of additional relevant measures, but for the purpose of this case study, in light of this thesis, adding more measures does not necessarily enhance the outcome.

- the time it takes for the team to complete a mission objective
- activity times per agent (human and robot)
- power consumption and energy usage for the robot agent
- robot command sequence send by the Earth-based team to the robot
- amount and type of data send to Earth by the robot, and used for decision analysis and historical record
- amount of surface covered during a traverse

We call these *outcome measures*. By identifying such outcome measures, we are able to identify the output variables that a Brahms simulation should be able to calculate. By investigating the ability of the Brahms language to model such output variables, and the ability of Brahms to determine these variables during a simulation run we are able to evaluate the use of Brahms in a more objective manner.

8.3.2 Output variables

Given the above outcome measures we can define the output variables of the simulation. The simulation will produce the output variables, and they can be used to determine the performance of the designed work system. If we have enough confidence in the validity of the model, we can use these output variables to draw conclusions about the operational correctness of the particular work system design. Table 8-1 shows the output variables that will be generated during the simulation:

Table 8-1. Simulation output variables

Outcome Measures	Output Variables
human activities performed	a list of activities for each human-agent
robot activities performed	a list of activities for each robot-agent
the time it takes for the team to complete a mission objective	the total duration of a high-level mission objective activity
activity times per agent (human and robot)	the duration time of each specific agent-activity
power consumption and energy usage for the robot agent	the total power consumed by the robot agent, the energy used in each robot activity, the battery power left after each robot activity,
robot command sequence send by Earth to the robot	data send to the robot
amount and type of data send to Earth and used for decision analysis and historical record	for each data transmission made by the robot, its type and the amount of data
amount of surface covered during a traverse	length and time of traverses, number of mission relevant stops

8.4 MODEL VERIFICATION AND VALIDATION

In system design, i.e. when designing a system that does not yet exist, it seems only possible to do model verification. Model validation is difficult, because we cannot compare the outcome of the model with the outcome from an existing source system (Zeigler et al. 2000, chapter 14, fig.1). In this section, I describe how the simulation in this case study is verified and validated.

8.4.1 Verification

As described in chapter 6.9.5, verification is the attempt of establishing that the simulation relation holds between the simulator and the model being simulated. In other words, does the simulator execute the model correctly? Correctly in this context means, does the simulator execute the model as intended, after the compiler has been able to correctly translate the model source code?

The model is developed based on the Apollo HFE deployment model from the second case study on the one hand, and the Victoria mission proposal on the other. This data allows for a relatively accurate design of

the activities and the robot. To verify the correctness of the simulation of the model, I presented the conceptual design model of the Victoria work system to the principal investigator, and a science team member of the Victoria mission, and based on their feedback I implemented the conceptual design into a Brahms model. I also interacted with co-investigators of the Victoria mission on the modeling of the rover and its instruments. With one of the designers of the Victoria rover from Carnegie Mellon University I had design interactions on the energy performance of the rover. With the designers of the lunar surface drill (Honeybee Robotics, Ltd.) and the research scientists responsible for the Neutron Detector instrument, I discussed the operations and energy consumption of these instruments. After syntax error revisions, and because of the two previous case studies, we can be confident that the Brahms simulator executes Brahms models correctly. I then verified the model simulation using the visual AgentViewer application. After several debugging cycles, I verified that the outcome of the simulation produced the correct behavior of all the agents in the system over time.

8.4.2 Validation

The point of the exercise is to test the use of simulation as a design aid for the implementation of the actual system. The model consists of a number of modeled agents (humans and robots) and systems that could individually be validated. However, the overall system behavior is what determines the effectiveness.

Since there is no real-world system data, the way I have been able to validate is to use a rather qualitative comparison step; i.e. validating the model based on responses from the Victoria principal investigator. After the model was simulating correctly, I presented this outcome to the principal investigator of the Victoria mission and he validated its behavior. This gives a relative confidence that the model indeed simulates the appropriate level of behavior of the future work system.

8.4.3 Model fidelity

An important concept in design models is *fidelity*. Fidelity is often used for a combination of validity and detail. Thus, a high-fidelity model may refer to a model that is high both in detail and in validity. However, one needs to be aware of the fact that high detail is a necessary, but not sufficient condition for high fidelity. There can be a highly detailed, but invalid model that is very much in error. This tacit assumption is especially important in design models, because design models are very difficult to validate before a system is implemented and we are able to validate the model against the implementation. Therefore, it is important to realize that high detail does not eliminate the validity concern in work system design models.

Model detail depends on the objective of the modeling effort. The more demanding the question, the greater the resolution needs to be to answer the question. The level of detail of the human-robotic work system to be designed should be high enough so that the modeling activity is relatively simple and fast, but on the other hand detailed enough to be relevant for mission-operation designers, robot designers, and/or software developers.

The collaboration between the humans on Earth and the robot on the Moon is grounded in the performance of a specific task. What needs to be included in the model should be based on all the relevant activities and contextual information that influence this collaborative activity. The level of detail at which to model each aspect depends on our ability to identify those aspects that have influence. There is no hard and fast rule that can be applied to the modeling level. For example, it is obvious that the communication time-delay between the Earth and the Moon needs to be included in whatever collaborative activity is being designed. However, it is not clear if and how the make-up of the mission science-team has to be included in the model. I could decide to model every individual member of the science-team as separate agents, while on the other hand I could decide to model a whole team as one agent communicating their decisions to other teams. From previous field-test simulations it seems obvious that the work practices of the human teams has great influence on the collaboration with the robot in performing field science (Thomas et al. 1999) (Cabrol et al. 2001).

In system design, the measures of the effectiveness of the system in accomplishing its goals are required to evaluate the design alternatives. This suggests that the outcome measures from Table 8-1 could be used to measure the effectiveness of a specific work system design model. One of the most important measures for

the evaluation of an alternative is the power consumption and energy usage of the robot during the model simulation. For a design alternative to be accepted it will at least need to satisfy energy constraint (3.0). The level at which to model is the level that can show and explain all the events that influence this constraint. The objective in the rest of this chapter is to describe how this was done in the Victoria model.

8.5 MISSION OPERATIONS WORK SYSTEM DESIGN

The work that is involved in the Victoria mission is distributed over a number of human teams, and the Victoria rover. In a sense, we can view the Victoria science team as a user of the semi-autonomous rover. The science team is a user from the perspective that the rover is on the lunar surface in service of the Earth-based science team. On the other hand, the rover is not merely a computer system with a user in the traditional sense of the word. The rover is more of a collaborator with its user. From the perspective of a team performing a scientific lunar surface exploration mission, the rover is part of the team. The work practice of this team is what will make the difference in the performance of this team. As such, the rover cannot be simply viewed as a complex piece of machinery that needs to be remotely controlled by its user, but instead the rover has to be seen as an integral part of the team performing the work; Who is doing what, where, when, and how? The purpose of this case study is to show how a model of the prescribed work practice can aid the mission operation system (MOS) designers in designing the most efficient work system for the mission, given its objectives.

8.5.1 Functional division of activity

Organizational structures of mission operation teams for robotic missions can be based on similar functions for more traditional remote-sensing missions. Wall & Ledbetter describe the organizational structure of remote-sensing missions in functional terms (Wall and Ledbetter 1991). The two top functions described are:

1. Keep the robot safe and functioning, and
2. Request, collect, process and analyze the data sent back by the robot.

However, in the case of a semi-autonomous rover mission the rover is defined as a member of the mission operations team. Therefore, I identify a third important top-level function.

3. Perform the science activity and send data back to Earth.

These functions require two distinct data flow processes: *uplink*, the definition, preparation and transmission of instructions and data to the robot; and *downlink*, the collection, transmission and processing of data from the robot. Wall & Ledbetter describe the sub-functions necessary for accomplishing these two data flow processes from an Earth-based mission operations point of view. (see Table 8-2).

Table 8-2. Relation of data flow process to functions (from (Wall and Ledbetter 1991))

	Robot/Spacecraft operation and safety	Request, collect, process, analyze data
Uplink process	<ol style="list-style-type: none"> 1. Commands for engineering operation of robot/spacecraft 2. Maneuver commands 3. Telecommunications commands 4. Emergency or anomaly resolution commands 5. Engineering data loads 	<ol style="list-style-type: none"> 1. Command sequences for experiment operation 2. On-board data storage commands 3. Payload pointing commands 4. Long-term planning for science opportunities
Downlink process	<ol style="list-style-type: none"> 1. Monitoring of health and status telemetry from robot/spacecraft subsystems 2. Subsystem trend analysis 3. Subsystem performance prediction 4. Quality of downlink signal 	<ol style="list-style-type: none"> 1. Experiment data collection 2. Data processing and enhancement 3. Image processing 4. Data quality assessment

The functions in Table 8-2 need to be performed collaboratively by the mission operation teams and the rover. How this happens is constrained by the work system that needs to be designed. The model will prescribe the work system by incorporating a model of the work practice, including all the Earth-based teams, the rover, their communication actions, as well as the hardware and software systems they use.

Figure 8-3 gives a pictorial representation of the currently known elements and their relative geographical location during the Victoria mission. The Science Team consists of a number of sub-teams, all co-located in Building 244 at NASA Ames Research Center, Moffett Field, California. The sub-teams are the Science Operations Team (SOT), the Instrument Synergy Team (IST), and the Data Analysis and Interpretation Team (DAIT). There are two other supporting teams outside the Science Team. These are the Data and Downlink Team (DDT) and the Vehicle and Spacecraft Operations Team (VSOT). All these teams work together to perform the mission. In doing so, their objective is to accomplish the scientific objectives of the mission. They communicate with the Victoria rover on the lunar surface, using the Universal Space Network (USN) via two separate communication links, the high capacity S-Band direct Earth to rover link, as well as the UHF communication link via Victoria's lunar orbiter.

The flow of data from the rover will be dominated by contextual and multi-spectral image data, but will also include thermal emission, neutron spectrometer, time-of-flight mass spectrometer (TOF-MS), X-ray spectrometer (APXS), microscopic imaging, and various engineering data of scientific interest. This data will come to NASA Ames via the USN data connection and will be automatically converted in near real-time to accessible data formats that can be made available to the teams via data access and visualization applications. In addition to this continuous data conversion activity, the data will be streamed to a redundant DVD storage facility—that will then immediately be made available to the Planetary Data System (PDS) in raw format. Within a 24 hours time period, the Victoria science team will release the images of greatest interest on the public available Victoria web site.

In the next sections I will describe the design of this work system through the design of the agent model, the object model, their activity models and the geographical model

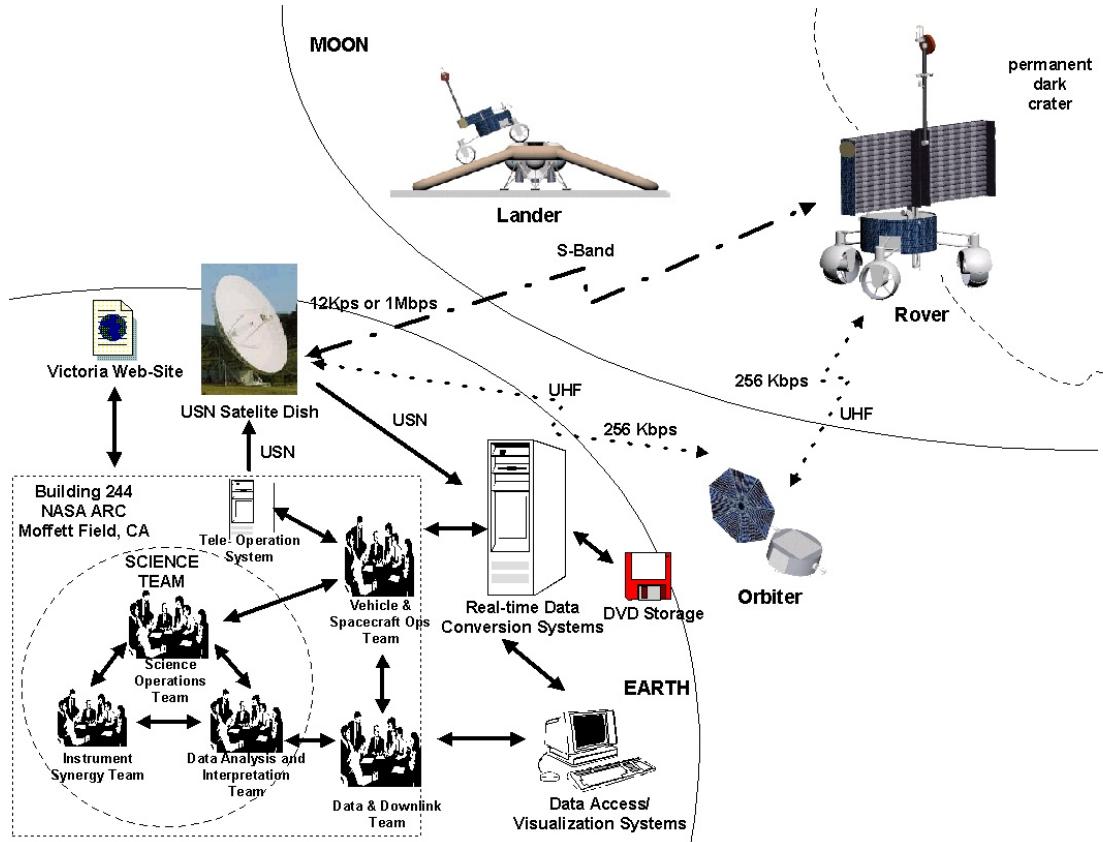


Figure 8-3. Victoria work system

8.5.2 Agent model design

Figure 8-4 shows the group membership hierarchy on which the design of the work system is based. The agents in the model are the Earth-based human teams and the Victoria rover, as shown in Figure 8-3. The teams are represented as agents, because at this moment it is not possible to describe the composition of each team in more detail. This means that the activities represented in the model are at the team-level, and it remains unspecified how the teams themselves inter-operate. For example, the “plan a command sequence” activity of the SOT represents the work of the whole team, while the individual activities of each team member remain unspecified. As the team structure and inter-operation of the teams become known, we could update the model to reflect more specific internal team-design. The modular agent-based design allows us in the future to decompose the team into multiple agents, as well as decompose each team-activity into more specific team member activities.

The VictoriaRover is modeled as an agent, i.e. an instance of the group Rover, which is a subgroup of the group DataCommunicator. Agents of the group DataCommunicator know how to create and communicate data objects. This is relevant for rovers, because they need to communicate information to Earth. The VictoriaRover agent can communicate either over the S-Band directly to Earth, or over the UHF-band via the orbiter.

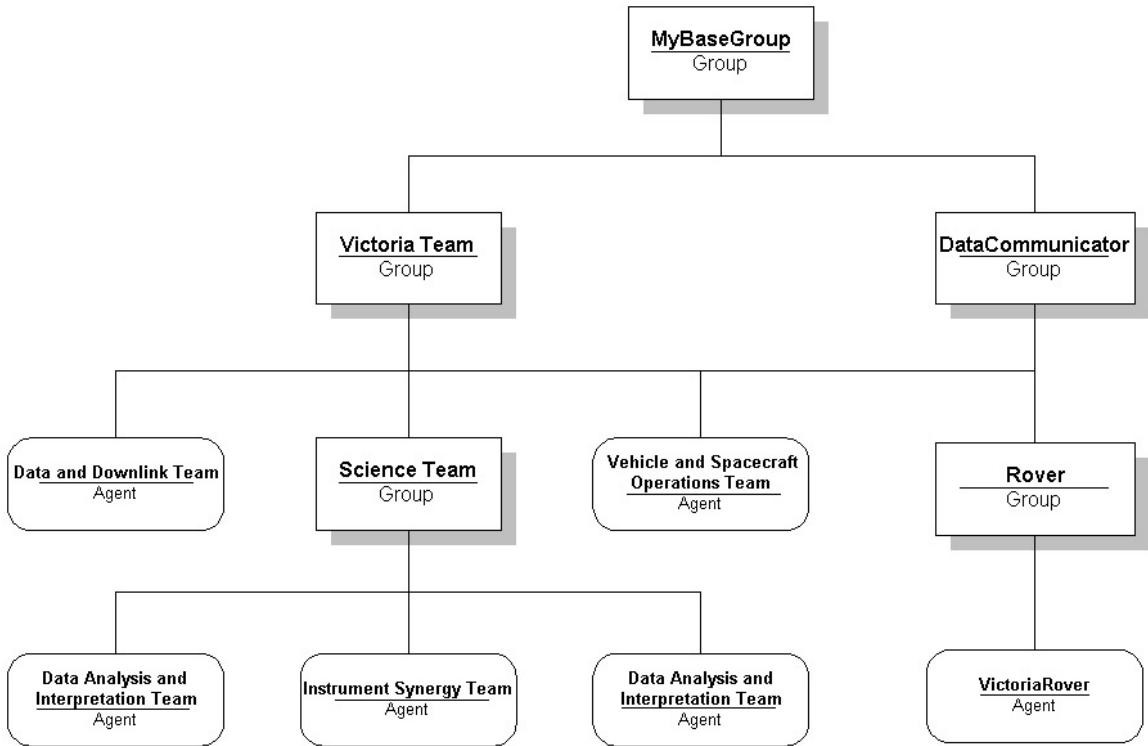


Figure 8-4. Victoria Agent Model

Given the above teams, Table 8-3 shows a possible distribution of the functions from Table 8-2 over the Victoria teams. Different teams collaborate together to perform these functions. How such collaborations happen depends on the work practice, specified in the situation-action rules (i.e. the workframes) of the different agents.

Table 8-3. Functional activity distribution over Victoria teams

	Science Operations Team	Instrument Synergy Team	Data Analysis and Interpretation Team	Data and Downlink Team	Vehicle and Spacecraft Operations Team	Rover
Uplink process	1. Maneuver commands 2. Command sequences for experiment operation 3. Payload pointing commands 4. Long-term planning for science opportunities	1. Commands for engineering operation of robot/spaceship 2. Emergency or anomaly resolution commands 3. Payload pointing commands 4. Long-term planning for science opportunities	1. Long-term planning for science opportunities	1. Telecommunications commands	1. Commands for engineering operation of robot/spaceship 2. Maneuver commands 3. Telecommunications commands 4. Emergency or anomaly resolution commands 5. Command sequences for experiment operation 6. On-board data storage	1. Command execution 2. Long-term planning for science opportunities

					commands 7. Payload pointing commands	
Downlink process		1. Monitoring of health and status telemetry from robot subsystems 2. Subsystem trend analysis 3. Subsystem performance prediction	1. Data quality assessment 2. Experiment data collection	1. Quality of downlink signal 2. Experiment data collection 3. Data processing and enhancement	1. Monitoring of health and status telemetry from robot subsystems 2. Subsystem trend analysis 3. Subsystem performance prediction	1. Experiment data collection 2. Monitoring of health and status telemetry from robot subsystems

For the purpose of this study, the model only includes some of the functions from Table 8-2, and consequently also from Table 8-3.

8.5.3 Object model design

The object model consists of the classes and instances of the artifacts, as well the statically and dynamically created data objects. The Victoria object model (Figure 8-5) includes classes for the science instruments on the rover, as well as other objects contained in the rover, such as the carousel and the battery. Furthermore, the model includes the data communicator class, which includes the objects for S-band and Uhf communication. The simulation scenario, presented in the next section, only requires the S-bandMGA antenna on the rover. The model also includes the software system that is needed to receive and convert the mission data, as well as an object that represents the data visualization system needed to present the Victoria team with the data in a usable format. The Data and CoreSample classes are used to dynamically create data instances and lunar core sample objects, during the simulation.

8.5.4 Geography model design

The geography model is similar to that of the two Apollo models, since we are again modeling geographical locations on Earth and on the Moon. Figure 8-6 shows the Victoria geography model design, as is depicted in Figure 8-3. There are two distinct areas of interest on Earth; The Building244 area where the Victoria teams and systems are located, and the UsnSatelliteLocation area where the UsnDish1 satellite dish is located. On the Moon, the areas represented are locations for the specific scenario that is being simulated (see next section). There is a location called ShadowEdgeOfCraterSN1, which represents the location the rover is at the start of the simulation. This is the location on the shadow edge that is in crater SN1. Another location that is important during the scenario is the area ShadowArea1InCraterSN1. This represents the specific location in the permanent shadowed SN1 crater where the rover will perform a drilling activity. The LandingSite area is only represented for completeness, and does not play a significant role in the simulation of the actual scenario.

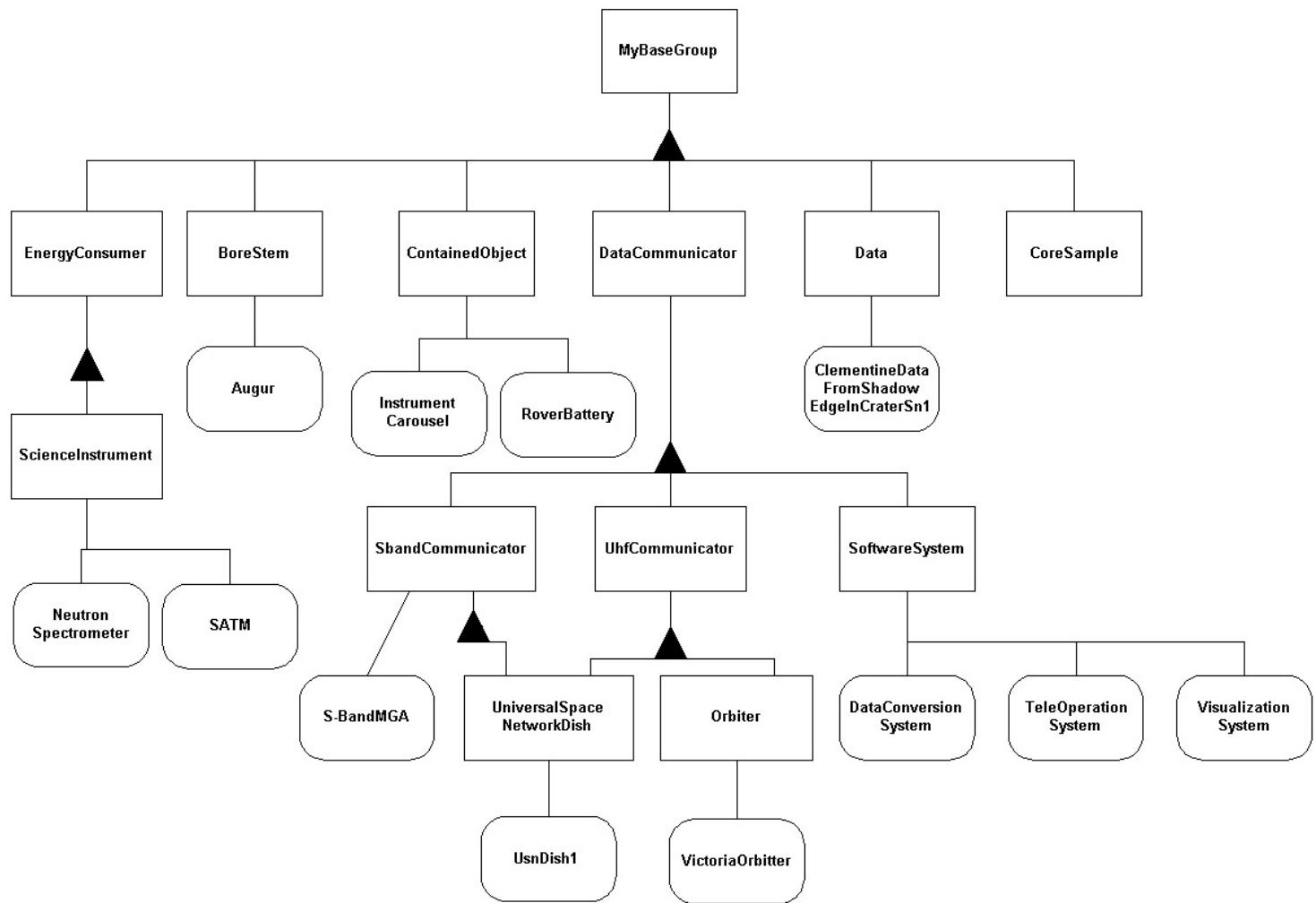


Figure 8-5. Victoria Object Model

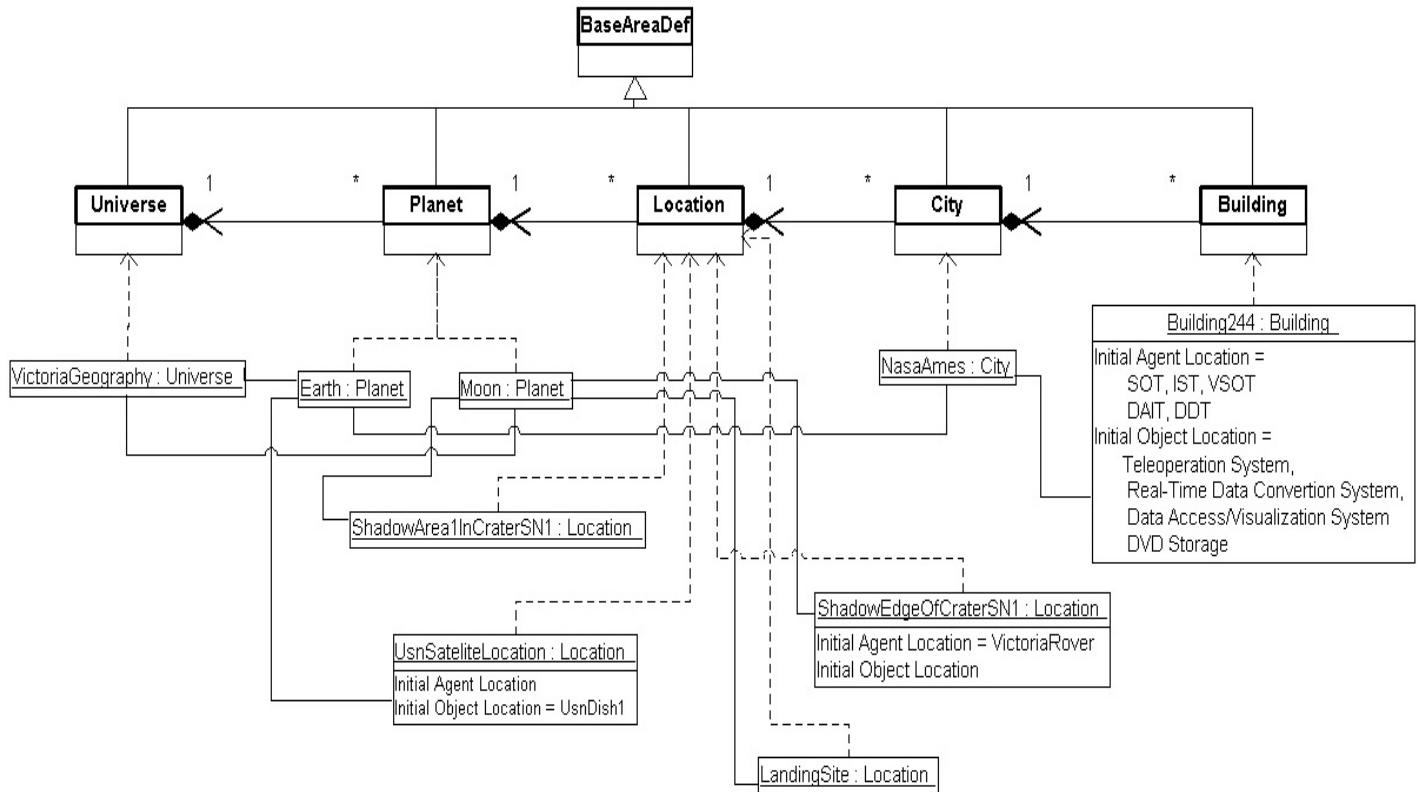


Figure 8-6. Victoria Geography Model

8.6 MODEL SIMULATION SCENARIO

The Victoria proposal spells out a number of surface activities that will be performed by the rover, in collaboration with the teams on Earth. For this case study I selected the activity of *searching for water in permanently shadowed craters*, as described in the proposal:

[T]he Victoria team will proceed to the crux of the primary science objective, i.e., the characterization of volatiles, including ice in a permanently shadowed crater or depression. The rover will traverse from the landing site area to a preselected nearby crater that contains a permanently shadowed area. It will seek ice and other volatiles that are expected to be present in such a permanently shadowed cold trap. Several regions close to the pole exist that fulfill Victoria's exploration criteria:

Site number 1 is a depression located at 89.5°/70°W. The depression is approximately 10 km across. Both radar and Clementine data indicate that it is permanently shadowed in its center. Assuming that a landing site close to the pole is selected, high resolution coverage exists from the pole along a ridge to the edge of the depression.

Site number 2 is located 88.5°/120°E. Clementine showed that there is a plateau in this region receiving large amounts of illumination. Several dark zones surround this plateau and appear to be permanently in the dark. The good hires coverage of this region, coupled with Victoria team members [...] who are particularly knowledgeable about the lunar polar regions and their topography, will ensure an optimization of the current existing data and a minimization of environmental risks.

Upon arriving at the chosen crater, the rover will travel down into the selected crater and to the edge of the shadow. Then, a full battery charge will be confirmed and the rover will traverse into the crater's dark area for a duration of approximately one hour. Over the next few Earth-days several rover traverses will be performed into this shadowed zone, each lasting up to three hours or longer. The Victoria's rover is capable of reasonably high speeds on much of the lunar terrain, but it is anticipated

that shadowed zones will be traversed at speeds not exceeding 1 m/sec. During this traverse, the rover will periodically stop and deploy its neutron detector to seek hydrogen within the first half meter of the surface. Either at the detection of hydrogen or at a fixed time, the rover will use the drill to collect samples from a depth of approximately 1 m below the surface. Samples will be examined by the microscopic imager, APXS (modified for the detection of hydrogen), TDL looking for volatile water and by the laser VIMS. For this phase of investigation, the integration time of the neutron detector and APXS will be approximately 15 minutes and for the laser VIMS and TDL less than a minute each. The 1m drilling for a core will take a half-hour or less. The analysis of the core sample will give us a stratigraphic record of the texture, volatiles, elemental analysis and unambiguous water ice present in the top meter of the surface. At half its traverse time, the rover will retrace its tracks back to lighted areas to recharge its batteries.

The part of the scenario that is modeled is the traverse into crater site number 1. The model scenario is as follows:

The rover has arrived at the shadow edge of crater site number 1. The battery has been fully charged. Based on the data analysis by the Earth-based teams, of the Clementine data available for the shadow edge area of crater site number 1, the science team now decides where to go into this crater and search for water ice. While the rover is traversing into the crater, it is taking hydrogen measurements with the Neutron Spectrometer. When the rover arrives at the assigned location within this crater and it finds hydrogen there, the science team decides it should start drilling 10cm into the surface using the SATM, and collect a 1.0cc lunar sample. When the rover receives this command, it starts the drilling activity and finally deposits the sample into the instrument carousel.

Table 8-4 describes these two instruments in terms of the science it is used for.

Table 8-4. Victoria rover instruments used during scenario

Instrument	Science
Neutron spectrometer  <p>Lunar Prospector Neutron Detector Dr. Darrell Drake Los Alamos National Laboratory</p>	Detect hydrogen within the first half meter of the lunar surface below the rover. The most likely form of this hydrogen is water ice.
SATM drill  <p>Sample Acquisition and Transfer Mechanism Stephen Gorevan, Honeybee Delvatec</p>	SATM can penetrate the lunar regolith to depths of over 1 m to acquire samples while preventing cross-contamination. The SATM is equipped with a sample cavity volume capable of between 0.1 to 1.0 cc to acquire samples of different lengths and at different depths below the lunar surface.

In this scenario the rover uses two of its instruments, the Neutron Spectrometer and the lunar surface drill, called SATM⁶³. In the next section, I first describe the activities of the rover during the scenario. After that I describe the activities of the Earth-based teams. The reason for ordering the discussion of the model this way is because the reader will have an easier time understanding the activities on Earth, by first having an understanding of the rover's activities. The rover activities are a result of the Earth-based team's behavior, and they better explain the objective of the scenario.

8.7 ROVER ACTIVITY

The Victoria rover is modeled as an agent, whereas the neutron spectrometer and SATM instruments are modeled as separate science instrument objects contained in the rover agent. Both instruments are modeled to perform the science activities according to their definition. Figure 8-7 shows the VictoriaRover's activities during simulation of the scenario. The NeutronSpectrometer object is active and creates a HydrogenData_1 object containing the hydrogen data that is send to Earth, while the VictoriaRover is traversing to a permanently shadowed area within the crater, known as site number one (see *Move to area in crater and look for hydrogen* in Figure 8-7). Next, the rover is waiting for the next command sequence from Earth (see *Waiting for command from Science Team* in Figure 8-7). During this time the Earth-based teams are analyzing the hydrogen data and are deciding what to do next (see section 8.8). The simulation shows that the rover is given the command to search for water ice in the permanent dark area. This triggers the SATM instrument to start the drilling activity. Figure 8-7 shows the SATM and Augur objects performing activities in order to collect a sample from the lunar soil (see *Drill 10cm into surface and take 1cc sample* in Figure 8-7).

To collect a sample the SATM has to 1) lower its augur to the surface, 2) drill to the depth given as part of the command by the Earth-based Science Team (in this scenario the command says to take a 1.0cc sample at 10cm depth), 3) open the sample cavity door, 4) continue to drill to collect the sample, 5) closing the sample door when done, 6) retract the drill from the surface, and finally 7) depositing the collected sample on the instrument carousel. Figure 8-7 also shows the Augur object, contained in the SATM object. The Augur object creates the LunarSample_1 object as part of its activity to capture the lunar sample, after opening the sample door and continuing the drilling to collect the 1.0cc sample. The way the drilling activity works in the model is copied from the way the drilling activity worked in the previous Apollo HFE deployment case study (see section 7.5.3). This shows a low-level model reuse.

The activity times for drilling into the surface are dynamically derived during the simulation. Honeybee Robotics, Ltd.⁶⁴ provided the times for moving the augur to the surface, opening and closing the sample door, as well as the average time it takes to drill the augur into, and retracting it out of the lunar surface. Table 8-5 gives the sub-activity duration for the autonomous lunar sample collection activity by the SATM instrument on the Victoria Rover. Using these numbers in the model, the (autonomous) SATM object calculates the actual activity times dynamically, based on the sample collection depth and sample volume parameters provided externally (i.e. by the rover command).

Table 8-5. SATM collecting lunar surface sample activity times (from Honeybee Robotics, Ltd.)

Activity	Duration
Lower its augur to the surface	3.5 min, (10 cm/min), (0.18 Watt/Hr)
Drill augur nominal drill rate into the lunar surface	4 cm/min @ 12 Watts, 150 rpm, 10 lbs thrust
Open/close sample cavity door	1 min, (2 rpm), (0.05 Watt/Hr)
Retract the augur from lunar surface	7 cm/min
Move augur to start position above lunar surface	10 cm/min
Drop sample on Carousel	5 min (move to carousel, open sample door, take sample out, close sample door, move augur back). It will have an acceleration profile, but as it approaches the drop off interface on the carousel it will slow to about 1 cm/min

⁶³ Sample Acquisition and Transfer Mechanism

⁶⁴ Honeybee Robotics, Ltd. are the designers and makers of the SATM instrument.

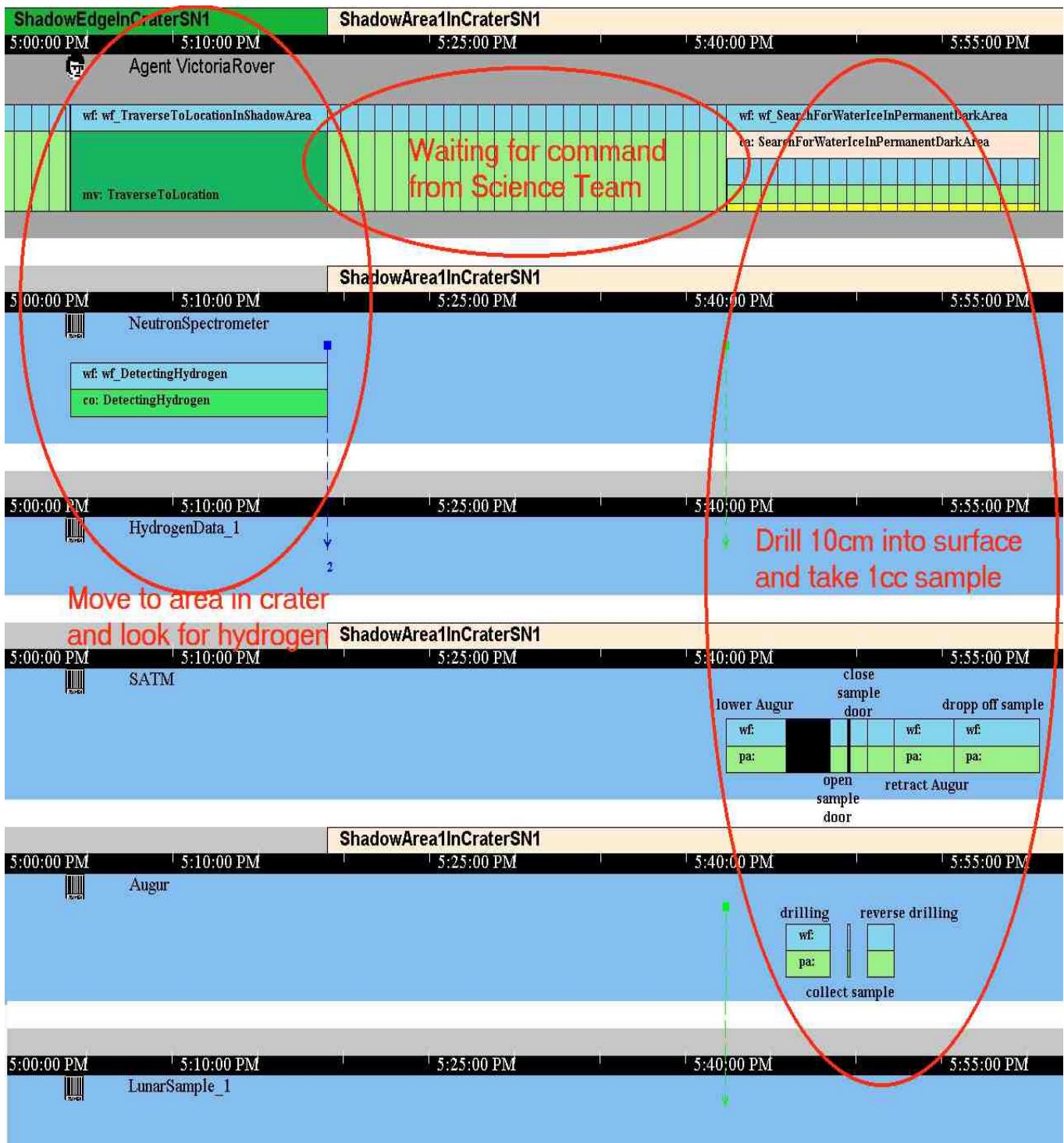


Figure 8-7. Victoria Rover scenario activities

Activities of the VictoriaRover agent are shown at the top, above the activities of the NeutronSpectrometer and SATM instruments. The HydrogenData_1 object is created by the NeutronSpectrometer, which represents the hydrogen data that it found and communicates back to Earth (the communication to Earth is not shown in this figure). The Augur object creates the LunarSample_1 object in the "collect sample" activity. This object represents the 1cc lunar sample that was captured during the drilling activity. The VictoriaRover commands the drill to start, getting its command from Earth, but then the drill performs the activity autonomously.

Table 8-6 gives the values of the list of activities and activity simulation time output variables for the Victoria rover, from the simulation shown in Figure 8-7. The complete list of Victoria Rover activities is comprised of all the parallel activities of the rover agent and the instruments used during the rover's activities. Table 8-6

shows the activities for the VictoriaRover agent, the NeutronSpectrometer object, the SATM object, and the Augur object contained in the SATM object.

Table 8-6. VictoriaRover and Science Instruments activities and times in seconds

	Activity	StartTime	EndTime	TotalTime
VictoriaRover	DoNothing	0	7514	7514
	ProcessUplinkData	7514	7516	2
	TraverseToLocation	7516	8416	900
	CommunicateToEarthTeam	8416	8418	2
	DoNothing	8418	9815	1397
	ProcessUplinkData	9815	9817	2
	SearchForWaterIceInPermanentDarkArea	9817	10914	1097
		Total	10914	
NeutronSpectrometer	DetectingHydrogen	7516	8416	900
	AddDataTypeToDataObject	8416	8417	1
			Total	901
SATM	ExtendAugur	9817	10027	210
	Drilling	10027	10180	153
	StopDrilling	10180	10181	1
	OpenSampleDoor	10181	10241	60
	StartSampleAcquisition	10241	10242	1
	Drilling	10242	10251	9
	StopDrilling	10251	10252	1
	CloseSampleDoor	10252	10312	60
	ReverseDrilling	10312	10404	92
	RetractAugur	10404	10614	210
	DropOffSampleOnCarousel	10614	10914	300
		Total	1097	
Augur	MovingIntoSurface	10027	10180	153
	SampleAcquisition	10242	10251	9
	MovingOutOfSurface	10312	10404	92
		Total	254	

The total duration of the complete scenario is given by the total time of the VictoriaRover's activities. Table 8-6 shows a total scenario duration time of 3 hrs and 2 min (10914 sec). This includes the *DoNothing* activity of the rover at the start of the scenario. During this activity the human teams on Earth are working towards a decision on the first rover command. This will be discussed in the next section. The total duration of the traverse into the permanent dark area, including the extraction of one lunar sample is 57 minutes (see Table 8-7).

Table 8-7. VictoriaRover agent output variables

	Actual Value	Hrs and Min	Length Traveled	Data Type
Total Duration Time	10914 sec	3 hrs, 2 min		
Duration For Rover (without the first DoNothing activity)	3400 sec	57 min		
Rover Traverse	900 sec	15 min	900 m	
Data Transmission To Earth	2 sec			Hydrogen data

8.8 TEAM ACTIVITIES

The SOT is a group of mission scientists who, at the start of each traverse of the rover, have some science objective they want to accomplish. The SOT is at the center of the decision making process for a specific mission objective. The SOT is the team that decides what activity the rover is to execute next. To do this, the SOT needs help from the DAIT. The DAIT team's responsibility is to make sure that the SOT has the appropriate mission data available to make a decision. Analyzing such mission data is a collaborative activity between the SOT, the DAIT, and the rover. The rover gathers data and sends it to Earth for analysis. The DAIT locates the correct data, which is then presented to the SOT. Together these two teams analyze the data, given the current mission objective and state. When the SOT has decided what the rover is to do next, it communicates this to the VSOT. This team consists of technical individuals who know how to operate the robot. The VSOT takes the decision of the SOT, and creates a command sequence for the robot. Using the teleoperation system, the VSOT executes the command sequence, which is then sent to the rover via the USN satellite dish. The rover is responsible for executing the activities as specified in the command sequence, as well as communicating its findings back to the Victoria Team. The next two sections explain how the scenario is simulated, given the work system design model.

8.8.1 Uplink process

Figure 8-8 shows how the uplink process for the *search for water ice in the permanent dark crater* scenario is simulated. The scenario starts with the DAIT team retrieving the Clementine data image of the shadow edge area, where the rover is located at the start of the scenario—this data is modeled as an static Clementine mission data object, available in the VisualizationSystem object at the start of the simulation. They review this image using the visualization system, which is shown in Figure 8-8 by the communication line at the top-left corner in the VisualizationSystem object.

This first example already shows some work system design decisions that were made: a) deciding that the DAIT team retrieves this image, and b) the fact that they do this without anyone requesting that they look at this data. This means that the DAIT needs to be aware of the location and situation of the rover at all times, as well as that they need to know that this data is available and needs to be retrieved, and where and how they can retrieve it. These activities were designed as a result of the fact that the function of the DAIT team is to perform *long-term planning for science opportunities*, given in Table 8-3.

Once the DAIT has retrieved the images, it communicates this to the SOT team, and they collaboratively analyze these images. This is shown in Figure 8-8 by the AnalyzeRoverImages activity that both the DAIT and the SOT team perform at the same time (see *Team Collaboration* in Figure 8-8). At the end of this analysis the SOT team plans the first rover command sequence (see *Rover Command Decision* in Figure 8-8). According to the scenario being simulated, the SOT decides that the rover needs to drive for a specified amount of time (15 min) into the crater to a specific location (ShadowArea1InCraterSN1), and while driving it should be using its neutron detector instrument to detect hydrogen in the lunar surface. This decision is communicated to the VSOT team (and the DAIT team). After this communication, the SOT waits for the rover's downlink data. The command sequence that is created is represented as the following communicated beliefs, by the SOT:

```
belief: (projects.victoria.VictoriaRover.nextActivity = MoveToLocationActivity)
belief: (projects.victoria.VictoriaRover.subActivity = projects.victoria.DetectHydrogenActivity)
belief: (projects.victoria.VictoriaRover.drivingTime = 900.0)
belief: (projects.victoria.VictoriaRover.gotoLocation = projects.victoria.ShadowArea1InCraterSN1)
belief: (projects.victoria.VehicleAndSpacecraftOpsTeam.transmitCommand = true)
```

Now the VSOT team starts its activity of creating the command sequence. This is done using the teleoperation software system, and is shown in Figure 8-8 by the communication of the command sequence from the VSOT team to the TeleOperationSystem (see *Communicate Command Sequence* in Figure 8-8). How this interaction between the TeleOperationSystem and the VSOT team works is not further specified, but could be seen as a high-level requirement for the development of the actual teleoperation system.

The TeleOperationSystem activity shows the TeleOperationSystem communicating the command sequence to the USN satellite dish UsnDish1, after which the UsnDish1 object communicates this data to the VictoriaRover, waiting at the shadow edge in crater SN1. The VictoriaRover agent receiving this information and acting upon it was described in the previous section on the rover activity (see *Move to area in crater and look for hydrogen* in Figure 8-7).

Figure 8-8 also explains why the VictoriaRover is performing the DoNothing activity at the start of the scenario; the length of the DoNothing activity is determined by the time it takes the Victoria team to collaboratively decide what the next command should be for the rover. All the collaborative activities that are performed during this first part of the simulation are part of the uplink process for requesting, collecting, processing and analyzing data, as shown in Table 8-2. The VictoriaRover is waiting for this uplink process to be completed.

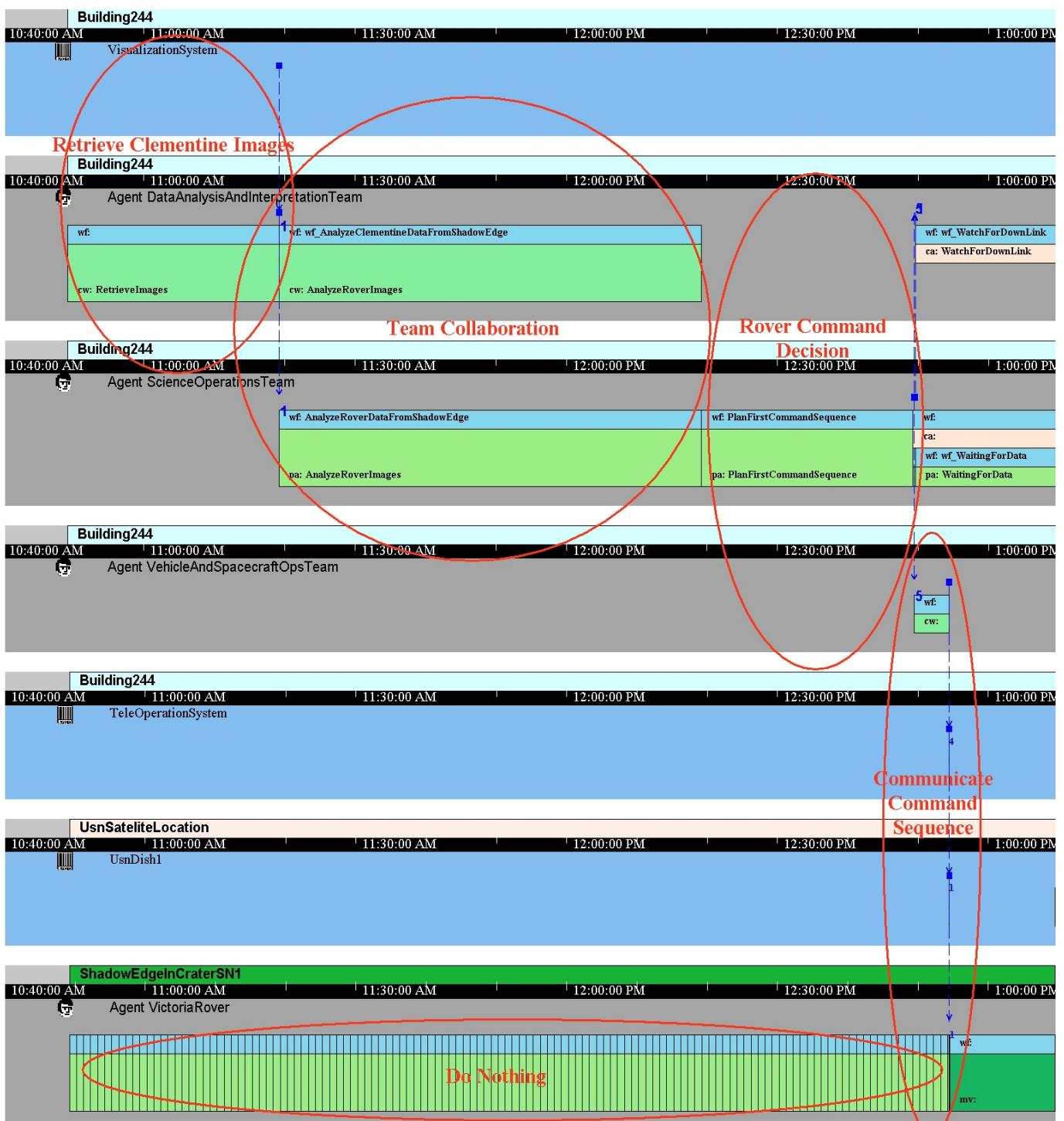


Figure 8-8. Simulation of first uplink command activities

8.8.2 Downlink process

After the rover has received the first uplink command it will start the traverse into the permanent dark crater. When the rover detects hydrogen in the *ShadowArea1InCraterSN1* location the downlink process starts. What happens during the downlink process is shown in Figure 8-9. The VictoriaRover agent contains the S-BandMGA object, which represents the S-Band transmitter on the rover. The VictoriaRover creates a data object with a) the current rover location information and b) the hydrogen data. This data object is then communicated to Earth, via the UsnDish1 object. The UsnDish1 object communicates this data to the DataConversionSystem (see *Downlink Process* in Figure 8-9). As can be seen in Figure 8-9, the DataConversionSystem performs two conversion activities, one for the hydrogen data and one for the location data from the rover. In the work system design, as implemented in the model, it has been decided that the data conversion system should be intelligent enough to handle the data conversion for and the transmission to the visualization system, without human intervention. This creates a certain level of requirements for these systems that have not been specified in more detail in the model, but could have easily been modeled in more detail.

When the VisualizationSystem receives the newly converted data, the system alerts the user, i.e. the DAIT team. This is implemented in the model through the creation of facts that simulate software “alarms” that are being detected by the DAIT agent using detectables (see *Detect, retrieve, interpret and communicate data* in Figure 8-9). This simulates the activities of a member of the DAIT. They are monitoring the VisualizationSystem while in the activity *WatchForDownlink*. This is shown in the activity timeline of the DAIT agent in Figure 8-9. When the DAIT agent detects that there is newly available neutron detector and location data, it retrieves the data from the VisualizationSystem object (i.e. the activities *RetrieveNeutronData*, *InterpretNeutronData*, and *FindRoverLocationData*). This simulates the DAIT team members looking at and interpreting the rover’s neutron and location data, using the visualization system. After these activities are performed, the DAIT team communicates their findings to the SOT. The scenario states that the hydrogen data suggest that the rover has found hydrogen in the *ShadowArea1InCraterSn1*. When the SOT hears these findings, it decides very quickly what the next command sequence for the rover is, and communicates this decision to the VSOT team (i.e. *CommunicateDoDrillActivity*) (see *Next Rover Command Decision* in Figure 8-9). The command sequence that is created is represented as the following communicated beliefs:

```
belief: (projects.victoria.VictoriaRover.nextActivity = SearchForWaterIceInPermanentDarkAreaActivity)
belief: (projects.victoria.VictoriaRover.subActivity = projects.victoria.DrillingActivity)
belief: (projects.victoria.SATM.lengthIntoSurface = 10.0)
belief: (projects.victoria.SATM.sampleVolume = 1.0)
belief: (projects.victoria.VehicleAndSpacecraftOpsTeam.transmitCommand = true)
```

The communication tells the VSOT that they have to transmit the command sequence to the VictoriaRover. The command sequence tells the VictoriaRover to start the *SearchForWaterIceInPermanentDarkAreaActivity*. It also tells the VictoriaRover that its sub-activity during this activity is to perform the *DrillingActivity*. The next commands are parameters for the *DrillingActivity* that the rover needs to know, to a) know how deep to drill and b) know how big of a sample to collect at that depth. Figure 8-9 shows a part of this second uplink process, which is performed in the same way as the first data uplink shown in *Communicate Command Sequence* Figure 8-8.

The length of this downlink and second uplink process determines the length of the second *DoNothing* activity of the VictoriaRover, which simulates the time the rover is waiting for the Victoria science team to decide the next command sequence for the rover (see *Waiting for command from Science Team* in Figure 8-9).

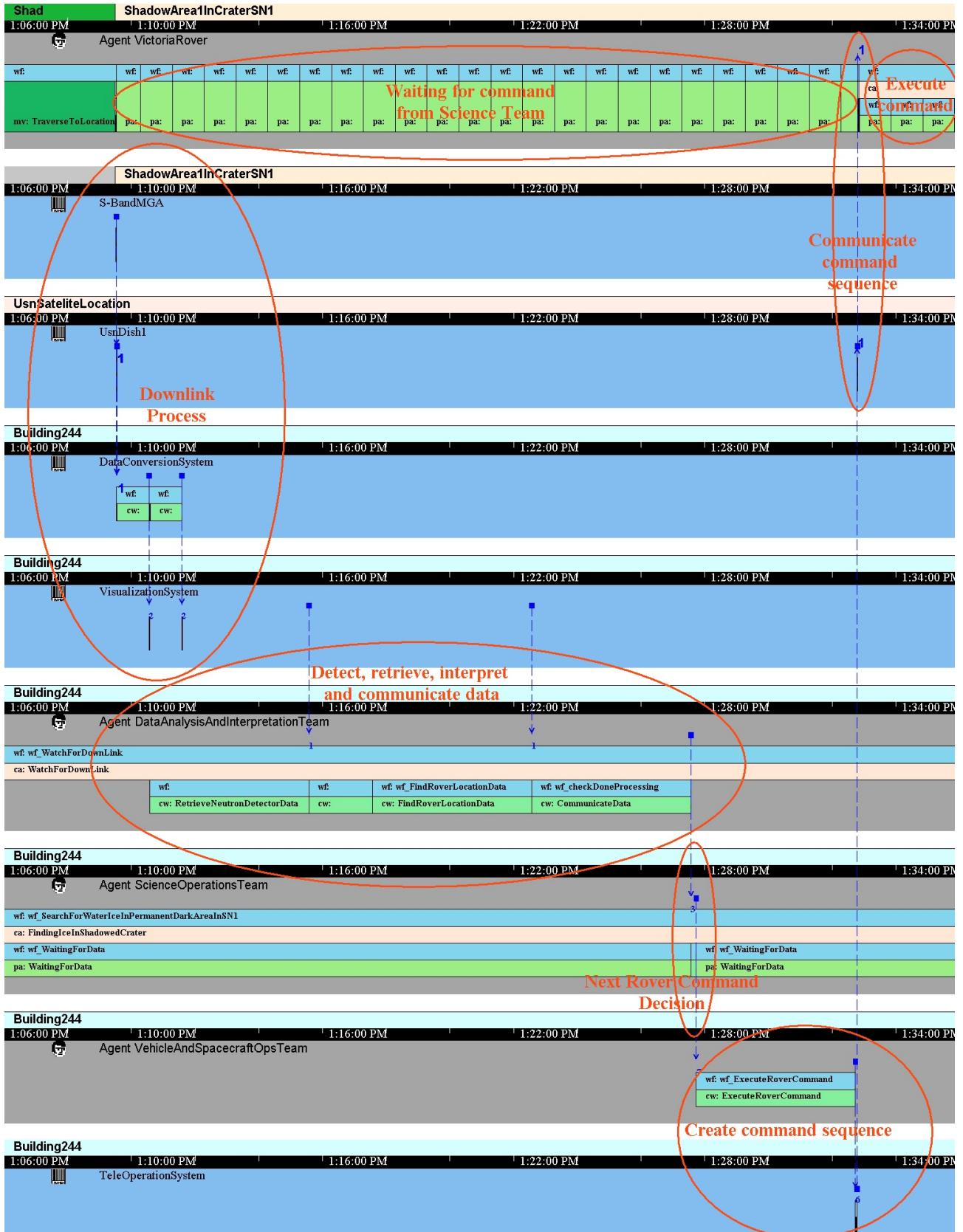


Figure 8-9. Simulation of downlink and second uplink command activities

8.8.3 Output variables for human agents

Given the activities of the Victoria team shown in Figure 8-8 and Figure 8-9, we can generate the output variables for the human agents, i.e. the different Science and Victoria Team agents. Table 8-8 shows the activities and activity times for each human agent in the model.

Table 8-8. Victoria Team activities and time in seconds

Agent	Group	Activity	StartTime	EndTime	TotalTime
DataAnalysisAndInterpretationTeam	ScienceTeam	RetrieveImages	0	1800	1800
		AnalyzeRoverImages	1800	5400	3600
		RetrieveNeutronDetectorData	8481	8781	300
		InterpretHydrogenData	8781	8901	120
		FindRoverLocationData	8901	9201	300
		CommunicateData	9201	9501	300
				Total	6420
ScienceOperationsTeam	ScienceTeam	AnalyzeRoverImages	1800	5400	3600
		PlanFirstCommandSequence	5400	7200	1800
		CommunicateNextRoverActivity	7200	7210	10
		AskToWatchForNewDownlink	7210	7220	10
		WaitingForData	7220	9501	2281
		CommunicateNextRoverActivity	9501	9511	10
		WaitingForData	9511	10914	1403
				Total	9114
VehicleAndSpacecraftOpsTeam	VictoriaTeam	ExecuteRoverCommand	7210	7510	300
		ExecuteRoverCommand	9511	9811	300
				Total	600

8.9 CALCULATING ROVER ENERGY USAGE

This section describes how the output variable for *rover energy usage* (e.g. E_{acti} from (2.0) in section 8.2.2) is calculated during the simulation. The reason for describing the techniques used in more detail is a) because this variable is of particular interest for judging the quality of the work system design for the Victoria mission, and b) the Brahms programming techniques used to model this variable is an important technique to understand for future modelers.

To calculate the total power consumption of the rover during the scenario we need to calculate the energy being used over time. This is done by identifying the energy usage during every primitive activity of the rover, based on each subsystem and instrument on the rover requiring power during a specific activity. Using equation (2.0) we can calculate the energy usage during each rover activity. The *total power consumption* of the rover during the scenario can then be calculated by adding all the energy usages for each rover activity:

$$\text{Total Power Consumption} = \sum_{i=0}^n E_{acti} \quad (4.0)$$

shows the data calculated from equations (2.0) and (4.0). The energy usage during the rover's activities consists of the energy used by the rover, its subsystems, the Neutron Spectrometer and SATM.

Table 8-9. Energy usage for the rover during scenario

	Activity	Location	Power Needed For	Eact_i (W/hr)
VictoriaRover	DoNothing	Edge of crater SN1	N.A. ⁶⁵	
	ProcessUplinkData	Edge of crater SN1	N.A. ⁶⁵	
	TraverseToLocation	Shadow crater SN1	Thermal Protection during driving + Mobility during driving + Altitude Determination during driving + Command and Data Handling during driving	99.06
	CommunicateToEarthTeam	Shadow crater SN1	Thermal Protection during driving + Command and Data Handling during driving + Ground Link during driving	86.84
	DoNothing	Shadow crater SN1	Thermal Protection during driving + Command and Data Handling during driving	33.96
	ProcessUplinkData	Shadow crater SN1	Thermal Protection during driving + Command and Data Handling during driving + Ground Link during driving	0.37
	SearchForWaterIceln PermanentDarkArea	Shadow crater SN1	Thermal Protection during science + Command and Data Handling during science	49.52
Neutron Spectrometer	DetectingHydrogen	Shadow crater SN1	Take Spectral Image	0.02
	AddDataTypeToDataObject	Shadow crater SN1	N.A. ⁶⁶	0
SATM	ExtendAugur	Shadow crater SN1	Move Augur Platform	0.18
	Drilling	Shadow crater SN1	Nominal Drilling	30.60
	StopDrilling	Shadow crater SN1	N.A. ⁶⁶	0
	OpenSampleDoor	Shadow crater SN1	Open Chamber Door	0.05
	StartSampleAcquisition	Shadow crater SN1	N.A. ⁶⁶	0
	Drilling (reverse)	Shadow crater SN1	Nominal Drilling	1.80
	StopDrilling	Shadow crater SN1	N.A. ⁶⁶	0
	CloseSampleDoor	Shadow crater SN1	Close Chamber Door	0.05
	ReverseDrilling	Shadow crater SN1	Nominal Drilling	18.20
	RetractAugur	Shadow crater SN1	Move Augur Platform	0.18
	DropOffSampleOnCarousel	Shadow crater SN1	Sample Drop	0.20
Total Power Consumption during scenario				321.03 Watt

To implement the calculation of the energy usage for the rover, I needed to operationalize the calculation of the energy needed for each subsystem during a rover activity (see Table 8-9). Here I show the way this is done using the simplest rover activity as an example. From Table 8-9 we can see that the energy the rover uses during the DoNothing activity is defined by the energy needed for Thermal Protection during driving + Command and Data Handling during driving. What this means is that even while the rover is standing still and “doing nothing,” it consumes power for its thermal protection and its commanding and data handling for its subsystems, such as its processor board. The rover designers⁶⁷ were able to provide the power consumption specification for these power consuming low-level activities of the rover (Table 8-10).

⁶⁵ This energy is not added to the total energy used, because this before the rover starts the traverse in the permanent dark area. The battery is charged to full capacity at the moment the traverse begins.

⁶⁶ There is no energy usage associated with this activity.

⁶⁷ The Robotics Institute at Carnegie Mellon University; <http://www.ri.cmu.edu/centers/frc/index.html>

Table 8-10. Rover power consumption data

Subsystem	Permanent Shadow Traverse w/growth (W)	Permanent Shadow Science Investigation w/growth (W)	Shadow Hibernation w/growth (W)
Thermal			
Thermal Radiator	0	0	0
Multi-layer Insulation	0	0	0
Thermal Switches			
Thermal Coatings	0	0	0
Temperature Sensors	0.225	0.225	0
Heaters	39.375	50.625	84.375
Thermal Total	39.6	50.85	84.375
Command & Data Handling			
Processor Board	24.48	24.48	12.24
Non-Volatile Memory Board	3.6	3.6	1.8
Command & Telemetry Board	3	3	0
ADCS/Payload Interface Board	3	3	0
Motion Control Board	7.722	1.755	0
Power Distribution/Propulsion Driver/Heater Control Board	2.4	2.4	2.4
Charge Control/Array Switching	0	0	0
DC-DC Converter	4.8	4.8	4.8
Hardware Box	0	0	0
Command & Data Handling Total	49.002	43.035	21.24

© The Robotics Institute, CMU

The power consumption data is represented for the VictoriaRover agent as initial-beliefs (so that the agent can use them) and initial-facts (so that the instrument objects can use them), using six attributes of type *double*:

```
agent VictoriaRover memberof Rover, DataCommunicatorGroup {
    initial_beliefs:
        (current.powerNeededForThermalProtectionDuringShadowDriving = 39.6);
        (current.powerNeededForThermalProtectionDuringShadowScience = 50.85);
        (current.powerNeededForThermalProtectionDuringShadowHibernation = 84.375);
        (current.powerNeededForCommandAndDataHandlingDuringShadowDriving = 49.002);
        (current.powerNeededForCommandAndDataHandlingDuringShadowScience = 43.035);
        (current.powerNeededForCommandAndDataHandlingDuringShadowHibernation = 21.24);
    initial_facts:
        (current.powerNeededForThermalProtectionDuringShadowDriving = 39.6);
        (current.powerNeededForThermalProtectionDuringShadowScience = 50.85);
        (current.powerNeededForThermalProtectionDuringShadowHibernation = 84.375);
        (current.powerNeededForCommandAndDataHandlingDuringShadowDriving = 49.002);
        (current.powerNeededForCommandAndDataHandlingDuringShadowScience = 43.035);
        (current.powerNeededForCommandAndDataHandlingDuringShadowHibernation = 21.24);
}
```

Using these beliefs (and facts) the Brahms model can calculate the energy used during the *DoNothing* activity. This is done in the workframe *wf_Waiting*:

```

workframe wf_Waiting {
    repeat: true;
    variables:
        forone(double) hourlyratio;
        forone(double) thermalprotectionpower;
        forone(double) commanddatahandlingpower;

    when (knownval(hourlyratio = 60 / 3600) and
          knownval(thermalprotectionpower = current.energyNeededForThermalProtectionDuringShadowDriving
                     * hourlyratio) and
          knownval(commanddatahandlingpower =
                     current.energyNeededForCommandAndDataHandlingDuringShadowDriving * hourlyratio))
    do {
        DoNothing(0, 60);
        conclude((current.energyUsedInActivity = thermalprotectionpower), bc:100, fc:100);
        conclude((current.energyUsedInActivity = current.energyUsedInActivity + commanddatahandlingpower),
                 bc:100, fc:100);
        conclude((VictoriaRover.consumedEnergy = true), bc:0, fc:100);
    }
}

```

Every 60 seconds, this workframe calculates the energyUsedInActivity belief and fact, based on the energy used for the thermal protection and the command and data handling for the duration of the DoNothing activity (60 sec). The model includes this type of calculation in every workframe for the VictoriaRover agent, the Neutron Detector object, the SATM object, and the Augur object. Figure 8-10 and Figure 8-11 show the calculated data from the Brahms MS AccessTM68 simulation history database. Using simple database queries in MS ExcelTM68, I was able to create the bar graphs showing the Energy level calculated in each workframe. The numbers in Figure 8-10 and Figure 8-11 correspond with those in Table 8-9.

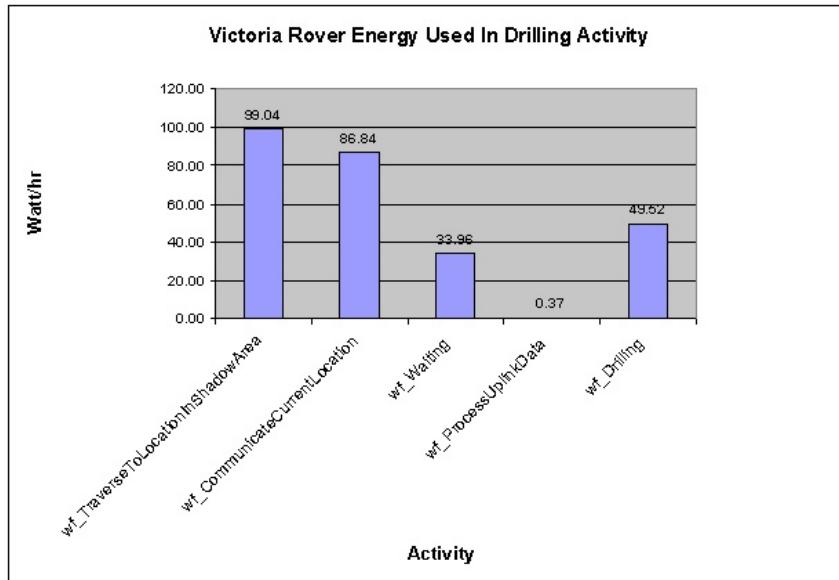


Figure 8-10. Rover energy used in drilling activity from simulation history database

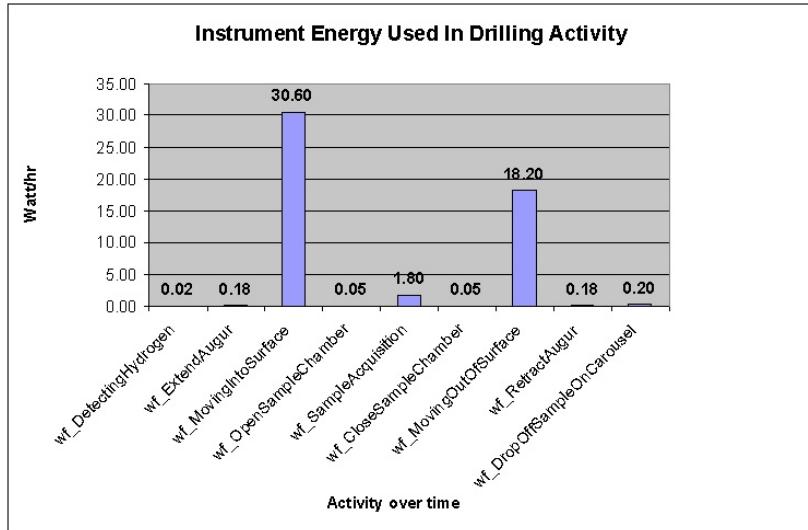


Figure 8-11. Instrument energy used in drilling activity from simulation history database

Besides the ability to show the energy usages per activity, it is also possible to generate a line graph representing the overall rover energy usage during the traverse into the crater. This is given in Figure 8-12.

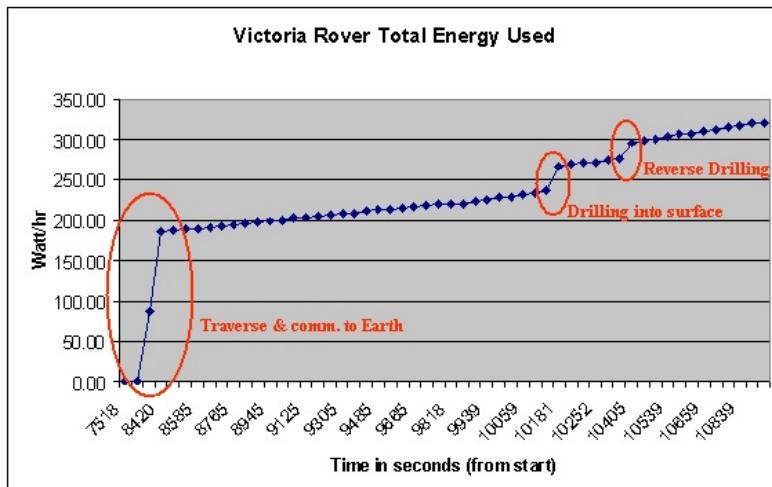


Figure 8-12. Rover total energy usage during traverse into crater

Up to this point, I have described how we calculate the first six output variables from Table 8-1. Another important aspect is being able to determine if the model adheres to constraint (3.0). To determine this, I modeled the battery of the rover (object RoverBattery) to keep track of the power drainage during the rover's activities. Each time the rover agent or one of the instrument objects calculates the *energyUsedInActivity* value for a specific activity, it triggers the RoverBattery object to calculate how much power there is left in the battery. The calculation of the *powerLeftToUse* attribute of the RoverBattery object allows us to show if the current work system design model violates the energy constraint (3.0) (see Figure 8-13).

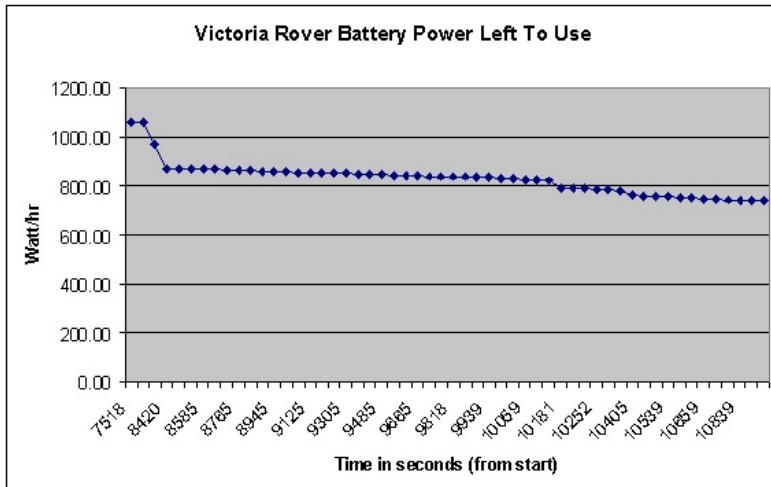


Figure 8-13. Battery power left, based on constraint (3.0)

Besides the power left to use after the scenario, another interesting variable is the *energy usage rate* by the rover.

$$\text{EnergyRate} = \text{Total Power} / P_{\text{battery}}(\text{start of traverse}) \quad (5.0)$$

Figure 8-13 tells us that given the energy used in the scenario—drive 900m into the crater, and take one 1.0cc sample at 10cm depth—with the current work system design, the robot has used almost a third of its power:

$$\text{EnergyRate}_{\text{drilling in permanent dark crater}} \sim 0.30$$

This variable represents the *rover power consumption effectiveness* of the work system design, and is a measure that can be used to compare different work system designs for a model scenario.

This concludes the description of the model and the simulation output. In the next and last section, I will conclude with some observations and give an answer to the research question stated in the beginning of this chapter.

8.10 CONCLUSION

This third and last case study investigates the use of Brahms in a design activity. Design is an activity in which the designer develops a model of a future system. In that sense, the design model *prescribes* the future system. Developing a prescriptive model is very similar to developing a predictive model, because we can view a prescriptive model as predicting the future behavior of a system. However, there is a major difference between these two modeling activities. In a prescriptive model the system being modeled does not yet exist. This has certain consequences for the use of the model.

First, it means that a *design scenario* should drive bottom-up development of the model. Without a realistic scenario the design activity can only be done top-down. Secondly, the value of a design model in a design project is subjectively based on the aid provided to the designers. This is an often overlooked, but very important aspect of using modeling and simulation in a design activity. The questions of what should be modeled, and what is the outcome of such a model are directly related to the aid that the model should provide to the designers. Finally, the difference between prescriptive and predictive models changes the way we can verify and validate the model. A predictive model can be validated based on predictions of past events. By showing that a predictive model can predict past events we can become confident about the predictions of future events, which can later be validated. A validated model of an existing system can be used to predict the behavior of that system in the future. In a prescriptive model we do not have this ability. There is no ability to validate the design upfront. To validate the design we will have to implement the model in the real world. At that moment the prescriptive model becomes a predictive model, and we can use the

same validation approach as with predictive models. However, to implement a prescriptive model without a good feeling about the validity of the model requires a costly leap of faith on the side of the engineers.

Good design tools are hard to come by. Good design tools for the design of work systems are almost non-existent. Having described this case study and the developed Victoria model in the previous sections, I will describe here some of the findings in performing this prescriptive modeling activity, and in doing so, will try to answer the research question that was posed in the first section of this chapter.

The research question, here restated, was:

Can the Brahms modeling and simulation environment be used to prescribe a realistic work practice in the design of the human-robotic collaboration during a robotic lunar mission?

8.10.1 Answering the research question

Described in section 8.3, an objective evaluation of the use of Brahms for a descriptive design model can be done by evaluating to what extent the model is able to generate the defined outcome variables. Because the outcome variables have been defined by the outcome measures that were defined based on the modeling objective (see Figure 8-2), we can evaluate the use of Brahms for this objective by evaluating if and how we can calculate such outcome variables in a Brahms model. Table 8-11 presents how the Brahms model was able to provide the outcome variables, defined for the Victoria model objective.

To evaluate the ability to calculate each output variable from the simulation of the model, I use the following criteria:

1. Is the variable calculated in the model, and therefore, do we need to use the power of the Brahms language to perform the calculation?
2. Is the variable calculated as part of the Brahms simulation, and therefore, can we display the variable by executing a pre-specified SQL-query on the simulation history database that is created by Brahms?
3. Do we need to calculate the variable post-simulation, by executing a pre-specified SQL-query on the simulation history database that is created by Brahms?

In terms of ease of modeling, we can see that the easiest way to calculate a simulation outcome variable is the case in which we can use method (2) to show the result. Calculating a variable using method (2) means that we can use the intrinsic power of the history of the simulation events, captured by the Brahms simulation engine. The modeler does not have to perform any additional work to be able to generate these outcome variables from the simulation. The simulation engine keeps track of these measurements automatically. In both (1) and (3) the modeler has to perform extra work. With method (3) the simulation engine keeps track of most of the measurements, but the modeler needs to do some specific SQL development after the simulation is complete. Therefore, the ability for the simulation engine to provide the measurements without any extra modeling work depends on the type of data that can be extracted from the history database. In case the engine does not provide the needed data “for free,” the modeler needs to use method (1). This means that the modeler needs to add specific variable calculation code to the model itself. This makes the modeling effort more complex on the one hand, but allows the modeler to extend the possible outcome measurements, and thus provides the modeler with a flexible approach.

Table 8-11. Outcome variable evaluation

Output Variables	Variables Calculated in Brahms Model	How Calculated/Displayed	Method
A list of activities for each human-agent	No specific variable calculated in the model	Displayed by executing a pre-specified SQL-query on the simulation history database	2
A list of activities for each robot-agent	„	Displayed by executing a pre-specified SQL-query on the simulation history database	2
The total duration of a high-level mission objective activity	„	Calculated by executing a pre-specified SQL-query on the simulation history database	3
The duration time of each specific agent-activity	„	Displayed by executing a pre-specified SQL-query on the simulation history database	2
The total power consumed by the robot agent The energy used in each robot activity The battery power left after each robot activity	RoverBattery.energyUsed [VictoriaRover EnergyConsumer].energyUsedInActivity RoverBattery.energyLeftToUse	Calculated in workframes associates with the agent or object and displayed by executing a pre-specified SQL-query on the simulation history database	1
Data send to the robot	No specific variable calculated in the model	Displayed by executing a pre-specified SQL-query on the simulation history database	2
For each data transmission made by the robot, its type and the amount of data	„	„	2
Length and time of traverses, Number of mission relevant stops	„	„	2

Table 8-11 shows us that using Brahms I was able to calculate every outcome variable that was defined for the given modeling objective. This gives us an objective answer to the research question, namely:

Brahms can be used to prescribe a realistic work practice in the design of the human-robotic collaboration during a robotic lunar mission.

In this case study the objective was to investigate the use of Brahms in as a prescriptive modeling environment. Although the conclusion is that Brahms can be used as a modeling tool for the design of mission operations work systems, the question remains if such a project is worth the time and effort. Quantifying the added value and cost-benefit of a Brahms modeling project would require a focused effort, which falls outside of this thesis. However, in the next two sections, I present some of my personal evaluation of the perceived values and benefits in this case study.

8.10.2 Added value of Brahms

In this section I discuss the added value of using Brahms in the design of the Victoria MOS. The Victoria mission proposal was not selected for funding in 2001, therefore the results from this case study did not feed back into the next mission design cycle. However, there are some subjective benefits that can be discussed.

Relationship between rover design and work system design

The Brahms simulation study operationalized the inherent relationship between the design of the rover and instruments and the design of the total mission operations work system. Everyone involved in the Victoria proposal was convinced that this relationship existed and played a role in the total design of the mission. However, current mission design tools are not capable of showing these relationships. Therefore, the analysis of this relationship and the impact on the design of the systems are currently only done informally by the mission designers. Design decisions that are based on these relationships are currently made based on the experience of the mission designers involved. Since every NASA mission is considerably different from previous missions, these experiences are often not based on high confidence data.

In this case study, I have shown that with Brahms we are able to operationalize the relationships in a simulation of both the work system and the hardware and software systems. This shows a tremendous potential benefit in mission design.

Science return

The mission designers proposed that the Victoria rover could spend about 2 hours within a permanent-dark crater. They proposed that within these two hours they could make several stops in the crater, and take lunar samples. The SATM designers pride themselves in the proposal of being able to drill to a maximum of 1m into the lunar surface, and take a 1.0cc sample.

However, these numbers are not substantiated based on a simulation of the permanent-dark crater mission scenario in which the work of the Earth-based teams was taken into account. The reason for this is simply because there are no tools available to do such analysis. The only way to get somewhat accurate scenario data is by re-enactment of the scenario in field tests. Such field tests are a) not possible until all or some of the mission systems have been developed, or are at least in prototype stage, b) difficult to setup, c) time consuming, and d) costly. Therefore, at the mission proposal stage there is currently no capability to perform any scenario re-enactment.

This case study shows that Brahms could be used to perform such scenario re-enactment virtually through simulation of the scenario, based on a work practice model of the MOS. This capability would provide mission designers with better tools to develop more realistic mission scenarios, and thus could do a better job in quantifying the possible science result in a mission.

Rover battery requirements

Currently, the battery design for the Victoria rover is developed based on weight and volume constraints for the rover, and on the power consumption of the rover's systems and instruments during the mission. Whether the Victoria mission would ever be successful depends on the amount of science that can be done during the mission, given the capability of the total MOS, including the capabilities of the rover and the Earth-based teams.

This case study showed that the amount of science that can be done in a mission into a permanent dark crater is less than was proposed in the mission proposal—there is about 25% less time available to do science. Although the model is incomplete, and the study was only a preliminary study to show the capability of Brahms, it can be said that having this modeling and simulation capability provides the mission designers with a tool that can help evaluate design decisions and generate more accurate design requirements for hardware and software systems for the mission.

In a personal conversation with the PI of the Victoria mission, the PI told me that he would now consider redesigning the battery system for the rover. This is strong testimonial evidence that this case study added value to the Victoria proposal.

High-level requirements for mission support systems

Similar to the generation of requirements for the rover hardware, the model simulation showed the relationships between the data uplink and downlink activities of the human teams, the rover, and the needed software support systems part of the MOS. The model does not go into details of user interface design, but does show the relationship between human activity and the software system's functionality requirements in support of these activities. A Brahms simulation model is useful as a software requirement specification of how the software systems fit in a human-centered work system.

Mission field test support

NASA Mission designers a) design MOS and b) test the MOS in time-consuming and costly field tests. Using Brahms to create initial designs and provide data before and during such field tests would provide the mission designers with an evaluative capability that feeds back valuable data from the field tests to the mission design efforts.

To highlight this benefit, I briefly describe a real-life problem that is currently being addressed in the '03 MER mission, being designed at JPL. I then discuss how the '03 MER mission designers are solving the problem, and contrast this with how a Brahms model and simulation effort could benefit the mission designers:

The '03 MER mission is a planned 2003 mission to Mars with two identical Athena rovers. One of the mission operation work system design problems has to do with having the Earth-based human teams use Mars-time or Earth time during this long-term mission (90 days or longer). The issue is that a Martian day—sol—is 24 hours, 39 minutes in Earth time. Mission designers have created a work group, which has a charter to study the advantages and disadvantages of working Earth time versus Mars time, versus some combination of these times, on personnel who work the duration of the mission. The working group must understand the problems inherent with the different operation methods and deliver a detailed report and recommendation, within two months.

Without going into much detail, some of the attributes of the mission that need to be considered by the work group are:

- Provide capability for commanding the rovers every Martian sol.
- Provide margin for dealing with contingencies within the nominal timeline.
- Minimize the impact on personal lives.
- Ensure information of key information across shift boundaries.

- Resiliency in face of anomalies.
- Support operation for two rovers simultaneously.
- Maximize the potential for science return.

These are some of the issues that the work group needs to address in their recommendation to the mission commander. Their current approach is to divide the work group into two advocate teams—Pro-Mars and Pro-Earth—that will each independently research the pros and cons from their points of view. After this, they will present their findings and defend their group positions. They will then come together and adopt the best of both plans, compiling a report with recommendations.

The work group could benefit from a Brahms simulation of the '03 MER mission operations work system, by running separate what-if scenarios based on a model that implements the Pro-Mars and Pro-Earth team's design in the Brahms model. For each possible design, the Brahms model could generate appropriate variables that allowed the work group to compare the different designs.

Such a use of Brahms would be extremely useful for the '03 MER mission if the model would be available at the start of the task of the work group, and if the changes to the model, representing the different team designs, could be implemented in at most a couple of weeks. If it is possible to create such a MER model in Brahms is an empirical question, but the Victoria case study is positive evidence that this is possible, and shows that the potential benefit for the '03 MER mission is high. Today, the work group has no tools available to support them in this difficult task.

8.10.3 Cost-benefit of using Brahms

In this section I discuss the potential cost of using Brahms in the design of the Victoria MOS. Given the added value of a Brahms simulation, I discuss the cost of developing the Victoria model. There are a number of important criteria that should be mentioned: a) the model that was developed was a relatively small model and did not include all the necessary detail for a real modeling effort, b) the model was developed by one individual, outside of the mission proposal team, c) the modeler was an experienced Brahms user, and d) the modeler was able to re-use some model libraries from the previous Apollo case-studies. Although, there is no detailed cost data available, the cost of the Brahms modeling effort in this case study could be defined as follows:

Model development time

The largest amount of time for developing the model was in creating the initial design of the work system as a conceptual model (about 70% of the total modeling time). Such a design needs to be done with or without the use of Brahms, and is thus not dependent on the use of Brahms.

The amount of time it took to develop the Brahms model (less than one man-month) is small in relation to the design and development of the total mission (at least 4 years).

Cost benefit for mission field tests

Using Brahms to create initial designs and provide data before and during such field tests would cut down the field test time, and thus would create a measurable cost reduction in the total mission.

This concludes the description of the last case study. In the next chapter, I give my overall conclusions for the research presented in this thesis.

9. CONCLUSIONS

This thesis described a work practice modeling and simulation methodology, based on the Brahms language and simulation environment. The Brahms language is the result of research on how to model and simulate work processes. I argued that to assess a work system's quality, modeling the informal and circumstantial interactions is essential. Such a model is referred to as a *model of work practice*. I described a theory of modeling work practice that includes the representation of collaboration between people and systems, communication and interaction, all while being located in a geographical environment. This theory is operationalized in the Brahms modeling language and simulation environment. To verify and validate the theory I aimed to show that the Brahms language is complete and sufficient to represent human behavior at the work practice level. I performed three case studies that show the different ways of using work practice simulation models; 1) describing an existing work practice, i.e. the Apollo 12 ALSEP Offload, 2) predicting future agent-behavior based on a general model of work practice, based on the Apollo 15 & 16 missions, and 3) designing a new work practice, i.e. the mission operations of a proposed robotic mission to the Moon.

To recap what has been presented in this thesis, in the first chapter I introduced the problems with modeling work processes based on workflow modeling, and stated the two research questions. Part 1 of the thesis discussed several existing human behavior modeling approaches. Besides workflow modeling, I discussed related cognitive approaches, distributed-AI systems that include a multiagent approach, and systems from computational organization theory. I argued that none of these approaches and systems include all the relevant aspects for modeling at the work practice level. I then described the theory for modeling work practice and the Brahms language and simulation environment. Part 2 presented the research design and the three case studies in detail.

In this last chapter, I present conclusions based on my experience in designing Brahms and applying it to real-world domains. First, I reflect on the two research questions and to what extent they have been answered. Next, I describe the contributions that have been made to the scientific communities of modeling and simulation, agent-based systems, and a new emerging field called human-centered computing. I end with some comments on future research topics on work practice modeling and simulation.

9.1 REFLECTIONS ON RESEARCH QUESTION

I started this research by stating two research questions (see Chapter 1.2). Here I reflect on these two questions, with the goal of describing to what extent I have been able to answer them. I will show that with using Brahms in the three case studies, I have successfully answered the first question. I will then show that, while developing and applying the Brahms environment for answering the first question, I have also been able to answer the second question.

9.1.1 Research question 1

The main research question was stated as follows:

How can we model an organization's work practice in such a way that we include people's collaboration, "off-task" behaviors, multi-tasking, interrupted and resumed activities, informal interactions, and geography?

In chapter 2, I explained how other human behavior modeling systems and approaches lack, in some fundamental way, the ability to include the aspects of work practice that are mentioned in the research question. The challenge for Brahms then is to include symbolic representations for these aspects. While some aspects can be represented as part of the model, other aspects—interrupt and resume, "off-task" behavior, informal interaction—are shown as emergent phenomena during a simulation. I answer the research question by describing how Brahms represents each of the aspects, based on the experience I gained in the case studies.

1. Collaboration

In chapter 3.2.3 I defined collaboration in two ways. First, as a mental awareness by the people involved in the collaboration. This awareness does not necessarily have to exist at the same time, in the same place, and in the same way for every individual in the collaboration. However, the awareness is created at the moment the collaborators are in their individual activities, making them feel they are collaborating. Secondly, I defined a form of collaboration I call *indirect* collaboration. Such type collaboration exists when the collaborators are not directly aware of the collaboration. For example, a company sales representative adds an order to the order database, which is picked up downstream in the order process by another employee to act upon. In short, collaboration integrates the activities of individuals in a group as a whole, a *collaborative activity* with a purpose, and thus over time establishing a community of practice.

In Brahms we can represent collaboration between agents (and/or objects) in three fundamental ways. First, we can describe the dependencies between agents' activities in terms of the result of one agent's activity (i.e. changes in the world state) triggering another agent's activity. An agent's interaction with and movement of objects, and the use of these objects in subsequent activities explicitly represents this. For example, in the Apollo 12 model the simulation shows the commander moving the first ALSEP package out of the way, so that the lunar module pilot can finish lowering the second package to the ground.

Secondly, the Brahms language allows for explicit representation of communication between agents (and/or objects). During a simulation, agents can react to received communications from other agents or objects. Brahms allows the modeler to represent send and receive agent (and/or object) communication activities. This provides not only the ability to show agents talking to each other, but also an agent reading from or writing to an object, which allows for representing indirect communications between people. Another important aspect of modeling communication in Brahms is the ability to model the communication devices (such as the voice loop and the communication time delay) and the practice of using them.

Third, the ability to represent an agent's beliefs about other agents' activities, location, and beliefs. This allows the modeler to represent that an agent will or will not perform a certain activity, based on the beliefs about another agent's state. For example, in the HFE deployment model the commander agent knows the location of the lunar module pilot agent and is aware of its activity at the moment it puts the ALSEP packages onto the lunar surface. This awareness is part of the reason the commander agent starts its activity of driving the lunar rover to the ALSEP deployment area. The other reason is the plan on his cuff checklist.

Given the ability to model these aspects of agent interaction, I conclude that Brahms allows the modeler to represent the necessary parts of collaboration and explain the emergent result of collaborative activity during a simulation. Whether Brahms' ability to model collaboration is sufficient is an empirical question. I was able to represent the collaboration in the three case studies, and therefore I conclude that the Brahms language is sufficient for modeling collaboration as shown in the case studies.

2. Interrupt and resume

It is very natural for people to be interrupted while in an activity, such as being in a meeting with someone when the telephone rings. A person can easily interrupt the meeting conversation, answer the phone and engage in a completely separate conversation. When finished with the telephone conversation, the person can easily resume the meeting where it was left. The Brahms workframe-activity subsumption architecture is based on this very principle (see sections 4.4.7, 4.6.2, and 4.6.4).

This means that the modeler does not have to specify when or how workframes or activities can be interrupted. It is an inherent property of the agent's activity execution space. Thus, Brahms handles people's natural behavior of interrupt and resume of activities. Specifically, the general error-recovery model and the conversation policy model for asking and answering questions, designed in the second case study (Chapter 7), were made possible by this property of the Brahms architecture.

3. Off-task behaviors

Off-task behaviors affect the rhythm work. Someone's activity rhythm is often disturbed by the demands of others in the environment. People often interrupt activities of others, such as asking to participate in an unscheduled meeting, or a telephone call. In Brahms we can easily represent such "off-task" behaviors. Every behavior is represented as the execution of an activity in a workframe. There is no distinction between

workframes, except for the preconditions that determine when they are triggered, and their relative priority. Workframes are declarative statements of when activities are performed. Often, people react to their environment, such as reacting to a ringing telephone, or someone entering their location. People refer to this as “off-task”, because of their goal-oriented perspective of the work. However, with an activity-oriented view of work this can simply be seen as reactive behavior. For example, in the Apollo 12 model the lunar module pilot took photographs of the commander, because of a nice reflection on his visor. This activity was not planned in the overall scheme of offloading the ALSEP. It was simply a reaction to an external event, i.e. the nice reflection. It could be seen as “off-task.” However, in Brahms this photograph activity is represented no differently than the “on-task” activities for the agent offloading the ALSEP. The reason for this is that in Brahms a modeler does not distinguish an agent’s activity as either being on-task or off-task, because Brahms does not represent tasks and goals, but only an agent’s activities.

Off-task behaviors are possible to represent due to the interrupt and resume capability made possible by the subsumption architecture. I thus conclude that Brahms allows the modeler to easily represent off-task behaviors of agents (and/or objects).

4. Multi-tasking

People’s ability to multi-task can be defined in three ways. First, different tasks can be performed by a group of people at the same time. Second, people can multi-task by interrupt and resume (see above). Third, people can perform multiple tasks simultaneously, such as being on the phone while driving a car.

In Brahms it is possible to represent all three ways. First, because Brahms is an agent language, we can represent multi-tasking with multiple agents working on different activities at the same time. For example the simultaneous activities of the CDR and LMP in the Apollo case studies, and more specifically, the VictoriaRover agent moving into the crater while detecting hydrogen with the HydrogenSpectrometer object. Secondly, a single agent can perform “activity context switching.” All the workframes of an agent are vying to become the current workframe the agent is executing. At each moment in time, a workframe’s preconditions can evaluate to true, making it the current workframe and the activity inside it the agent’s current activity. Consequently, an agent might be working on one activity at time t and another activity at time $t+1$, then back to the previous activity at time $t+2$. Such activity context switching emulates people’s multi-tasking behavior. A third form of multi-tasking can be exemplified with the LMP agent CharlieDuke, in the HFE deployment, connecting the HFE to the Central Station while removing the HFE subpallet. In Brahms we model such human multi-tasking abilities with activity decomposition, i.e. by using a composite activity. A higher-level composite activity represented the overall activity of removing the HFE, while the sub-activities within it represented those activities that were performed while removing the subpallet, such as the activity of connecting the HFE to the Central Station. Due to the Brahms subsumption architecture, the agent can be in the activity of removing, *and* at the same time in the sub-activity of connecting. Consequently, the subsumption architecture together with the composite activity language construct, allows representing the simultaneous multi-activity context for an agent.

One limitation of the Brahms architecture is that it is *not* possible for one agent to perform two or more primitive activities at the same time. This limits an agent to perform only one action in the world at a time, and therefore it is *not* possible to represent, for example an agent communicating while moving. The way we would have to represent in Brahms an agent moving *while* communicating is by representing one of the two actions as a composite activity, with the other action as a primitive activity within a workframe in the composite activity. For example, we could represent the movement as a composite activity, and the communication action as a communication activity within a workframe in the composite activity and also a move activity for the actual movement action as a workframe within the composite activity.

I thus conclude that Brahms allows the modeler to represent multi-tasking, by either using multiple agents (or objects), or using one agent and representing multi-tasking with composite activities. Brahms also shows multi-tasking as emergent agent-behavior during the simulation, shown by the agent’s activity context switching made possible by the subsumption architecture. However, representing multiple same-time actions (i.e. primitive activities) in the world is not possible.

5. Informal interaction

The multiagent COT and DAI systems, described in chapter 2, all represent agent interaction based on *formal* agent communication. Formal communication *prescribes* what can be communicated to who and when. Formal interactions are idealizations of how they *should* be performed. However, in practice people often interact informally. Often it is difficult to determine when informal interactions will take place, which makes it hard to model. Most formal models leave out serendipity in the interaction between agents or represent it stochastically. Although Brahms allows for the representation of formal interaction between organizational roles, you can also represent informal communications between agents outside of the context of their formal roles. Brahms agents can come together in geographical locations, detecting each other's presence and begin an informal interaction. Agents can also detect objects in the environment that were created or placed there by other agents. Together with reactive behavior this allows for informal interaction to happen. When such interactions take place is emergent from each agent's independent activity and movement within their environment. Because Brahms agents can detect facts in the world (such as location of objects and agents) and react to them, it allows for serendipity.

The astronaut in the Apollo 12 ALSEP Offload model taking a photograph of his partner, because of a beautiful reflection in his visor, is an example of an informal interaction that I was able to represent in Brahms. However, one obvious aspect of informal interaction is that the agent communication is not *previously* specified as part of a particular activity. How to handle agent communication not "hard-coded" into the model is a challenge, and was addressed in the second case study. Part of the answer is the modeler's representation of non-task specific activities, such as the abstracted conversation policies represented in the HFE Deployment model (chapter 7). Such policies allow agents to react appropriately to informal interaction that are outside the formal procedures, such as sudden questions, or not previously specified occurring external events. Indeed, these types of informal interaction constitute an important part of the work practice.

Although informal interaction is difficult to model, Brahms provides computational possibilities for incorporating informal behavior. Important Brahms language concepts for representing informal agent behavior are detectables, activity priorities, activity inheritance and polymorphism, and parameterized agent communication. Besides these language constructs, the built-in interrupt and resume behavior of the simulation engine is also essential for this. However, it is up to the modeler to develop his or her model in a way that makes agents behave in situations that informally present it self. This is a challenge and an art at the same time. Modeling informal agent behavior is accomplished by generalizing behavior into more general applicable activities of communication and assistance, represented at a higher, non-tasks specific level.

$$\text{informal behavior} = \text{syntactic and simulation provisions in Brahms} + \\ \text{modeler's representation of non-task specific activities}$$

I conclude that a Brahms agent's reactive capability, as well as its ability to detect state changes in the environment (i.e. fact detection) provides modeling and simulating informal interactions.

6. Cognitive behavior

There are various ways Brahms models cognitive behavior. Brahms agents can assert new beliefs or change existing beliefs. This can be done in three ways. First, agents can contain traditional production rules—thoughtframes—that fire in forward-chaining mode. Secondly, Brahms agents can receive new beliefs through a communication with other agents or objects. Thirdly, Brahms agents can detect facts in the world that turn into beliefs for the agent, which allows for the simulation of cognitive stimuli, such as visual, auditory, and touch. Both communicating agents and fact detection in a representation of the physical environment distinguishes Brahms models from more traditional cognitive modeling approaches, such as Soar and ACT-R.

There is another important Brahms language feature that makes Brahms models different from the more traditional cognitive and business process models. Workframes contain consequences, which can create new beliefs for agents, thus creating an agent's internal belief-state of the world, as well as creating new facts, simulating the agent changing the world state. Brahms agents act—perform situated activities within

workframes—based on the individual beliefs the agent accumulates. This activity-based paradigm makes the workframes of a Brahms agent represent the modeler's interpretation of the agent's situated actions in the world. The agent's current, pending and interrupted activities represent the agent's behavior in relation to the rest of the agents and objects in the world. I refer to this as a *model of situated activity*. A workframe-instantiation relates an agent's world-model—its belief-state—to its current activity state, the environment and the other agents and artifacts in it. A Brahms model represents a third-party's model of agent behavior in practice, not, as is the case with cognitive models, a model of the agents' internal cognitive behavior. As is shown in Figure 9-1, a Brahms model is in between a detailed model of the problem-solving behavior (i.e. cognitive behavior), and an abstract model of business process. Instead, it is a model of activity behavior in practice.

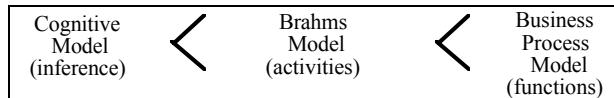


Figure 9-1. Relation of Brahms to other models of work

The subsumption architecture and the composite activity language concept together is one of the most important aspects of the Brahms environment. It provides the modeler with the ability to represent a person's individual actions within context of the person's high-level activities. This makes it possible to model an individual's situated activities in their social, behavioral, cultural, and geographical context. In other words, Brahms allows modeling of people's activities in the way people conceive of themselves being situated in a social environment, group or organization.

7. Geography

The Brahms ability to model geography has been mentioned a number of times in the above descriptions. Representing geographical locations, as well as their state—as facts—allows us to model the relation between the physical world and the agent's activities and cognitive state. People's activities are always located in a geographical space. People's environment and its impact—in the form of constraints—on their ability to act is as important as their ability to reason. This is because people always act within a geographical space of arranged objects that trigger and support their reasoning as an “external” memory (Hutchins 1995). Therefore, representing the relation between the physical location of an agent and activities within this location is one of the most important aspects of modeling people's work practice. This was shown in the case study models, where location and movement of agents and artifacts over time played an important role in their behavior. For example, the drilling of the HFE drill holes in the second case study, and the drilling of the SATM in the third case study.

Brahms allows for a simple representation of the physical world as area definitions and areas and located objects and agents. This limited capability still proved powerful for representing the work practice of the agents and objects in the case studies. However, the representational capabilities in the current Brahms language are limited, and should be extended to allow Brahms to connect to a more detailed virtual representation of the physical world, such as 3D virtual reality models.

One of the limitations of the Brahms geography model is an agent's behavior while moving from one location to another (during a move activity). Agents move over specified paths between locations. Currently, the modeler specifies how long it takes for an agent to move across the path to the connected location on the other end. It would be better to change this to represent not the travel time of the agent, but the length of the path in some metric. Then, according to the speed of the agent (or object) moving over the path, the engine can calculate the travel time required.

Also, a path is not a location, and thus it is not possible for an agent to “run into” another agent that is moving over the same path in a different or same direction at the same time. When an agent's move activity is interrupted, due to a higher priority activity, the agent will “move back” to the start location before the actual move activity is executed. One solution is to implement a path as a type of area, i.e. an area with a certain length and width. This would allow modeling an agent's movement incrementally and instantaneously in a continuous plane, and would allow integration with a 3D virtual-world in which Brahms agents are represented as “bots” (Damer 1997).

A more detailed geography model would allow the specification of 3D worlds with a coordination system, as was originally specified in (Steenvoorden 1995). Such a representation would also allow the representation of an agent's field of vision, for example when facing a particular agent or object, or looking in certain directions. This capability would have allowed the simulation of the field of vision of the television camera object on the lunar rover in the Apollo HFE deployment model (Chapter 24). This would allow for the simulation of what the CapCom agent in mission control could see on the video screen in mission control.

Even though Brahms currently has limited representational capabilities for modeling geography, the current capability does allow for the modeling of the interaction of agents within their environment, as was shown in the case studies. However, the interaction between objects and agents is conceptual, and not based on a model of physics. For example, in the Apollo HFE deployment model the simulated error situation where the commander trips over the "floating" cable connected to the Central Station could not be modeled at the level of physical force between the astronaut's leg movement and the cable. Again, integrating Brahms with a virtual world could allow modeling of the physics.

This concludes the discussion of the first and main research question. In conclusion, I state that I have shown that the Brahms language and simulation environment is able to represent all the aspects of work practice, with some mentioned limitations. I have shown that with the Brahms environment we are able to model and simulate a large number of work process types at the work practice level. However, it is important to mention some examples of types of activities that the current Brahms environment is not able to simulate:

- An agent following another agent; e.g. a patient following a nurse.
- Conversations while walking; e.g. having a conversation in the hallway while walking to a meeting.
- Multiple agents carrying objects together; e.g. carrying a large suitcase together.

These types of activities can be modeled in an abstracted form, but not at the level at which behavior is coordinated between the multiple agents and objects involved. Next, I discuss the second research question.

9.1.2 Research question 2

The secondary research question stated in chapter 1 was:

What is the added value of computer simulation in a model-based approach?

I have shown three case studies where Brahms was used to model and simulate a work practice. In the conclusion sections of the three case study chapters, I showed to what extent we can use the Brahms language and simulation engine to *describe, predict, and prescribe* a work practice. In all three cases the hypothesis was confirmed, and I thus concluded that the Brahms language and simulation engine is by and large sufficient and complete to model and simulate work practice. However, the question "why do we need to simulate?" often comes up. Although simulation has been around since the sixties, in fields such as operations research and industrial engineering (Markowitz et al. 1963) (Tocher 1963) (Mize and Cox 1968) (Kiviat et al. 1969) (Forrest 1970), some engineering analysis and design methodologies have only recently started adding simulation capabilities (Christie 1999). One reason is that only in the late eighties computer systems became powerful enough to support the computational resources needed. Today the benefits of model-based approaches have been widely accepted (Simon 1955) (Newell 1982) (Schreiber et al. 1993) (Yourdon 1989) (Jacobson 1994) (Rumbaugh et al. 1998), while the benefits of computer simulation is less well understood. In other words, what is the value of adding simulation to the modeling process described in chapter 3.3.4 (see Figure 3-6)?

The simple answer is that a static model—non-computational—does not show the behavioral changes of the model elements and their relations over time. Therefore, using a static model we are unable to describe the influence of time on the model. What does this mean for a model of work practice? I will answer this by discussing the effect in terms of negative impact and added value of not being able to simulate the case

studies, and consequently the added value of simulation. The way to read the tables below is that everything from Table 9-1 could also be included in Table 9-2 and Table 9-3, and that everything from Table 9-2 could also be included in Table 9-3. Therefore, the three tables together provide a complete list of impacts and added values.

Case Study 1

Table 9-1 shows the effect of not being able to simulate a descriptive, predictive or prescriptive Brahms model. This analysis is based on the static model and model simulation from case study 1.

Table 9-1. Effects of not being able to simulate in Case Study 1

Element/Outcome	Impact of Not Being Able to Simulate	Added Value of Simulation
Activity Start/End Times	The model is not able to show the start/end times of when agents perform activities.	Shows how the agents spend their time during the course of the simulation. I.e. when activities are executed, started and stopped.
Activity Performance	The model is not able to show which agents perform what group activity.	Shows who performs what group activity when.
Activity Interrupt/Resume	The model is not able to show when activities are interrupted by other activities, and why.	Shows the rhythm of work for an agent. Shows how agents are interrupted, why and by what or who.
Communication	The model is not able to show specific agent communication—agent communicates what, when, and with whom. The model is not able to show what communication tool is used for what activity.	Shows the communication pattern of agents over time. It shows the impact of communication on the performance of agent activities. It shows the communication tools used during specific communication activities; which tool is used when and why (based on where the agent is, and what tool is available).
Movement	The model is not able to show the movement of agents, carrying, moving and creating objects in specific locations, at specific times.	Shows when agents/objects move to or are in specific locations. Shows agent/object movement patterns over time.
Interaction	The model is not able to show which agents work together, and what makes that happen (communication, fact detection, belief creation or location).	Shows when and why agents/objects interact.
Inference	The model is not able to show the change of beliefs over time.	Shows when, where and why agents receive new beliefs.

Case Study 2

Table 9-2 shows the effect of not being able to simulate a predictive or a prescriptive model. This analysis is based on the static model and model simulation outcome from case study 2.

Table 9-2. Effect of not being able to simulate in Case Study 2

Element/Outcome	Impact of Not Being Able to Simulate	Added Value of Simulation
Prediction of Communication	The model is not able to show what will happen if an agent forgets to communicate at a specific time.	Ability to predict agent interaction based on modeled communication policies.
Prediction of Activities	The model is not able to show how an agent recovers from an error situation.	Predict error resolution behavior in specific error situations.
Activity Scheduling	The model is not able to show how an agent's dynamically created schedule is executed.	Ability to schedule an agent's activities dynamically, based on specified plans.
Activity Duration	The model is not able to predict the total duration of the drilling activity, based on the number and length of bore stems used.	Ability to dynamically calculate activity durations based on situation specific data or events.

Case Study 3

Table 9-3 shows the effect of not being able to simulate a prescriptive model. This analysis is based on the static model and model simulation outcome from case study 3.

Table 9-3. Effect of not being able to simulate in Case Study 3

Element/Outcome	Impact of Not Being Able to Simulate	Added Value of Simulation
Calculating Variables	The model is not able to calculate the rover's energy consumption, based on the activities performed.	Ability to dynamically calculate outcome variables based on situational events and activities.

In general, leaving out simulation in a model-based approach means that the model's users will not be able to show any of the effects of timing-relations that exist in the static model. It would be possible to show a static timing model, such as a sequence of activities or flow of data, but without a simulation capability this can never show the emergent interaction and changes over time (similar to the task layer in a static CommonKADS model (van Harmelen and Balder 1992) (Schreiber et al. 2000)).

In a work practice model this means a lack of start and stop times of agent/object activities, agent/object communication, agent/object movement, dynamic agent/object interaction, agent/object activity interruptions and resumes, and inferences over time. To conclude, a static work practice model is not able to predict, nor is it able to calculate. Therefore, a static model does not allow for what-if analysis, which makes a static model less suitable for design and evaluation of new work practices.

9.2 SCALING OF BRAHMS MODELS

It is useful to say something about the issue of scaling Brahms models to larger work systems than those modeled in the case studies. There are two forms of scaling Brahms models I will talk about. First, there is the issue of modeling large work systems. The size of a work practice model can be measured in the number of groups and agents, as well as in the number of work activities represented in the model. The second issue is the issue of scaling the simulated work time in a work system model. The case studies in this thesis model two small and one medium size work system, and all three case studies simulate a short work time interval.

9.2.1 Managing model detail

By managing the detail of the different parts of a work system, we can manage scaling the models up to larger models. This can be done in two ways: teams as agents, and different detail for different parts of the model.

9.2.1.1 Teams as agents

By modeling groups of people as single agents we can easily scale a model to large organizations. The design of the agent model is completely flexible, and the modeler can decide what teams are modeled as agents, and which individuals as agents. This approach was used in the third case study, where the organization that was modeled was an order of magnitude larger than that of the two Apollo case studies. This was accomplished by modeling the science sub-teams as agents, while the Victoria rover was modeled as an agent. Doing this made the total number of agents not much larger than those in the Apollo models.

9.2.1.2 Different detail for different parts of the model

Model detail can be managed easily for different parts of the model. It is the modeler's choice to decide which agent and objects are modeled in what detail. This approach was successfully used in the Apollo models and the Victoria model. In the Apollo model, the detail of the activity of the CapCom agent was very small. In the Victoria model, the detail of the rover and the instruments models are much higher than those of the science teams. This was partly due to the fact that no more information was available, but it was also deliberately based on the chosen objective of the model. The approach is general, and can be used for modeling complex work systems.

9.2.1.3 Adding more detail

One of the big issues is using a model over a long project life cycle. The cost benefit of developing a complex Brahms model will go up if we can use the model in the different life-cycle phases of a project. There is a great potential that a Brahms modeling effort could be used in the complete life cycle of a development project. Models from one phase of the project can be easily reused in the next phase. Due to the Brahms agent language, a Brahms model can be easily made more specific by adding more detailed agents and activities. A model of one agent per group can be easily scaled-up to multiple agents of the same group, by simply adding more members of the group. Currently, future research interests are in showing the use of Brahms from analysis to design and implementation for developing agent-based software systems. Future research will focus on generation of agent software systems from a Brahms model.

9.2.2 Modeling longer time intervals

Scaling Brahms models to model large organizations is not the most difficult scaling issue. This seems to be easily handled by the Brahms language. However, the experience to date is only with models that simulate a relatively small time period of the total work activities in the modeled work system. The case study models simulate no more than a couple of hours of work. However, models will certainly have to simulate a complete work day, and longer. For example, if we would model the work-life of astronauts in the International Space Station we would certainly want to model several days, if not weeks of work. A model of

a human mission to Mars would easily need to simulate the trip to Mars, as well as the time on Mars and the trip back to Earth. Such models would need to simulate months, if not years of work activity.

At this moment, I have no relevant data of models that simulate one day, let alone weeks, months or even years. There are several research issues that show up when we think of scaling the simulated work time to such levels. For example, how do we handle training, learning, forgetting of agents that are simulated for weeks or months?

Given the limited simulated work time in the case studies, I cannot give a substantiated argument about scaling Brahms models to simulate more than a day of work in an organization. More research is needed to answer the question of scaling simulations to simulating longer time periods.

9.2.3 Gathering data for large organizations

The biggest scaling issue for Brahms models is in the ability to gather organizational work practice data on which to base a model (methodology M1, from chapter 3.3.4). Creating Brahms models is relatively easy. One simply needs to learn the Brahms programming language and the Brahms modeling approach. However, work practice data gathering is a serious bottleneck. Observational work inside existing work systems is a) a lot of work, and b) not always possible. The ability or inability to gather the relevant data for developing a Brahms model is one of the most problematic issues in scaling up the methodology.

9.3 SCIENTIFIC CONTRIBUTIONS

In this thesis, I have presented a new methodology for modeling human and system work processes. This new methodology is based on combining a number of new and existing representational paradigms—agent-based, object-based, geography-based, rule-based, belief-based, and activity-based. This has led to an extension of knowledge in several areas of management and computer science. The methodology for modeling and simulating work practice presented in this thesis extends the knowledge of modeling and simulation of work processes, design and development of agent-based systems, and the recently emerging field of human-centered systems (Kling et al. 1997). In this section, I briefly discuss the contributions that were made to these scientific fields.

9.3.1 Modeling and simulation of work processes

Current state of the art in workflow or business process modeling is based on an information process-oriented paradigm. Business processes are abstractions of how work gets done in organizations. Collaboration between people and organizations is represented as the processing of information; who gets what when? People's work activities are seen as long- and short-term transactions in which information is either being sent or being requested. People's work practice knowledge is abstracted away in so called business rules, which describe the business process logic for when and how to process information.

The work in this thesis shows a modeling and simulation methodology using a new agent-based and activity-based paradigm. Modeling work processes using this paradigm is shown to be more naturally located in an organization's work practice, i.e. how the work really gets done. The case studies in this thesis are about a work process domain that few of us would suggest to be relevant to business process research. However, after an initial reluctance to see the relevance and when looking at the work practice issues at hand, we see many similarities with work processes in Earth-based organizations. The Apollo Astronauts on the Moon were working, not just “looking around.” They were performing planned activities, having goals and objectives. The work during an Apollo mission, or a robotic mission for that matter, is distributed across many teams, co-located and distributed in space and time. Information is being communicated in real-time, as well as in “batch mode.” Activities span multiple days and multiple organizational handovers, while tools and systems are used to perform the work.

Using the Brahms language and simulation environment, I was able to model the work process at a level that is not possible in today's work process tools. The Brahms research was started because of a failed attempt to model a coordination role using a workflow paradigm. In contrast, coordination roles are easily represented with the agent-based paradigm in Brahms, as is shown with the modeling of CapCom during

the Apollo missions. A Brahms model focuses the modeler's attention on the activities of individuals or groups of individuals. Work is described in terms of an agent's behavior over time, and its actual impact on the organizational goal. A model of work practice is a holistic approach to process modeling. It allows for multiple views of the work process: organization, individual activity, communication and geographical. This is in contrast to only a process view, where the focus is on transformation of process input to output and not on how this happens in practice. Using the modeling methodology described in this thesis will allow the business process modeling community to model processes in organizations down to the work practice level. The approach makes it possible to represent business rules based on how things really work, not based on how things should work according to idealized prescriptive formulations. Representing a business process as a team of cross-organizational agents, collaborating together in performing their activities is more natural. The benefit of this approach will especially show its value in the implementation of new business processes and systems, where changes in the current work practice of the organization are inevitable.

9.3.2 Agent-based systems

Current state-of-the-art software agent research focuses on two types of agent-based systems (Bradshaw 1997); 1) software agents that behave independently and autonomously, regardless of their interaction with humans, and 2) software systems that interact with people in such a way that people ascribe agency to the software system, as for them the system acts as-if it is independent and/or autonomous. However, there is a movement to the middle where agent researchers are trying to combine these two notions of software agents into one definition that define agent-based systems (Bradshaw 1996) (Shoham 1993) (Genesereth and Nilsson 1994) (Weiss 1999) (Wooldridge and Jennings 1995). A non-exhaustive list of research topics that are included in this movement is: teamwork, mobility, intelligence, reactivity, learning, and personality.

The contribution that this thesis makes to this research area, although not the main topic of the work, is to push the use of an agent-based system—Brahms—to represent the behavior of people, not merely software agents. Representing people's behavior, by definition, includes all the agent research topics listed above. This is not a new revelation, for the simple reason that the whole notion of agency is derived from humans. However, what is shown by the work in this thesis is how we can represent collaboration between people. By extension, we can use this to understand how we could represent teamwork for software agents, or a combination of human and software agents. What was also shown is how to represent people acting based upon and interacting with their environment, including movement and reactive behavior. By extension, we can use this to represent how software agents act upon and interact with the hardware and software environment they "live" in. Last, but not least, the Brahms agent architecture was intentionally based on multi-tasking and the interrupt and resume ability of humans. In this thesis, I have shown that the Brahms subsumption architecture is highly appropriate for representing human behavior as deliberate or reactive activity-based behavior. By extension, it seems useful to use the same subsumption architecture in the development of intelligent or flexible software agents that must interact with humans.

9.3.3 Human-centered computing

There is a strong move towards a multi-disciplinary approach for the development of computer systems in particular, and technology systems in general. Computer scientists, systems designers, and social scientists are coming together to frame a computing paradigm shift. This is a true shift from the way computer systems have conventionally been designed and implemented. Usually, systems are being developed according to a technology-centered design approach. This approach adheres to the rule, the more and the "better" the technology, the better and useful the tools. This is the mindset of most computer scientists and system developers today. The technology is put in the center, while the users and how they work are on the periphery. Even though we know about participatory design (Ehn 1988) (Greenbaum and Kyng 1991), most computer systems today are still developed in isolated technology organizations and the ivory laboratory towers of companies. Obtaining a better understanding of how the users perform their work, based on workplace observations and ethnographical studies, and modeling and simulation of the work practice, is rarely accomplished during the requirements analysis, design and implementation of systems.

However, software development is changing. More and more computer and information science departments in the United States are starting to offer graduate courses in human-centered systems or human-centered computing (e.g. UC Berkeley, Indiana University, UC Irvine). The National Aeronautics and

Space Administration (NASA) has a significant funding budget for Human-Centered Computing (NASA 2000). The National Science Foundation (NSF), in 1997, held a workshop on Human-Centered Systems (Flanagan and Huang 1997).

Human-centered computing is proposed as a new paradigm for system development. The analysis has to be based on a deep understanding of the concrete social and technical environment, rather than some vague idea of a generic user. In this thesis, I have shown a methodology that allows system designers to do this, using a model-based simulation approach.

Kling, et al (1997) specify a number of research directions. I would argue that the research in this thesis has contributed to at least two of these research directions: a) characterization and theories of human-centered systems, and b) modeling and representing human-centered systems. As a contribution to the first, I have defined a theory of modeling work practice that allows us to characterize the realistic work process of a human organization and how systems fit in and are being used within this process. The second mentions modeling human-centered systems. I have shown such a capability by developing a methodology for analyzing and designing human work systems.

9.4 FUTURE RESEARCH

In this last section, I take the liberty to present some of my ideas on future research that is needed in the area of work practice modeling and simulation. This section may serve as a guide for future research in this area

9.4.1 Work system design methodology

Although I presented a methodology for modeling and simulating work practice, much needs to be done to provide the business and software engineering community a methodology for designing human-centered work systems in existing and new organizations. Designing a total work system is a multi-disciplinary effort. It combines business anthropology, business management, knowledge-based systems engineering, and information technology development. A work system design methodology should adhere to a holistic human-centered approach in which the design is based on the people's existing work practice, the constraints of the work environment, the ability to integrate new technology with existing technology, and the adherence to the philosophy that technology has to be human-centered.

The Brahms methodology is only one piece of the puzzle. It provides a worldview, an approach and a tool that can be applied in this process. However, a complete work system design methodology needs to provide a path from analysis and design to implementation. Modeling and simulation can be used within each phase of this path, but future research needs to work out how this should be done effectively. I state that current business process redesign methodologies lack theories, methods and tools to be applied in work system design efforts. Instead, today's business process redesign methodologies only focus on information systems design for an organization's electronic-business and globalization efforts, without a notion of what it means to develop human-centered systems.

9.4.2 Enhancing the Brahms language

There are several areas in which research needs to be conducted to enhance the Brahms language with more sophisticated capabilities to represent human behavior.

Learning

First and most obvious is the notion of learning. Currently, Brahms agents do not learn. What constitutes learning is a difficult and widely researched topic. However, what seems particularly interesting in the learning of human activity is how humans learn to do something over time by a form of apprenticeship, and by collaborating with others. This goes beyond the research in machine learning and data mining, although it might be useful to investigate the use of machine-learning techniques. What seems to be needed is a theory about learning through watching others, as well as a theory of learning from practice and the circumstantial aspects of performing activities. How do people learn to perform new activities by participating

in ongoing activities with others? Social scientists sometimes refer to this as *legitimate peripheral participation* (Lave and Wenger 1991). How do people learn by observing and mimicking? How does their performance change over time? With the notion of learning comes the notion of forgetting. Activity performance degradation is an important aspect of forgetting. What is needed is a theory of human learning of activities that also explains activity degradation and forgetting over time.

Researchers are getting interesting results in the area of behavior-based robotics. However, most learning in such behavior-based systems is at a very low-level, such as learning to map the world, and navigate in that world. More recently, the field of cognitive robotics or *cognobotics* is starting to deal with learning of new cognitive behavior in robots (Brooks 1997). What is needed is learning of complex high-level activities and when and where to perform them. This is still an area where more research needs to be done.

Biological and social primitives

Another important aspect of human activity is the biological constraints on the human body. We need to understand which biological constraints have an impact on activity performance, and how to model such an impact. Examples include fatigue, hearing and field of vision (Freed 1998). Research needs to be conducted in order to develop a theory of the impact of these constraints on human activity, and a way to represent these constraints within the Brahms environment. Either as part of the language or as part of the simulation engine.

Personality, emotion and social constraints are another important part of human behavior. For example, trustworthiness and liking or disliking people impacts how we collaborate. How do we represent the social and personality behavior of, for example, a teenage gang, or a team of astronauts going on a three-year mission to Mars, living in a confined space together?

More research in this area would help us develop representational schemes for including these types of phenomena in a Brahms model.

A. BRAHMS LANGUAGE

A.1. NOTATION CONVENTIONS (Backus Naur Form)

This appendix defines the modeling language for Brahms. This document is to be used by model builders to create Brahms models. Brahms model builders will have to comply to the language as defined in this document. This document is also used as a requirements specification for the parser that needs to be build to parse these models.

The language element are defined in BNF (Backus Normal Form) grammar rules. The notation used in these grammar rules is given in table 1.

Construct	Interpretation
<code>::= * + {} [] .</code>	Symbols part of the BNF formalism
<code>X ::= Y</code>	The syntax of X is defined by Y
<code>{X}</code>	Zero or one occurrence of X
<code>X*</code>	Zero or more occurrences of X
<code>X+</code>	One or more occurrences of X
<code>X Y</code>	One of X or Y (exclusive or)
<code>[X]</code>	Grouping construct for specifying scope of operators e.g. [X Y] or [X]*
<code>symbol</code>	Predefined terminal symbol of the language
<code>symbol</code>	User-defined terminal symbol of the language
<code>symbol</code>	Non-terminal symbol

Table 9-4: Synopsis of the notation used

Identifiers (ID)

name ::= [letter][letter | digit | '-']*

letter ::= 'a' | 'b' | ... | 'z' | 'A' | 'B' | ... | 'Z' | '_'

digit ::= '0' | '1' | ... | '9'

blank-character ::= ' ' | '\t' | '\n' | '\f' | '\r'

number ::= [integer | double]

integer ::= {+ | -} unsigned

unsigned ::= [digit]+

double ::= [integer, unsigned]

truth-value ::= true | false

literal-string ::= " [letter | digit | '-' | ':' | ';' | '?'] "

literal-symbol ::= name

It is possible to add comments to models. One line comments need to start with '/'. Multi-line comments have to start with '/*' and end with '*/'.

A.2. SYNTAX OF A MODEL (MOD)

Syntax

```
model ::= [import] import-declaration ; ]*
[ GRP.group |
AGT.agent |
CLS.object-class |
OBJ.object |
COC.conceptual-object-class |
COB.conceptual-object |
ADF.area-def |
ARE.area |
PAT.path ]*
```

import-declaration ::= single-type-import | multi-type-import

single-type-import ::= concept-name | library-name , concept-name

concept-name ::= ID.name

library-name ::= ID.name | library-name , ID.name

multi-type-import ::= * | library-name . *

Semantics

The import declaration allows for the import of specific concepts or for the import of a library of concepts. The import of a specific concept is realized by referencing it's name. The name of the concept must be the same as the name of the file in which it is stored. The extension of the file must always be '.b'.

To reference a specific concept in a library the library-name can be used. The library name reflects the directory in which the concept is stored with a 'library-path' as it's base path. So for example if the library-path is

library-path = C:\brahms

and we have an import statement like

```
import nynexst.phonemodel.PhoneUsers;
```

then the concept PhoneUsers is expected to be found in the file

```
C:\brahms\nymexst\phonemodel\PhoneUsers.b
```

It is also possible to reference all concepts in a specific library. The wildcard '*' can be used in place of a specific concept-name. The following import statement will import all concepts in the phonemodel library:

```
import nynexst.phonemodel.*;
```

This statement will import all concepts defined in the directory C:\brahms\nymexst\phonemodel defined in the files with the extension '.b' assuming the library-path is set to 'C:\brahms'.

The import statement:

```
import *;
```

will import all concepts defined in the files with extension ‘.b’ that are in the same directory as the file in which the import statement is defined.

By default every model imports the ‘brahms.base.*’ library (referred to as the ‘BaseModel’) containing base constructs for groups and classes and containing standard available classes and relations. The import of this library does not have to be defined explicitly.

A.3. SYNTAX OF AN AGENT (AGT)

Syntax

agent ::=

```
agent agent-name { GRP.group-membership }{  
  { display : ID.literal-string ; }  
  { cost : ID.number ; }  
  { time_unit : ID.number ; }  
  { location : ARE.area-name ; }  
  { GRP.attributes }  
  { GRP.relations }  
  { GRP.initial-beliefs }  
  { GRP.initial-facts }  
  { GRP.activities }  
  { GRP.workframes }  
  { GRP.thoughtframes }  
}
```

agent-name ::= ID.name

Semantics

Group membership

An agent can be a member of one or more groups. When an agent is a member of a group the agent will 'inherit' attributes, relations, initial-beliefs, initial-facts, activities, workframes and thoughtframes from the group(s) it is a member of. All attributes and relations are inherited including private ones (an agent can be seen as an instance of a group in terms of object oriented practices). In case the same constructs are encountered in the inheritance path always the most specific construct will be used, meaning that a workframe defined for the agent has precedence over a workframe with the same name defined in one of the groups of which the agent is a member.

Defaults

Every agent in a model is by definition a member of 'BaseGroup' defined in the 'BaseModel' library which is imported by definition for every model. The 'BaseGroup' defines built-in attributes, relations, initial-beliefs, initial-facts, activities, workframes and thoughtframes as defaults for agents and groups. The 'BaseGroup' membership does not have to be defined explicitly. Other defaults are:

```
display = <agent-name>  
cost = 0  
time_unit = 0  
location = none
```

Constraints

1. The name of an agent must be unique amongst agents in a model.
2. The time_unit defines the time in seconds.

A.4. SYNTAX OF A GROUP (GRP)

Syntax

group ::=

```
group group-name { group-membership }{  
    { display : ID.literal-string ; }  
    { cost : ID.number ; }  
    { time unit : ID.number ; }  
    { attributes }  
    { relations }  
    { initial-beliefs }  
    { initial-facts }  
    { activities }  
    { workframes }  
    { thoughtframes }  
}
```

group-name ::= ID.name

group-membership ::= memberof group-name [, group-name]*

attributes ::= attributes : [ATT.attribute]*

relations ::= relations : [REL.relation]*

initial-beliefs ::= initial beliefs : [BEL.initial-belief]*

initial-facts ::= initial facts : [FCT.initial-fact]*

activities ::= activities : [activity]*

activity ::= [CAC.composite-activity |
 PAC.primitive-activity |
 MOV.move-activity |
 COA.create-object-activity |
 COM.communicate-activity |
 BCT.broadcast-activity]

workframes ::= workframes : [WFR.workframe]*

thoughtframes ::= thoughtframes : [TFR.thoughtframe]*

Semantics

Group membership

In a model a hierarchy of groups can be built by defining the group-membership. A group can be a member of more than one group. When a group is a member of a group the member-group will 'inherit' the attributes, relations, initial-beliefs, initial-facts, activities, workframes and thoughtframes from its parent groups. Private attributes and relations are not inherited, only public and protected attributes and relations are inherited. In case the same constructs are encountered in the inheritance path always the most specific construct will be used, meaning that a workframe defined for a group lowest in the hierarchy tree has precedence over a workframe with the same name higher in the hierarchy.

Cost and Time-Unit

The cost and time-unit are used for statistical purposes and define the cost/time-unit (in seconds) for work done by members of the group. The members of the group can override the cost and time-unit figures.

Defaults

Every group in a model is by definition a member of 'BaseGroup' defined in the 'BaseModel' library which is imported by definition for every model. The 'BaseGroup' defines built-in attributes, relations, initial-beliefs, initial-facts, workframes and thoughtframes as defaults for groups. The 'BaseGroup' membership does not have to be defined explicitly. Other defaults are:

```
display = <group-name>
cost   =0
time_unit =0
```

Constraints

1. The name of a group must be unique amongst groups in a model.
2. The time_unit defines the time in seconds.

A.5. SYNTAX OF AN OBJECT CLASS (CLS)

Syntax

object-class ::=

```
  class object-class-name { class-inheritance }  
    { display : ID.literal-string ; }  
    { cost : ID.number ; }  
    { time unit : ID.number ; }  
    { resource : ID.truth-value ; }  
    { GRP.attributes }  
    { GRP.relations }  
    { GRP.initial-beliefs }  
    { GRP.initial-facts }  
    { GRP.activities }  
    { GRP.workframes }  
    { GRP.thoughtframes }  
}
```

object-class-name ::= ID.name

class-inheritance ::= extends object-class-name [, object-class-name]*

Semantics

Class inheritance

In a model a hierarchy of classes can be built by defining the class inheritance. A class can inherit from more than one class, so multiple inheritance is supported. When a class is a subclass of a class the subclass will ‘inherit’ the attributes, relations, initial-beliefs, initial-facts, activities, workframes and thoughtframes from its parent classes. Private attributes and relations are not inherited, only public and protected attributes and relations are inherited. In case the same constructs are encountered in the inheritance path always the most specific construct will be used, meaning that for example a workframe defined for a class lowest in the hierarchy tree has precedence over a workframe with the same name higher in the hierarchy.

Cost and Time-Unit

The cost and time-unit are used for statistical purposes and define the cost/time-unit (in seconds) for work done by instances of the class. The instances of the class can override the cost and time-unit figures.

Resource

The resource attribute defines whether or not instances of the class are considered to be a resource when used in an activity (resource attribute is set to true) or whether the instances of the class are considered something that is worked on (resource attribute is set to false). The resource attribute is used in relation with the touched-objects definition for activities (see the semantical description of touched-objects in the definition of the primitive-activity).

Defaults

Every class in a model is by definition a member of ‘BaseClass’ defined in the ‘BaseModel’ library which is imported by definition for every model. The ‘BaseClass’ defines built-in attributes, relations, initial-beliefs, initial-facts, workframes and thoughtframes as defaults for classes. The ‘BaseClass’ membership does not have to be defined explicitly. Other defaults are:

```
display = <class-name>  
cost = 0  
time_unit = 0  
resource = false
```

Constraints

1. The name of a class must be unique amongst classes in a model.
2. The time_unit defines the time in seconds.

Current Limitations

The current simulation engine does not support class inheritance. The parser will take care of attribute, relation, workframe and thoughtframe inheritance by just ‘copying’ those constructs to every class in the inheritance tree. Not possible however is assuming that the simulation engine will know of the existence of the inheritance hierarchy. A model builder can for example not define a variable of type Location and assume that objects being instances of the subclass Building will be bound to the variable. This functionality is planned for a future version of the simulation engine.

The current simulation engine also does not support the use of thoughtframes in object classes. This again will be supported in a future release of the simulation engine.

A.6. SYNTAX OF AN OBJECT (OBJ)

Syntax

object ::=

```
object object-name instanceof object-class-name
  { COB.conceptual-object-membership }{
    { display : ID.literal-string ; }
    { cost : ID.number ; }
    { time_unit : ID.number ; }
    { resource : ID.truth-value ; }
    { location : ARE.area-name ; }
    { GRP.attributes }
    { GRP.relations }
    { GRP.initial-beliefs }
    { GRP.initial-facts }
    { GRP.activities }
    { GRP.workframes }
    { GRP.thoughtframes }
  }
```

object-name ::= ID.name

Semantics

Conceptual object membership

An object can be part of one or more conceptual objects by defining the conceptual-object-membership for the object. This allows for later grouping of statistical results for the object with other objects in one conceptual object.

Resource

The resource attribute defines whether or not the object is considered to be a resource when used in an activity (resource attribute is set to true) or whether the object is considered something that is worked on (resource attribute is set to false). The resource attribute is used in relation with the resources definition for activities (see the semantical description of resources in the definition of the primitive-activity).

Defaults

```
display = <object-name>
cost = 0
time_unit = 0
resource = <the resource attribute value of object-class-name>
location = none
```

Constraints

1. The name of an object must be unique amongst objects in a model. It is possible to give an object the same name as an object defined in a library.
2. The time-unit defines the time in seconds.

Current Limitations

The current simulation engine does not support thoughtframes for objects. A future release of the simulation engine will provide support for thoughtframes on objects.

A.7. SYNTAX OF A CONCEPTUAL OBJECT CLASS (COC)

Syntax

```
conceptual-object-class ::=  
  conceptual_class conceptual-class-name {  
    display : ID.literal-string ;}  
    { GRP.attributes }  
    { GRP.relations }  
  }
```

conceptual-class-name ::= ID.name

Semantics

Defaults

display = <conceptual-class-name>

Constraints

1. The name of a conceptual-object-class must be unique amongst the conceptual-object-classes in a model.

A.8. SYNTAX OF A CONCEPTUAL OBJECT (COB)

Syntax

```
conceptual-object ::=  
  conceptual_object conceptual-object-name instanceof conceptual-class-name  
    { conceptual-object-membership }  
  {  
    { display : ID.literal-string ; }  
    { GRP.attributes }  
    { GRP.relations }  
  }  
  
conceptual-object-name ::= ID.name  
  
conceptual-object-membership ::=  
  partof conceptual-object-name [ , conceptual-object-name ]*
```

Semantics

Conceptual-object-membership

A conceptual-object can in itself be a member of other conceptual object forming a hierarchy of concepts for grouping statistical results.

Defaults

display = <conceptual-object-name>

Constraints

1. The name of a conceptual-object must be unique amongst the conceptual-objects in a model.

A.9. SYNTAX OF AN ATTRIBUTE (ATT)

Syntax

```
attribute ::= { private | protected | public } type-def attribute-name { attrib-body } ;  
  
attribute-name ::= ID.name  
  
type-def ::= [ class-type-def | value-type-def ]  
  
class-type-def ::=  
    [ Agent |  
    Group |  
    Class |  
    ConceptualClass |  
    AreaDef |  
    GRP.group-name |  
    CLS.object-class-name |  
    COC.conceptual-class-name  
    ADF.area-def-name ]  
  
value-type-def ::= [ int | double | symbol | string | boolean ]  
  
attrib-body ::= {  
    { display : ID.literal-string ; }  
}
```

Semantics

Attribute scope

Attributes are always defined within a group, agent, conceptual-class, conceptual-object, class or object definition and cannot be defined outside any of these concepts or inside of any other concepts. Attributes can have different scopes within the specified concepts defined by one of the keywords private, protected or public.

Private attributes:

Private attributes are scoped down to only the concept for which it is defined. The private attribute is not inherited by sub groups or sub classes (agents /objects that are members/instances of the group/class will inherit the attribute) and the private attribute can only be referenced by initial beliefs, initial facts, workframes and thoughtframes for that specific concept.

Protected attributes:

Protected attributes are inherited by sub groups and sub classes. Protected attributes can only be referenced by initial beliefs, initial facts, workframes and thoughtframes of the concept for which the attribute is specified or any of the sub groups/sub classes and of agents/objects that are members/instances of the sub group(s)/class(es).

Public attributes:

Public attributes are similar to protected attributes, the only difference is that they can be referenced by initial beliefs, initial facts, workframes and thoughtframes in any group, agent, class or object.

Value assignment

Value assignment of attributes differs from value assignments in third and fourth generation computer languages (which usually use an assignment operator like '=' or ':='). Assignment of a value for an attribute is done through beliefs and facts.

Defaults

The default scope of an attribute is ‘public’.

display = <attribute-name>

Constraints

1. The name of an attribute must be unique within the definition of a group, agent, class or object. In case of a name conflict in multiple inheritance (two different concepts from which is inherited define an attribute with the same name) the following conflict resolution strategy is chosen. If both attributes are of the same type just one definition will remain with the same name and same type. If the types of the attributes differ an error will be generated.

Current Limitations

The current simulation engine cannot handle group-name types, which would bind an attribute to only agents that are members of the defined group similar to classes. A future release of the simulation engine will allow for group-names to be used as a type definition.

A.10. SYNTAX OF A RELATION (REL)

Syntax

```
relation ::= { private | protected | public } ATT.class-type-def relation-name  
{ attrib-body } ;
```

relation-name ::= ID.name

Semantics

Relation scope

Relations are always defined within a group, agent, conceptual-class, conceptual-object, class or object definition and cannot be defined outside any of these concepts or inside of any other concepts. Relations can have different scopes within the specified concepts defined by one of the keywords private, protected or public.

Private relations:

Private relations are scoped down to only the concept for which it is defined. The private relation is not inherited by sub groups or sub classes (agents /objects that are members/instances of the group/class will inherit the relation) and the private relation can only be referenced by initial beliefs, initial facts, workframes and thoughtframes for that specific concept.

Protected relations:

Protected relations are inherited by sub groups and sub classes. Protected relations can only be referenced by initial beliefs, initial facts, workframes and thoughtframes of the concept for which the relation is specified or any of the sub groups / sub classes and of agents/objects that are members/instances of the sub group(s)/class(es).

Public relations:

Public relations are similar to protected relations, the only difference is that they can be referenced by initial beliefs, initial facts, workframes and thoughtframes in any group, agent, class or object.

Defaults

The default scope of a relation is 'public'.

display = <relation-name>

Constraints

1. The name of a relation must be unique within the definition of a group, agent, class or object. In case of a name conflict in multiple inheritance (two different concepts from which is inherited define a relation with the same name) the following conflict resolution strategy is chosen. If both relations are of the same type just one definition will remain with the same name and same type. If the types of the relations differ an error will be generated.

A.11. SYNTAX OF AN INITIAL-BELIEF (BEL)

Syntax

initial-belief ::= {[value-expression | relational-expression]} ;

value-expression ::= obj-attr evaluation-operator value |
obj-attr equality-operator object-ref

equality-operator ::= = | !=

evaluation-operator ::= BEL.equality-operator | ≥ | >= | ≤ | <=

obj-attr ::= the ATT.attribute-name of object-ref | object-ref , ATT.attribute-name

object-ref ::=

AGT.agent-name |
OBJ.object-name |
ARE.area-name |
VAR.variable-name |
PAC.param-name |
current

value ::= ID.literal-string | ID.number | PAC.param-name

relational-expression ::=

object-ref REL.relation-name object-ref { is ID.truth-value }

Semantics

Constraints

1. Variables and parameters are not allowed in the definition of an initial belief.
2. The attribute type and the right hand side value-type of a value-expression must be the same.
3. The left hand side and right hand side types in a relational expression must match the types as defined for the relation used in the relational expression.

A.12. SYNTAX OF AN INITIAL-FACT (FCT)

Syntax

initial-fact::=

{ [BEL.value-expression | BEL.relational-expression] } ;

Semantics

Constraints

1. Variables and parameters are not allowed in the definition of an initial fact.
2. The attribute type and the right hand side value-type of a value-expression must be the same.
3. The left hand side and right hand side types in a relational expression must match the types as defined for the relation used in the relational expression.

A.13. SYNTAX OF A WORKFRAME (WFR)

Syntax

workframe ::=

```
workframe workframe-name {  
    { display : ID.literal-string ; }  
    { repeat : ID.truth-value ; }  
    { priority : ID.unsigned ; }  
    { variable-decl }  
    { detectable-decl }  
    { [ precondition-decl workframe-body-decl ] |  
        workframe-body-decl }  
}
```

workframe-name ::= ID.name

variable-decl ::= variables : [VAR.variable]*

detectable-decl ::= detectables : [DET.detectable]*

precondition-decl ::= when ({ [PRE.precondition] [and PRE.precondition]* })

workframe-body-decl ::= do { [workframe-body-element]* }

workframe-body-element ::= [PAC.activity-ref | CON.consequence]

Semantics

Repeat

A workframe can be performed one or more times depending on the value of the 'repeat' attribute. A workframe can only be performed once if the repeat attribute is set to false. A workframe can be performed repeatedly if the repeat attribute is set to true. In case the repeat attribute is set to false, the workframe can only be performed once for the specific binding of the variables at run-time. The scope of the repeat attribute of a workframe defined as part of a composite activity is limited to the time the activity is active, meaning that the workframe with a specific binding and a repeat set to false will not execute repeatedly while the composite activity is active. As soon as the composite activity is ended the states are reset and in the next execution of the activity it is possible for the workframe with the same binding to be executed. So only for top-level workframes the state will be stored permanently during a simulation run.

Priority

The priority attribute is currently not used. At this moment priorities have to be defined on the level of activities. The workframe priority will be deduced based on the priorities of the activities defined within the workframe. The workframe will get the priority of the activity with the highest priority.

Defaults

```
display = <workframe-name>  
repeat = false  
priority=0
```

Constraints

1. The workframe name has to be unique within the definition of a group, agent, object-class or object.

A.14. SYNTAX OF A THOUGHTFRAME (TFR)

Syntax

thoughtframe ::=

```
thoughtframe thoughtframe-name {  
    { display : ID.literal-string ; }  
    { repeat : ID.truth-value ; }  
    { WFR.variable-decl }  
    { [ WFR.precondition-decl thoughtframe-body-decl ] }  
}
```

thoughtframe-name ::= ID.name

thoughtframe-body-decl ::= **do** {[thoughtframe-body-element ;]* }

thoughtframe-body-element ::= CON.consequence

Semantics

Repeat

A thoughtframe can be performed one or more times depending on the value of the 'repeat' attribute. A thoughtframe can only be performed once if the repeat attribute is set to false. A thoughtframe can be performed repeatedly if the repeat attribute is set to true. In case the repeat attribute is set to false, the thoughtframe can only be performed once for the specific binding of the variables at run-time. The scope of the repeat attribute of a thoughtframe defined as part of a composite activity is limited to the time the activity is active, meaning that the thoughtframe with a specific binding and a repeat set to false will not execute repeatedly while the composite activity is active. As soon as the composite activity is ended the states are reset and in the next execution of the activity it is possible for the thoughtframe with the same binding to be executed. So only for top-level thoughtframes the state will be stored permanently during a simulation run.

Defaults

```
display = <thoughtframe-name>  
repeat = false
```

Constraints

1. The thoughtframe name has to be unique within the definition of a group, agent, object-class or object.
2. It is not possible to use unassigned variables in thoughtframes, therefor the definition of these variables is not allowed.
3. The consequences in thoughtframes can only conclude beliefs.

A.15. SYNTAX OF A COMPOSITE ACTIVITY (CAC)

Syntax

```
composite-activity ::=  
  composite_activity PAC.activity-name{ { PAC.param-decl [, PAC.param-decl]* } }  
{  
  { display : ID.literal-string ; }  
  { priority : [ ID.unsigned | PAC.param-name ] ; }  
  { end condition : [ detectable | nowork ] ; }  
  { WFR.detectable-decl }  
  { GRP.activities }  
  { GRP.workframes }  
  { GRP.thoughtframes }  
}
```

Semantics

Declaration and reference

All activities have to be declared in the activities section of either a group, agent, class, object, or composite-activity. The declared activities can then be referenced in the workframes defined for the group, agent, class or object.

Parameters

It is possible to define input parameters for composite activities. These input parameters are in the first place used to pass on variables defined in a workframe for use in the workframes defined for the composite activity and second they can be used to make activities more generic. In the reference the values for the input parameters have to be passed.

Priority

Activities can be assigned a priority. The priorities of activities in a workframe are used to define the priority of a workframe. The workframe will get the priority of the activity with the highest priority defined in the workframe.

Duration

Composite activities themselves do not have a duration. Composite activities are decomposed into other workframes that a concept can work on as part of the activity which eventually result in a primitive activity to be executed having a specific duration.

End-condition

The end-condition of a composite activity defines how a composite activity can be ended. There are three possibilities:

1. Only end it on the basis of an end-activity detectable. The end-condition has to be set to 'detectable'. When a detectable having an action type of 'end-activity' is detected the composite activity will be ended.
2. End the activity when there's no more work to work on. The end-condition has to be set to 'no-work'. If none of the workframes as defined in the composite-activities can be worked on the activity will be ended.
3. End the activity when there's no more work to work on, or when an end-activity detectable is detected. The end-condition has to be set to 'no-work' and an end-activity detectable needs to be defined for the composite activity. This case combines the first two cases.

Defaults

```
display =<activity-name>  
priority=0
```

```
end_condition = nowork
```

Constraints

1. The name of an activity must be unique within the definition of a group, agent, class, object, or composite-activity.
2. The input parameter type of a parameter defined in the declaration of an activity must be the same as the input value type or variable type in the reference of the activity.
3. The parameters assigned to any of the attributes must be of the correct type.

A.16. SYNTAX OF A PRIMITIVE ACTIVITY (PAC)

Syntax

```
primitive-activity ::=
  primitive activity activity-name { param-decl [, param-decl]* } }
  {
    { display : ID.literal-string ; }
    { priority : [ ID.unsigned | param-name ] ; }
    { random : [ ID.truth-value | param-name ] ; }
    { min duration : [ ID.unsigned | param-name ] ; }
    { max duration : [ ID.unsigned | param-name ] ; }
    { resources }
  }

activity-name ::= ID.name

param-decl ::= param-type param-name

param-type ::= ATT.type-def | listof ATT.type-def

param-name ::= ID.name

resources ::= resources : [ param-name | OBJ.object-name ]
  . . . [ , [ param-name | OBJ.object-name ]* ];

activity-ref ::= activity-name { param-expr [, param-expr]* } );

param-expr ::= GRP.group-name |
  AGT.agent-name |
  CLS.object-class-name |
  OBJ.object-name |
  COC.conceptual-class-name |
  COB.conceptual-object-name |
  ARE.area-name |
  VAR.variable-name |
  ID.number |
  ID.literal-symbol |
  ID.literal-string |
  ID.truth-value |
  list-expr

list-expr ::= ( param-expr [, param-expr]* )
```

Semantics

Declaration and reference

All activities have to be declared in the activities section of either a group, agent, class, object, or composite-activity. The declared activities can then be referenced in the workframes defined for the group, agent, class or object.

Parameters

It is possible to define input parameters for primitive activities. These input parameters can be used to make activities more generic. In the reference the values for the input parameters have to be passed.

Priority

Activities can be assigned a priority. The priorities of activities in a workframe are used to define the priority of a workframe. The workframe will get the priority of the activity with the highest priority defined in the workframe.

Duration

Activities in general have a duration. The duration of the activity can be defined to be a fixed amount of time. The random attribute has to be set to false and the max-duration attribute has to be set to the maximum duration in seconds. The duration of the activity can also be defined to be a random amount of time. To define a random amount of time the random attribute has to be set to true, the min-duration attribute has to be set to the minimum duration of the activity in seconds and the max-duration attribute has to be set to the maximum duration of the activity in seconds.

Resources

Artifacts (objects) can be defined as being a resource or not by setting the resource attribute to either true or false. In general artifacts that are worked on by agents are not considered to be a resource in an activity (a form, a fax). Artifacts that are used by an agent in an activity are considered to be resources (a fax machine, a telephone).

It is possible to associate artifacts with activities for statistical purposes and for the purpose of generating object flows by defining them in the list of resources for an activity. Artifacts which are defined as resources are also called resource objects. Resource objects associated with activities will only collect statistics and will not be used for the object flow generation. Artifacts which are defined not to be a resource and which are associated with an activity are also called touch objects. Touch objects should be associated with one or more conceptual object(s) for object flow purposes and statistical purposes.

Defaults

```
display  =<activity-name>
priority =0
random   =false
min_duration =0
max_duration=0
resources  =none
```

Constraints

1. The name of an activity must be unique within the definition of a group, agent, class, object, or composite-activity.
2. The input parameter type of a parameter defined in the declaration of an activity must be the same as the input value type or variable type in the reference of the activity.
3. The parameters assigned to any of the attributes must be of the correct type.
4. The parameter types for resources must be of type <object-class-name>, the parameters assigned to the resources must be either VAR.variable-name or OBJ.object-name.
5. The minimum duration of the activity defines the minimum duration in seconds.
6. The maximum duration of the activity defines the maximum duration in seconds.

A.17. SYNTAX OF A CREATE OBJECT ACTIVITY (COA)

Syntax

```
create-object-activity ::=  
  create object PAC.activity-name { { PAC.param-decl [ ,PAC.param-decl ]* } }  
  {  
    { display : ID.literal-string ; }  
    { priority : [ ID.unsigned | PAC.param-name ] ; }  
    { random : [ ID.truth-value | PAC.param-name ] ; }  
    { min duration : [ ID.unsigned | PAC.param-name ] ; }  
    { max duration : [ ID.unsigned | PAC.param-name ] ; }  
    { PAC.touched-objects }  
    action : [ new | copy | PAC.param-name ] ;  
    source : [ CLS.object-class-name | OBJ.object-name | PAC.param-name ] ;  
    destination : [ CLS.object-class-name | OBJ.object-name | PAC.param-name ] ;  
    { location : [ ARE.area-name | PAC.param-name ] ; }  
    { conceptual object : [ COB.conceptual-object-name | PAC.param-name ]  
      [ , [ COB.conceptual-object-name | PAC.param-name ] ] ; }  
    { when : [ start | end | PAC.param-name ] ; }  
  }
```

Semantics

Declaration and reference

All activities have to be declared in the activities section of either a group, agent, class, object, or composite-activity. The declared activities can then be referenced in the workframes defined for the group, agent, class or object.

Parameters

It is possible to define input parameters for create-object activities. These input parameters can be used to make activities more generic. In the reference the values for the input parameters have to be passed. It is recommended to make the first three parameters in a create-object activity:

1. action
2. source object
3. destination object

Priority

Activities can be assigned a priority. The priorities of activities in a workframe are used to define the priority of a workframe. The workframe will get the priority of the activity with the highest priority defined in the workframe.

Duration

Activities in general have a duration. The duration of the activity can be defined to be a fixed amount of time. The random attribute has to be set to false and the max-duration attribute has to be set to the maximum duration in seconds. The duration of the activity can also be defined to be a random amount of time. To define a random amount of time the random attribute has to be set to true, the min-duration attribute has to be set to the minimum duration of the activity in seconds and the max-duration attribute has to be set to the maximum duration of the activity in seconds.

When

The when attribute defines when the actual action has to take place, at the 'start' of the activity or at the 'end' of the activity.

Defaults

```
display = <activity-name>
priority=0
random = false
min_duration = 0
max_duration=0
resources = none
location  = <location of source object>
conceptual-object = <conceptual-object of source object>
when = end
```

Constraints

1. The name of an activity must be unique within the definition of a group, agent, class, object, or composite-activity.
2. The input parameter type of a parameter defined in the declaration of an activity must be the same as the input value type or variable type in the reference of the activity.
3. The parameters assigned to any of the attributes must be of the correct type.
4. The parameter types for resources must be of type <object-class-name> or list-of <object-class-name>, the parameters assigned to the resources must be either VAR.variable-name or OBJ.object-name or a list of any of these two.
5. The action parameter must be of type *symbol* and its input values must be one of *new* or *copy*.
6. The source parameter type has to be of type CLS.object-class-name and its input value must be one of VAR.variable-name, CLS.object-class-name, or OBJ.object-name.
7. The destination parameter type has to be of type CLS.object-class-name and its input value must be one of VAR.variable-name, CLS.object-class-name, or OBJ.object-name.
8. The location parameter type has to be of type ADF.area-def and its input value must be one of VAR.variable-name or ARE.area-name.
9. The conceptual-object parameter type has to be of type COC.conceptual-class-name or list-of COC.conceptual-class-name and its input value(s) must be one of VAR.variable-name, COB.conceptual-object-name or a list of any one of these elements.
10. The when parameter must be of type *symbol* and its input values must be one of *start* or *end*.
11. The minimum duration of the activity defines the minimum duration in seconds.
12. The maximum duration of the activity defines the maximum duration in seconds.
13. If the source of the create-object action is a class-name only the *new* action is allowed.

A.18. SYNTAX OF A MOVE ACTIVITY (MOV)

Syntax

move-activity ::=

```
move PAC.activity-name ( { PAC.param-decl [, PAC.param-decl]* } ) {  
  { display : ID.literal-string ; }  
  { priority : [ ID.unsigned | PAC.param-name ] ; }  
  { random : [ ID.truth-value | PAC.param-name ] ; }  
  { min_duration : [ ID.unsigned | PAC.param-name ] ; }  
  { max_duration : [ ID.unsigned | PAC.param-name ] ; }  
  { PAC.resources }  
  location : [ ARE.area-name | PAC.param-name ] ;  
}
```

9.4.3 Semantics

Declaration and reference

All activities have to be declared in the activities section of either a group, agent, class, object, or composite-activity. The declared activities can then be referenced in the workframes defined for the group, agent, class or object.

Parameters

It is possible to define input parameters for move activities. These input parameters can be used to make activities more generic. In the reference the values for the input parameters have to be passed. It is recommended to make the first parameter the goal-location of the move activity.

Priority

Activities can be assigned a priority. The priorities of activities in a workframe are used to define the priority of a workframe. The workframe will get the priority of the activity with the highest priority defined in the workframe.

Duration

Activities in general have a duration. In case of the move activity it is not necessary to define a duration of the activity. The duration of the activity is calculated by the simulation engine using the distance definitions defining the distance between two locations. The simulation will determine the shortest path of travel from the current location to the goal location and calculate the travel time based on the distance. The duration of the move activity will in that case be the same as the calculated travel time. It is possible however to still define the duration of the activity. If the simulation engine cannot find a travel path then the defined duration will be used. The duration of the activity can be defined to be a fixed amount of time. The random attribute has to be set to false and the max-duration attribute has to be set to the maximum duration in seconds. The duration of the activity can also be defined to be a random amount of time. To define a random amount of time the random attribute has to be set to true, the min-duration attribute has to be set to the minimum duration of the activity in seconds and the max-duration attribute has to be set to the maximum duration of the activity in seconds.

Defaults

```
display = <activity-name>  
priority=0  
random =false  
min_duration =0  
max_duration=0  
resources =none
```

Constraints

1. The name of an activity must be unique within the definition of a group, agent, class, object, or composite-activity.

2. The input parameter type of a parameter defined in the declaration of an activity must be the same as the input value type or variable type in the reference of the activity.
3. The parameters assigned to any of the attributes must be of the correct type.
4. The parameter types for resources must be of type <object-class-name> or list-of <object-class-name>, the parameters assigned to the resources must be either VAR.variable-name or OBJ.object-name or a list of any of these two.
5. The goal-location parameter must be of type <area-def-name>.
6. The minimum duration of the activity defines the minimum duration in seconds.
7. The maximum duration of the activity defines the maximum duration in seconds.

A.19. SYNTAX OF A COMMUNICATE ACTIVITY (COM)

Syntax

```
communicate-activity ::=
  communicate PAC.activity-name ( { PAC.param-decl [ , PAC.param-decl ]* } ) {
    { display : ID.literal-string ; }
    { priority : [ ID.unsigned | PAC.param-name ] ; }
    { random : [ ID.truth-value | PAC.param-name ] ; }
    { min duration : [ ID.unsigned | PAC.param-name ] ; }
    { max duration : [ ID.unsigned | PAC.param-name ] ; }
    { PAC.resources }
    { type : [ phone | fax | email | face2face | pager | none | PAC.param-name ] ; }
    with : [ [ AGT.agent-name | OBJ.object-name | PAC.param-name ]
      [ , [ AGT.agent-name | OBJ.object-name | PAC.param-name ] ] ];
    about : TDF.transfer-definition , TDF.transfer-definition ]* ;
    { when : [ start | end | PAC.param-name ] ; }
  }
```

Semantics

Declaration and reference

All activities have to be declared in the activities section of either a group, agent, class, object, or composite-activity. The declared activities can then be referenced in the workframes defined for the group, agent, class or object.

Parameters

It is possible to define input parameters for communicate activities. These input parameters can be used to make activities more generic. In the reference the values for the input parameters have to be passed. It is recommended to make the first parameter(s) in a communicate activity the concepts with which is communicated.

Priority

Activities can be assigned a priority. The priorities of activities in a workframe are used to define the priority of a workframe. The workframe will get the priority of the activity with the highest priority defined in the workframe.

Duration

Activities in general have a duration. The duration of the activity can be defined to be a fixed amount of time. The random attribute has to be set to false and the max-duration attribute has to be set to the maximum duration in seconds. The duration of the activity can also be defined to be a random amount of time. To define a random amount of time the random attribute has to be set to true, the min-duration attribute has to be set to the minimum duration of the activity in seconds and the max-duration attribute has to be set to the maximum duration of the activity in seconds.

Type

The type attribute defines what type of communication is used. The type can be one of phone, fax, email, face2face, pager, or none (meaning not specified).

When

The when attribute defines when the actual communication has to take place, at the 'start' of the activity or at the 'end' of the activity.

Defaults

```
display = <activity-name>
priority=0
random = false
```

```
min_duration = 0  
max_duration=0  
resources = none  
type = none  
when = end
```

Constraints

1. The name of an activity must be unique within the definition of a group, agent, class, object, or composite-activity.
2. The input parameter type of a parameter defined in the declaration of an activity must be the same as the input value type or variable type in the reference of the activity.
3. The parameters assigned to any of the attributes must be of the correct type.
4. The parameter types for resources must be of type <object-class-name> or list-of <object-class-name>, the parameters assigned to the resources must be either VAR.variable-name or OBJ.object-name or a list of any of these two.
5. The type-parameter type has to be a literal-symbol and its values must be one of phone, fax, e-mail, face2face, pager, or none.
6. The with-parameter type has to be one of 'Agent' , CLS.object-class-name, list-of Agent or list-of CLS.object-class-name and its input value must be one of AGT.agent-name, OBJ.object-name, VAR.variable-name or a list of any of these elements.
7. The when parameter must be of type *symbol* and its input values must be one of *start* or *end*.
8. The minimum duration of the activity defines the minimum duration in seconds.
9. The maximum duration of the activity defines the maximum duration in seconds.

Current Limitations

Currently the simulation engine does not support communication with more than one agent or object at the same time.

A.20. SYNTAX OF A BROADCAST ACTIVITY (BCT)

Syntax

```
broadcast-activity ::=
broadcast PAC.activity-name ( { PAC.param-decl [ , PAC.param-decl ]* } ) {
    { display : ID.literal-string ; }
    { priority : [ ID.unsigned | PAC.param-name ] ; }
    { random : [ ID.truth-value | PAC.param-name ] ; }
    { min_duration : [ ID.unsigned | PAC.param-name ] ; }
    { max_duration : [ ID.unsigned | PAC.param-name ] ; }
    { PAC.resources }
    { type : [ phone | fax | email | face2face | pager | none | PAC.param-name ] ; }
    { about : TDF.transfer-definition, TDF.transfer-definition ]* ;
    { when : [ start | end | PAC.param-name ] ; }
}
```

9.4.4 Semantics

Declaration and reference

All activities have to be declared in the activities section of either a group, agent, class, object, or composite-activity. The declared activities can then be referenced in the workframes defined for the group, agent, class or object.

Parameters

It is possible to define input parameters for broadcast activities. These input parameters can be used to make activities more generic. In the reference the values for the input parameters have to be passed.

Priority

Activities can be assigned a priority. The priorities of activities in a workframe are used to define the priority of a workframe. The workframe will get the priority of the activity with the highest priority defined in the workframe.

Duration

Activities in general have a duration. The duration of the activity can be defined to be a fixed amount of time. The random attribute has to be set to false and the max-duration attribute has to be set to the maximum duration in seconds. The duration of the activity can also be defined to be a random amount of time. To define a random amount of time the random attribute has to be set to true, the min-duration attribute has to be set to the minimum duration of the activity in seconds and the max-duration attribute has to be set to the maximum duration of the activity in seconds.

When

The when attribute defines when the actual broadcast has to take place, at the 'start' of the activity or at the 'end' of the activity.

Defaults

```
display = <activity-name>
priority=0
random = false
min_duration = 0
max_duration=0
resources = none
when   = end
```

Constraints

1. The name of an activity must be unique within the definition of a group, agent, class, object, or composite-activity.
2. The input parameter type of a parameter defined in the declaration of an activity must be the same as the input value type or variable type in the reference of the activity.
3. The parameters assigned to any of the attributes must be of the correct type.
4. The parameter types for resources must be of type <object-class-name> or list-of <object-class-name>, the parameters assigned to the resources must be either VAR.variable-name or OBJ.object-name or a list of any of these two.
5. The when parameter must be of type *symbol* and its input values must be one of *start* or *end*.
6. The minimum duration of the activity defines the minimum duration in seconds.
7. The maximum duration of the activity defines the maximum duration in seconds.

A.21. SYNTAX OF A PRECONDITION (PRE)

Syntax

precondition ::= [**known** | **knownval** | **unknown** | **not**] (comparison)

comparison ::= [val-comp | rel-comp]

val-comp ::=

expression BEL.evaluation-operator expression |
BEL.obj-attr BEL.equality-operator ID.literal-symbol |
BEL.obj-attr BEL.evaluation-operator BEL.object-ref |
BEL.object-ref BEL.evaluation-operator BEL.object-ref

rel-comp ::=

BEL.obj-attr REL.relation-name BEL.obj-attr { **is** ID.truth-value } |
BEL.obj-attr REL.relation-name BEL.object-ref { **is** ID.truth-value } |
BEL.object-ref REL.relation-name BEL.object-ref { **is** ID.truth-value }

expression ::= term | expression [+ | -] term

term ::= factor | term [* | / | **div** | **mod**] factor

factor ::= primary | factor ^ primary

primary ::= _ primary | element

element ::= ID.number | (expression) | BEL.obj-attr | VAR.variable-name

Semantics

Precondition modifiers

A precondition is defined with one of four modifiers: known, knownval, unknown, or not. The modifiers have the following meaning.

known:

The modifier ‘known’ represents the possibility for an agent/object to have a belief/fact, but be unspecific as to whether the agent/objects knows the actual value.

For example, to evaluate the following precondition:

```
known(the color of car1 = blue)
```

The simulation engine would simply ignore the value “blue” specified in the precondition, and look for any belief of the form:

```
the color of car1 = ?
```

If the engine finds a belief of the form, as it would when the following belief is present:

```
the color of car1 = red
```

then the engine would evaluate the precondition as true. A simple relational precondition like:

```
known(John is-the-son-of Bill is true)
```

will evaluate to true when the engine finds the belief:

John is-the-son-of Bill is true

or

John is-the-son-of Bill is false

A more complex precondition like:

```
known(the service-tech of Cimap-order1 is-the-son-of the service  
tech of the Cimap-order2)
```

will evaluate to true if the following beliefs are present:

```
the service-tech of Cimap-order1 = <agent1>  
the service-tech of Cimap-order2 = <agent2>
```

where <agent1> and <agent2> can be either values or objects.

Since this outcome may be counterintuitive for the user, we advise the user to use variables together with the ‘known-value’ modifier, whenever possible, and use the known modifier only in cases when a simple lookup of the form ‘OA-V’ is done and the user does not care about the value of the object-attribute.

knownval:

The modifier ‘knownval’ (known value) means that the simulation engine must find a precise match for the precondition. The precondition is only true if matching beliefs/facts can be found for both the left hand side and the right hand side and if the relation between them is found as well. For example for a complex precondition such as:

```
knownval(the service-tech of Cimap-order1 is-the-son-of the  
service-tech of Cimap-order2)
```

the following beliefs must be present:

```
the service-tech of Cimap-order1 = <agent1>  
the service-tech of Cimap-order2 = <agent2>  
<agent1> is-the-son-of <agent2>
```

When using variables, the engine will find as many matches as there are valid instantiations for the variables.

unknown (aka no-knowledge-of):

When the modifier ‘unknown’ is used, the simulation engine looks at the beliefs of the agent or facts in the world for objects for possible matches of the precondition. If there are any matches, the precondition evaluates to false, if no matches are found the precondition evaluates to true. The ‘unknown’ modifier can be interpreted as ‘The agent/object has no beliefs/facts for <precondition>’. However, there are intricacies that need to be explained further.

When matching a precondition of the form: $O_1A_1=O_2$ or $O_1A_1=V$, the simulation engine looks for a belief of the form $O_1A_1=?$. The right hand side of the precondition is not used in the lookup. When a belief of the form $O_1A_1=?$ is found, the simulation engine interprets this to mean that the agent ‘knows’ about this object and attribute and thus the precondition is false.

When the precondition is of the form $O_1 \text{ rel } O_2$ is true/false. However, no matter what the truth of the relation is, the simulation engine will simply look up whether the agent/object possesses the belief/fact $O_1 \text{ rel } O_2$, and if so will evaluate the precondition to be false.

All other preconditions, require at least two steps for the simulation engine to determine the truth or falsehood of the precondition. Some of these precondition forms may have counterintuitive outcomes.

The forms $O_1A_1 = O_2A_2$, $O_1A_1 \text{ rel } O_2A_2$ require the simulation engine to evaluate both the left hand side and the right hand side. When a belief/fact for either side is not found, the precondition will be evaluated to true, if both are found the precondition will evaluate to false. For example given the following beliefs:

```
the color of car1 = blue  
the color of car2 = red
```

and the precondition:

```
unknown(the color of car1 = the color of car2)
```

The simulation engine will evaluate the precondition to false, because it finds beliefs for both "The color of car1 = ?" and "The color of car2 = ?". If the beliefs of the agent were as follows however:

```
the service-tech of Cimap-order1 = John  
the service-tech of Cimap-order2 = Bill
```

and the precondition would read:

```
unknown(the service-tech of Cimap-order1 is-the-son-of the  
service-tech of Cimap-order2)
```

The simulation engine will also evaluate the precondition to be false. This may seem counterintuitive and we therefore advise the user to use this form with care.⁶⁹

The other cases are of the form: $O_1A_1 = O_2$, $O_1 = O_2A_2$, $O_1 \text{ rel } O_2A_2$. These forms are evaluated in one lookup, namely $O_1A_1 = ?$ or $O_2A_2 = ?$. If a belief/fact of this form is found, no matter what the ? stands for (could be either a value or an object), the precondition is evaluated to false. For example, given the following beliefs:

```
the car of John = car1
```

the following precondition

```
unknown(the car of John belongs-to Nynex)
```

will evaluate to false, since the simulation engine will find a belief for "The car of John". Since this may seem counterintuitive, we advise the user to use this precondition form with care.

not (aka no-matching-beliefs):

Not works similar to knownval, but negates the resulting truth-value. The simulation engine will first try the knownval for the precondition. If the precondition with the knownval modifier evaluates to true then the precondition with the not modifier evaluates to false and vice versa.

Constraints

1. The left hand side attribute type and the right hand side value-type or right hand side attribute type of a value-expression must be the same.

⁶⁹ It would be more intuitive to evaluate the precondition to "true", because the agent does not have a belief that states that John is-the-son-of Bill. The pattern matching algorithm could be changed when the simulation engine is redesigned.

2. The left hand side and right hand side types in a relational expression must match the types as defined for the relation used in the relational expression.
3. Expressions must evaluate to a value.

A.22. SYNTAX OF A CONSEQUENCE (CON)

Syntax

consequence ::= conclude ((PRE.comparison) { , fact-certainty } { , belief-certainty }) ;

fact-certainty ::= fc : ID.unsigned

belief-certainty ::= bc : ID.unsigned

Semantics

Fact certainty

The fact certainty is a number ranging from 0 to 100 and represents the percentage of chance that a fact will be created based on the consequence. A fact certainty of 0% means that no fact will be created, 100% means that a fact will be created at all times.

Belief certainty

The belief certainty is a number ranging from 0 to 100 and represents the percentage of chance that a belief will be created based on the consequence. A belief certainty of 0% means that no belief will be created, 100% means that a belief will be created at all times.

Defaults

fc = 100

bc = 100

Constraints

1. In the comparison the left hand side attribute type and the right hand side value-type or right hand side attribute type of a value-expression must be the same.
2. In the comparison the left hand side and right hand side types in a relational expression must match the types as defined for the relation used in the relational expression.
3. The values of fact-certainty and belief-certainty range from 0 to 100 and represent a percentage.
4. A consequence defined in the body of a thoughtframe can only conclude beliefs. The fact certainty argument will be ignored, a warning will be generated in case the belief-certainty is set to 0.

A.23. SYNTAX OF A DETECTABLE (DET)

Syntax

detectable ::=

```
detectable-name {  
    when ([ whenever | ID.unsigned ]) }  
    detect (( PRE.comparison ) { , detect-certainty } )  
    then detectable-action ;  
}
```

detectable-name ::= ID.name

detect-certainty ::= dc : ID.unsigned

detectable-action ::= continue | impasse | abort | complete | end_activity

Semantics

When

For each detectable needs to be specified when the agent or object can detect a certain fact. There are two options:

whenever:

This means that the detectable is checked every time a new fact is asserted in the world and for an agent also every time a new belief is asserted.

at a specified time:

For the detectable needs to be specified exactly when the detectable needs to be checked by specifying a percentage varying from 0 to 100% specifying at what percentage of the workframe completion the detectable needs to check the fact set and belief set. These kind of detectables are only checked once.

Detect-certainty

The detect-certainty is a number ranging from 0 to 100 and represents the percentage of chance that a fact will be detected based on the detectable. A detect-certainty of 0% means that the fact will never be detected and basically means that the detectable is switched off. A detect-certainty of 100% means that a fact will always be detected based on the detectable.

Detectable action

There are 5 different detectable actions possible:

continue:

Has no effect, only used for having agents or object detect facts and turn them into beliefs.

impasse:

Impasses the workframe on which the agent or object is working until the impasse is resolved.

abort:

Terminates the workframe on which the agent or object is working immediately.

complete:

Terminates the workframe on which the agent or object is working immediately, but still executes all remaining consequences defined in the workframe. All remaining activities are skipped.

end_activity:

This action type is only meaningful when used with composite activities. Causes the composite activity on which the agent or object is working to be ended.

Defaults

when = whenever
dc = 100
action = continue

Constraints

1. In the comparison the left hand side attribute type and the right hand side value-type or right hand side attribute type of a value-expression must be the same.
2. In the comparison the left hand side and right hand side types in a relational expression must match the types as defined for the relation used in the relational expression.
3. The value of the detect-certainty ranges from 0 to 100 and represents a percentage.
4. The end-activity action type can only be used when a detectable is defined in a composite activity.

A.24. SYNTAX OF AN AREA DEFINITION (ADF)

Syntax

```
area-definition ::=
  areadef area-def-name
  {
    { display : ID.literal-string ; }
    { GRP.attributes }
    { GRP.relations }
    { GRP.initial-facts }
  }
```

area-def-name ::= ID.name

Semantics

Defaults

display = <area-def-name>

Constraints

1. The name of an area definition must be unique amongst the area definitions in a model.

A.25. SYNTAX OF AN AREA (ARE)

Syntax

area ::=

```
area area-name instanceof area-def-name { partof area-name } {  
    { display : ID.literal-string ; }  
    { GRP.attributes }  
    { GRP.relations }  
    { GRP.initial-facts }  
}
```

area-name ::= ID.name

Semantics

Area Decomposition

Areas can be decomposed into sub-areas. For example a building can consist of one or more floors. The decomposition can be modeled using the part-of relationship.

Defaults

display = <area-name>

Constraints

1. The name of an area must be unique amongst the areas defined in a model.

A.26. SYNTAX OF A PATH (PAT)

Syntax

```
path-def ::=
  path path-name {
    { display : ID.literal-string ; }
    area1 : ARE.area-name ;
    area2 : ARE.area-name ;
    { distance : ID.unsigned ; }
  }
```

```
path-name ::= ID.name
```

Semantics

Distance

The distance represents the time it takes to move from area1 to area2 and vice versa. In future versions of the language the distance will represent the actual distance and based on the transportation used to travel over the path the duration will be calculated.

Defaults

```
display = <path-name>
distance=0
```

Constraints

1. The name of a path must be unique amongst the paths defined in a model.
2. The distance represents the travel duration in seconds.

A.27. SYNTAX OF A TRANSFER DEFINITION (TDF)

Syntax

transfer-definition ::= transfer-action (PRE.comparison)

transfer-action ::= send | receive

Semantics

Transfer-action

The transfer action defines the direction of the communication. The 'send' action states that belief(s) of the initiating agent or object matching the transfer definition are transferred from the initiating agent or object to the non-initiating agent or object. The 'receive' action states that belief(s) of the non-initiating agent or object matching the transfer definition are transferred from the non-initiating agent or object to the initiating agent or object.

Constraints

1. In the comparison the left hand side attribute type and the right hand side value-type or right hand side attribute type of a value-expression must be the same.
2. In the comparison the left hand side and right hand side types in a relational expression must match the types as defined for the relation used in the relational expression.

A.28. VARIABLE (VAR)

Syntax

```
variable ::= {[ assigned | unassigned ]}
          [ collectall | foreach | forone ] { ATT.type-def } variable-name
          { variable-body } ;
```

variable-name ::= ID.name

```
variable-body ::= {
                 { display : ID.literal-string ; }
                 }
```

Semantics

(Un)assigned variables

Variables can be defined to be assigned variables or unassigned variables. By default variables are assigned. This typification is required due to limitations of the simulation engine. Assigned variables are bound to a context when an instance of a workframe or thoughtframe is created. Unassigned variables can only be defined in a workframe (not in a thoughtframe) and do not get a context when an instance is created of a workframe. The unassigned variable gets a context when used in a create-object activity (unassigned variable will be bound to the destination object), or in communications to bind the variable to a context based on what is communicated.

Quantification

Variables are of one of three quantification types: collect-all, for-each and for-one. The difference between the three quantification types is the way variables are bound to a specific context of a defined type (agent, object, or other value). The difference in binding is as follows:

for-each variable:

A for-each variable is bound to only one context. For each context that can be bound to the variable a separate instance is created for the workframe in which the variable is bound.

For example in the following frame:

```
workframe DoSomething {
    variables:
        assigned foreach(Order) order;
        when ((order is-assigned-to current))
        do {
            workOnOrder();
        }
}
```

There are three Order instances in the model (order1, order2, and order3) satisfying the precondition. For this workframe three instances will be created in which the for-each variable is bound to one of the orders in each frame instantiation. This means that the agent for which the workframe is defined can only work on one order at a time and will work on them in consecutive order if no interruptions take place.

collect-all variable:

A collect-all variable can be bound to more than one context. The variable will be bound to all contexts satisfying the condition in which it is defined. Only one frame instantiation will be created as a result of the binding with the collect-all variable. If we consider the same example as for for-each variables changing the quantification of the variable to collect-all.

```
workframe DoSomething {
```

```

variables:
    assigned collectall(Order) order;
when ((order is-assigned-to current))
do {
    workOnOrder();
}
}

```

Also assume that again three orders match with the precondition based on the beliefs of the agent, then all three orders are bound to the variable and one frame instantiation will be created for the agent to work on. This means that the agent for which this workframe is defined will work on all orders at the same time.

for-one variable:

A for-one variable can be bound to only one context. Only one frame-instantiation will be created as a result of the binding with the for-one variable. A for-one variable binds to the first context satisfying the condition in which it is defined. If we consider the same example as for for-each variables changing the quantification of the variable to for-one.

```

workframe DoSomething {
variables:
    assigned forone(Order) order;
when ((order is-assigned-to current))
do {
    workOnOrder();
}
}

```

Also assume that again three orders match with the precondition based on the beliefs of the agent, then one of the orders will be bound to the variable and one frame instantiation will be created for the agent to work on. This means that the agent only works on one order and it does not matter on which order. The other two orders will not be worked on.

Defaults

A variable is by default an assigned variable unless otherwise specified.
 display =<variable-name>

Constraints

1. The name of the variable must be unique within the definition of a workframe or thoughtframe.

REFERENCES

- Agha, G. A. (1986). *ACTORS: A model of Concurrent Computation in Distributed Systems*, The MIT Press, Cambridge, MA.
- Agre, P. E. (1995). "Computation and Embodied Agency." *Informatica*, 19(4):527-535.
- Anderson, J. R. (1976). *Language, memory, and thought*, Lawrence Erlbaum Associates, Hillsdale, NJ.
- Anderson, J. R. (1993). *Rules of the mind*, Lawrence Erlbaum Associates, Hillsdale, NJ.
- Anderson, J. R., and Lebiere, C. (1998). *The atomic components of thought*, Lawrence Erlbaum Associates, Mahwah, NJ.
- Angele, J., Fensel, D., Landes, D., and Studer, R. (1991). "KARL: An executable language for the conceptual model." *6th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, Canada, pp. 1-20.
- Banks, J., J.S. Carson, and Nelson, B. L. (1996). *Discrete-Event System Simulation, 2nd Edition*, Prentice-Hall, Upper Saddle River, NJ.
- Bond, A. H., and Gasser, L. (1988). *Readings in Distributed Artificial Intelligence*, Morgan Kaufmann, San Mateo, CA.
- Bradshaw, J. M. (1996). "KAoS: An Open Agent Architecture Supporting Reuse, Interoperability, and Extensibility." *Knowledge Acquisition Workshop (KAW'96)*, Banff, Alberta, Canada.
- Bradshaw, J. M. (1997). *Software Agents*, AAAI Press/The MIT Press, Menlo Park.
- Brooks, R., A. (1991). "Intelligence without representation." *Artificial Intelligence*, 47:139-159.
- Brooks, R. A. (1997). "From Earwigs to Humans." *Robotics and Autonomous Systems*, Vol. 20(2-4):291-304.
- C&L. (1994). "SPARKS(tm) 5.0 Training Manual." Coopers & Lybrand (United States).
- Cabrol, N. A., G. Chong-Diaz, C.R. Stoker, V.C. Gulick, R. Landheim, P. Lee, and al., e. (2001). "Nomad Rover Field Experiment, Atacama Desert (Chile), Science Result Overview." *Journal of Geophysical Research*.
- Calder, R. B., Smith, J. E., Courtemanche, A. J., Mar, J. M. F., and Ceranowcz, A. Z. (1993). "ModSAF Behavior Simulation and Control." *The Third Conference on Computer-Generated Forces and Behavioral Representation*, Orlando, Fla., pp. 347-356.
- Cardelli, L., and Wegner, P. (1985). "On Understanding Types, Data Abstraction and Polymorphism." *Computing Surveys*, Volume 17(4).
- Cargile, J. (1970). "A note on 'Iterated Knowings'." *Analysis*, 30:151-155.
- Carley, K., and Newell, A. (1990). "On the nature of the social agent." *The American Sociological Association annual meeting*, Washington, DC.
- Carley, K., Park, D., and Prietula, M. (1993). "Agent Honesty, Cooperation and Benevolence in an Artificial Organization." *AAAI workshop on AI and Theories of Groups and Organizations*, Washington, DC.
- Carley, K. M., Kjaer-Hansen, J., Newell, A., and Prietula, M. J. (1992). "Plural-Soar: A Prolegomenon to Artificial Agents and Organizational Behavior." *Artificial Intelligence and Management Theory*, M. Masuch and M. Warglien, eds., North-Holland, Amsterdam. pp. 87-118.
- Carley, K. M., and Prietula, M. J. (1994a). "ACTS Theory: Extending the Model of Bounded Rationality." *Computational Organization Theory*, K. M. Carley and M. J. Prietula, eds., Lawrence Erlbaum Associates, Hillsdale, NJ.
- Carley, K. M., and Prietula, M. J. (1994b). "Computational Organization Theory." Lawrence Erlbaum Associates, Hillsdale, NJ.
- Castelfranchi, C. (1995). "Guarantees for autonomy in cognitive agent architecture." *Intelligent Agents: Theories, Architectures, and Languages*, M. Wooldridge and N. R. Jennings, eds., Springer-Verlag, Heidelberg, Germany. pp. 56-70.
- Chaikin, A. (1994). *A man on the moon: Voyages of the Apollo astronauts*, Penguin Books, New York, NY.
- Charniak, E., and McDermott, D. (1986). *Introduction to Artificial Intelligence*, Addison-Wesley Publishing Company, Reading, MA.
- Checkland, P., and Scholes, J. (1990). *Soft Systems Methodology in Action*, John Wiley & Sons Ltd., Chichester, England.
- Christie, A. M. (1999). "Simulation: An Enabling Technology in Software Engineering." The Software Engineering Institute at Carnegie Mellon University,
<http://www.sei.cmu.edu/publications/articles/christie-apr1999/christie-apr1999.html>
- Clancey, W., J. (1997a). *Situated Cognition: On Human Knowledge and Computer Representations*, Cambridge University Press.

- Clancey, W. J. (1983). "The advantages of abstract control knowledge in expert system design." *AAAI-83*, pp. 74-78.
- Clancey, W. J. (1985). "Heuristic Classification." *Artificial Intelligence*(27):289-350.
- Clancey, W. J. (1992). "Model construction operators." *Artificial Intelligence*, 53(1):1-124.
- Clancey, W. J. (1997b). "The Conceptual Nature of Knowledge, Situations, and Activity." *Human and Machine Expertise in Context*, P. Feltovich, R. Hoffman, and K. Ford, eds., The AAAI Press, Menlo Park, CA. pp. 247-291.
- Clancey, W. J., Sachs, P., Sierhuis, M., and van Hoof, R. (1998). "Brahms: Simulating practice for work systems design." *International Journal on Human-Computer Studies*, 49:831-865.
- Clancey, W. J. a. B., Conrad. (1988). "Representing control knowledge as abstract tasks and metarules." *Computer Expert Systems*, L. B. a. M. J. Coombs, ed., Springer-Verlag, Heidelberg, pp. 1-77.
- Cohen, P. R., Greenberg, M. L., Hart, D. M., and Howe, A. E. (1989). "Trial by Fire: Understanding the Design Requirements for Agents in Complex Environments." *AI magazine*, Issue 3, Fall 1989, pp. 34-48.
- Cohen, P. R., and Levesque, H. J. (1990). "Intention is choice with commitment." *Artificial Intelligence*, 42:213-261.
- Compton, W. D. (1989). *Where No Man Has Gone Before; A History of Apollo Lunar Exploration Missions*, NASA Office of Management Scientific and Technical Information Division, Washington, DC.
- Corcoran, E. (1992). "Building Networks." *Scientific American*, Issue, pp. 118-120.
- Covregaru, A. (1992). "Emergence of Meta-Level Control in Multi-Tasking Autonomous Agents," University of Michigan.
- Damer, B. (1997). *Avatars! : Exploring and Building Virtual Worlds on the Internet*, Peachpit Press, Berkeley, CA.
- Davenport, T. H. (1993). *Process Innovation: Re-engineering Work through Information Technology*, Harvard Business School Press, Boston, MA.
- Davenport, T. H. (1995). "The fad that forgot people." *Fast Company*, Issue 1, November 1995.
- Dennett, D. C. (1984). *Elbow Room: The Varieties of Free Will Worth Wanting*, Bradford Book, MIT Press, Cambridge, MA.
- Dennett, D. C. (1987). *The Intentional Stance*, Bradford Book, MIT Press, Cambridge, MA.
- Dennett, D. C. (1991). *Consciousness Explained*, Little, Brown and Company, Boston, MA.
- Dourish, P. H., J., MacLean, A., Marqvardsen, P., Zbyslaw, A. (1996). "Freeflow: Mediating between representation and action in workflow systems." *Computer Supported Collaborative Work 1996*, Cambridge, MA, pp. 190-198.
- Ehn, P. (1988). *Work-Oriented Design of Computer Artifacts*, Arbetslivcentrum, Stockholm, Sweden.
- Engeström, Y. (1991). "Activity theory and individual and social transformation." *Multidisciplinary Newsletter for Activity Theory*, 7/8:6-17.
- Flanagan, J. L., and Huang, T. S. (1997). "NSF Workshop on Human-Centered Systems: Information, Interactivity, and Intelligence (HCS)."
- Forrest, P. W. (1970). *Simulation Modelling: A Guide to Using SIMSCRIPT*, John Wiley & Sons.
- Freed, M. (1998). "Simulating Human Behavior in Complex, Dynamic Environments," Ph.D. Thesis, Northwestern University.
- Gadamer, H.-G. (1976). *Philosophical Hermeneutics*, D. E. Linge, translator, University of California Press, Berkeley, CA.
- Gasser, L. (1991). "Social conceptions of knowledge and action: DAI foundations and open systems." *Artificial Intelligence*, 47:107-138.
- Gasser, L., and Hill, R. W. (1990). "Engineering coordinated problem solving." Palo Alto, CA.
- Genesereth, M. R., and Nilsson, N. (1987). *Logical Foundations of Artificial Intelligence*, Morgan Kaufmann Publishers, San Mateo, CA.
- Genesereth, M. R., and Nilsson, N. (1994). "Software Agents." *Communications of the ACM*, 37(7):48-53.
- Gerson, E. M. (1976). "On quality of life." *American Journal of Sociology*, 41:pp. 793-806.
- Godwin, R. (1999). *Apollo 12 - The NASA Mission Reports*, Apogee Books, Burlington, Canada.
- Goodwin, R. (1993). "Formalizing properties of agents." *CMU-CS-93-159*, School of Computer Science, Carnegie-Mellon University, Pittsburgh, PA.
- Greenbaum, J., and Kyng, M. (1991). "Design at Work: Cooperative design of computer systems." Lawrence Erlbaum, Hillsdale, NJ.
- Hammer, M., and Champy, J. (1993). *Re-engineering the Corporation*, Harper Collins Publishers, Inc., New York, NY.
- Heidegger, M. (1962). *Being and Time*, J. M. a. E. Robinson, translator, Harper & Row, New York, NY.
- Hintikka, J. (1962). *Knowledge and Belief*, Cornell University Press, Ithaca, NY.

- Holmback, H., M. Greaves, and Bradshaw, J. M. (1999a). "A pragmatic principle for agent communication." *Autonomous Agents* 99, Seattle, Washington, pp. 368-369.
- Holmback, H., M. Greaves, and Bradshaw, J. M. (1999b). "What is a conversation policy?" *Autonomous Agents '99 Workshop on Specifying and Implementing Conversation Policies*, Seattle, Washington.
- Hubbard, G. S., Feldman, W. C., Cox, S. A., Smith, M. A., and Chu-Thielbar, L. (1998). "Lunar Prospector: First results and Lessons Learned." *49th International Astronautical Congress*, Melbourne, Australia.
- Hutchins, E. (1995). "How a Cockpit Remembers Its Speeds." *Cognitive Science*, 19:265-288.
- Jacobson, I. (1994). *Object-Oriented Software Engineering: A Use Case Driven Approach*, Addison-Wesley Publishing Company, Reading, MA.
- Johansson, H., Dave Carr, Patrick McHugh, William A. Wheeler, III, and A. John Pendlebury. (1992). *Business Process Reengineering*, Coopers & Lybrand.
- Jones, E., M. (1997). "The Apollo Lunar Surface Journal." National Aeronautics and Space Administration, <http://www.hq.nasa.gov/office/pao/History/alsj/>
- Jones, R., Laird, J. E., Tambe, M., and Rosenbloom, P. S. (1994). "Generating Behavior in Response to Interacting Goals." *The Third Conference on Computer-Generated Forces and Behavioral Representation*, Orlando, Fla., pp. 317-324.
- Just, M. A., and Carpenter, P. N. (1992). "A capacity theory of comprehension: Individual differences in working memory." *Psychological Review*, 99:122-149.
- Kain, R. R., Koppa, R. J., Olmsted, J. G., and Montgomery, T. O. (1972). "Apollo 16 Final Lunar Surface Procedures." Manned Spacecraft Center, Houston, TX.
- Kang, M., Waisel, L. B., and Wallace, W. A. (1998). "Team Soar: A Model for Team Decision Making." *Simulating Organizations: Computational Models of Institutions and Groups*, M. J. Prietula, K. M. Carley, and L. Gasser, eds., AAAI Press/MIT Press, Cambridge, MA. pp. 23-45.
- Karbach, W., Voss, A., Schukey, R., and Drouwen, U. (1991). "Model-K: Prototyping at the knowledge level." *Expert Systems-91*, Avignon, France, pp. 501-512.
- Kiviat, P. J., Villanueva, R., and Markowitz, H. M. (1969). *The Simscript II Programming Language*, Prentice-Hall.
- Kleindorf, G. B., L. O'Neill, and Ganeshan, R. (1998). "Validation in Simulation: Various Positions in the Philosophy of Science." *Management Science*, 44:1087-1099.
- Kling, R., Star, S. L., Kiesler, S., Agre, P., Bowker, G., Attewell, P., and Ntuen, C. (1997). "BOG4 Report: Human Centered Systems in the Perspective of Organizational and Social Informatics." *NSF Workshop on Human Centered Systems (HCS)*, Arlington, VA.
- Klir, G. J. (1985). *Architecture of Systems Complexity*, Saunders, New York.
- Konolige, K. (1982). "A first-order formalization of knowledge and action for a multi-agent planning system." *Machine Intelligence*, J. E. Hayes, D. Mitchie, and Y. Pao, eds., Ellis Horwood, Chichester, England. pp. 41-72.
- Konolige, K. (1986). *A Deduction Model of Belief*, Morgan Kaufmann, San Mateo, CA.
- Kuutti, K. (1996). "Activity theory as a potential framework for human-computer interaction research." *Context and Consciousness: Activity Theory and Human-Computer Interaction*, B. A. Nardi, ed., The MIT Press, Cambridge, MA. pp. 17-44.
- Laird, J. E., Gongdon, C. B., and Coulter, K. J. (1999). "The Soar User's Manual, Version 8.2 (Edition 1)." University of Michigan.
- Laird, J. E., Newell, A., and Rosenbloom, P. S. (1987). "Soar: An architecture for general intelligence." *Artificial Intelligence*, 33:1-64.
- Latané, B. (1981). "The psychology of social impact." *American Psychologist*, 36:343-365.
- Lave, J., Murtaugh, M., and Roche, O. d. I. (1984). "The Dialectic of Arithmetic in Grocery Shopping." *Everyday Cognition: Its Development in Social Context*, B. Rogoff and J. Lave, eds., Harvard University Press, Cambridge, MA.
- Lave, J., and Wenger, E. (1991). *Situated Learning - Legitimate peripheral participation*, University Press.
- Law, A. M., and Kelton, W. D. (1991). *Simulation Modeling and Analysis (2nd Edition)*, McGraw-Hill, New York.
- Leont'ev, A. N. (1978). *Activity, Consciousness and Personality*, Prentice-Hall, Englewood Cliffs, NJ.
- Linster, M., and Musen, M. A. (1992). "Use of KADS to create a conceptual model of the ONCOCIN task." *Knowledge Acquisition*, 4(1)(Special issue: The KADS approach to knowledge engineering).
- Markowitz, H. M., Hausner, B., and Karr, H. W. (1963). *SIMSCRIPT: A Simulation Programming Language*, Prentice-Hall, Englewood Cliffs, N.J.
- Marshall, E. M. (1995). *Transforming the Way We Work: The Power of the Collaborative Workplace*, American Management Association, New York.
- McCarthy, J. (1978). "Ascribing mental qualities to machines." Stanford University AI Lab, Stanford, CA.

- Mead, G. H. (1934). *Mind, Self, and Society; From the standpoint of a social behaviorist*, University of Chicago Press, Chicago, IL.
- Meyer, D. E., and Kieras, D. E. (1997). "A computational theory of executive cognitive processes and multiple-task performance: Part 1. Basic mechanisms." *Psychological Review*, 104:3-65.
- Mize, J. H., and Cox, J. G. (1968). *Essentials of Simulation*, Prentice-Hall.
- Nardi, B. A. (1996). *Context and Consciousness: Activity Theory and Human-Computer Interaction*, The MIT Press, Cambridge, MA.
- NASA. (1969). "Apollo 12 Press Kit." National Aeronautics and Space Administration, Washington, DC.
- NASA. (1971). "Apollo 14 Press Kit." Release No. 71-3, National Aeronautics and Space Administration, Washington, DC.
- NASA. (1972). "Apollo Lunar Surface EVA Video." NASA/Johnson Space Center, Information Science Branch Video Repository (GP28), Houston, TX.
- NASA. (2000). "Research in Intelligent Systems." NRA2-37143, NASA Ames Research Center, Moffett Field, CA.
- Newell, A. (1973a). "Production systems: Models of control structures." *Visual Information processing*, pp. 463-526.
- Newell, A. (1973b). "You can't play 20 questions with nature and win: Projective comments on the papers of this symposium." *Visual information processing*, pp. 283-310.
- Newell, A. (1982). "The knowledge level." *Artificial Intelligence*(18):87-127.
- Newell, A. (1990). *Unified theories of cognition*, Harvard University Press, Cambridge, MA.
- Newell, A., and Simon, H. A. (1972). *Human Problem Solving*, Prentice-Hall, Englewood Cliffs, NJ.
- Nonaka, I., and Takeuchi, H. (1995). *The Knowledge-Creating Company*, Oxford University Press, Oxford.
- Nowak, A., and Latané, B. (1994). "Social dilemmas exist in space." Social dilemmas and cooperation, N. Schultz, W. Albers, and R. N. Mueller, eds., Springer-Verlag, Heidelberg.
- Polanyi, M. (1983). *The Tacit Dimension*, Peter Smith, Magnolia, MA.
- Preece, R. (1994). *Starting Research: An introduction to academic research and dissertation writing*, Pinter Publishers, London, UK.
- Prietula, M. J., Carley, K. M., and Gasser, L. (1998). *Simulating Organizations: Computational Models Of Institutions and Groups*, AAAI/MIT Press, Menlo Park, CA.
- Robinson, S. (1994). *Successful Simulation: A Practical Approach to Simulation Projects*, McGraw-Hill, Maidenhead, UK.
- Robinson, S. (1999). "Simulation Verification, Validation and Confidence: A Tutorial." *Transactions of The Society for Computer Simulation International*, Vol. 16(Number 2):63-69.
- Rogoff, B., and Lave, J. (1984). *Everyday Cognition: It's development in Social Context*, Harvard University Press, Cambridge, MA.
- Rumbaugh, J., Jacobson, I., and Booch, G. (1998). *The Unified Modeling Language Reference Manual*, Addison-Wesley.
- Sachs, P. (1995). "Transforming Work: Collaboration, Learning, and Design." *Communications of the ACM*, 38(9):pp. 36-44.
- Schön, D. A. (1982). *The Reflective Practitioner: How Professionals Think in Action*, Basic Books.
- Schreiber, G. (1992). "Pragmatics of the Knowledge Level," Ph.D. thesis, University of Amsterdam, Amsterdam.
- Schreiber, G., Akkermans, H., Anjewierden, A., Hoog, R. d., Shadbolt, N., Velde, W. v. d., and Wielinga, B. (2000). *Knowledge Engineering and Management: The CommonKADS Methodology*, The MIT Press, Cambridge, MA.
- Schreiber, G., Wielinga, B., and Breuker, J. (1993). *KADS: A Principled Approach to Knowledge-Based System Development*, Academic Press.
- Scott Morton, M., S. (1991). *The Corporation of 1990s, Information Technology and Organizational Transformation*, Oxford University Press, New York, NY.
- Searle, J., R. (1969). *Speech Acts*, Cambridge University Press, Cambridge, UK.
- Searle, J., R. (1975). "A taxonomy of illocutionary acts." *Language, Mind, and Knowledge*, K. Gunderson, ed., University of Minnesota, Minneapolis. pp. 344-369.
- Selvin, A., Shum, S. B., Sierhuis, M., Conklin, J., Zimmermann, B., Palus, C., Drath, W., Horth, D., Domingue, J., Motta, E., and Li, G. (2001). "Compendium: Making Meetings into Knowledge Events." *Knowledge Technologies 2001*, Austin, TX.
- Selvin, A. M., and Sierhuis, M. (1999a). "Argumentation in Different CSCA Project Types." *Workshop Computer-Supported Collaborative Argumentation for Learning Communities held at Computer-Supported Collaborative Learning'99*, Stanford, CA.

- Selvin, A. M., and Sierhuis, M. (1999b). "Case Studies of Project Compendium in Different Organizations." *Workshop Computer-Supported Collaborative Argumentation for Learning Communities held at Computer-Supported Collaborative Learning'99*, Stanford, CA.
- Shoham, Y. (1989). "Time for action." *Proceedings IJCAI-89*, Detroit, MI, pp. 954-959.
- Shoham, Y. (1993). "Agent-oriented programming." *Artificial Intelligence*, 60(1):51-92.
- Sierhuis, M. (1986). "PWES: Prototype Woningdelaars Expert System, een experimenteel expert system," *Gemeentelijke HTS*, Den Haag.
- Sierhuis, M. (2000). "Modeling and Simulation of Work Practices on the Moon." *Computational Analysis of Social and Organizational Systems 2000*, Carnegie Mellon University, Pittsburgh, PA.
- Sierhuis, M., and Clancey, W. J. (1997). "Knowledge, Practice, Activities, and People." *Proceedings of AAAI Spring Symposium on Artificial Intelligence in Knowledge Management*, <http://ksi.cpsc.ucalgary.ca/AIKM97/AIKM97Proc.html>
- Sierhuis, M., Clancey, W. J., Hoof, R. v., and Hoog, R. d. (2000a). "Modeling and Simulating Human Activity." *Autonomous Agents 2000 workshop on Intelligent Agents for Computer Supported Cooperative Work: Technology and Risks*, Barcelona. Spain.
- Sierhuis, M., Clancey, W. J., Hoof, R. v., and Hoog, R. d. (2000b). "Modeling and Simulating Human Activity." *AAAI Fall Symposium on Simulating Human Agents*, North Falmouth, MA, pp. 100-110.
- Sierhuis, M., Clancey, W. J., Hoof, R. v., and Hoog, R. d. (2000c). "Modeling and simulating work practices from Apollo12." *International Workshop on Simulation for European Space Programmes*, ESTEC, Noordwijk, The Netherlands.
- Sierhuis, M., and Selvin, A. M. (1996). "Towards a framework for collaborative modeling and simulation." *presented at the Workshop on Strategies for Collaborative Modeling and Simulation, CSCW '96*, Boston, MA.
- Sierhuis, M., Sims, M. H., Clancey, W. J., and Lee, P. (2000d). "Applying multiagent simulation to planetary surface operations." *COOP'2000 workshop on Modelling Human Activity*, Sophia Antipolis, France, pp. 19-28.
- Simon, H. (1955). "A behavioral model of rational choice." *Quarterly Journal of Economics*, 69:99-118.
- Simon, H. (1956). "Rational choice and the structure of the environment." *Psychological Review*, 69(4):493-513.
- Simon, H. (1976). *Administrative Behavior*, Free Press, New York, NY.
- Sims, M. H., Sierhuis, M., and Lee, P. (2000). "Understanding the Limits of Robot Collaboration in Field Camp Deployment on Mars." NASA Ames Research Center, Moffett Field, CA.
- Spudis, P. D. (1999). "Robots vs. Humans: Who Should Explore Space?" *Scientific American*, Issue 1, Spring 1999, pp. 24-31.
- Stasser, G. (1988). "Computer Simulation as a Research Tool: The DISCUSS Model of Group Decision Making." *Journal of Experimental Social Psychology*, 24:p. 393-422.
- Steenvoorden, E. C. (1995). "The geography of WorkFrame," MS. thesis, Centre of Excellence CIBIT, Utrecht, The Netherlands.
- Stewart, A. T. (1997). *Intellectual Capital: The New Wealth of Organizations*, Doubleday, New York.
- Suchman, L. A. (1987). *Plans and Situated Action: The Problem of Human Machine Communication*, Cambridge University Press, Cambridge, MA.
- Tambe, M., Johnson, W. L., Jones, R. M., Koss, F., Laird, J. E., Rosenbloom, P. S., and Schwamb, K. (1995). "Intelligent Agents for Interactive Simulation Environments." *AAAI Magazine*(Spring '95):pp. 15-39.
- Thomas, B. (1993). "PLACA, An Agent-Oriented Language," Ph.D. thesis, Department of Computer Science, Stanford University, Stanford, CA.
- Thomas, G., M. Reagan, E. A. Bettis III, N. Cabrol, and Rathe, A. (1999). "Analysis of science team activities during the 1999 Marsokhod rover field experiment: Implications for automated planetary surface exploration." The University of Iowa, Iowa City.
- Thorpe, J. A., Shiflett, J. E., Bloedorn, G. W., Hayes, M. F., and Miller, D. C. (1989). "The SIMNET Network and Protocols." Nr. 7102, BBN Systems and Technologies Corporation.
- Tocher, K. D. (1963). *The Art of Simulation*, Van Nostrand Company, Princeton, NJ.
- Tokoro, M. (1996). *Proceedings of the Second International Conference on Multi-Agent Systems*, Kyoto, Japan.
- Torok, D. (1992). "SPARKS: The Construction of the T.1 Process Models." NYNEX Science & Technology, Inc., White Plains, NY.
- Torrance, M. (1991). "The AGENT0 Programming Manual." Department of Computer Science, Stanford University, Stanford, CA.
- Tulving, E. (1969). "Episodic and semantic memory." *Organization of Memory*, E. Tulving and W. Donaldson, eds., Academic, New York.

- van Dijck, J. E., Hartog, J. A. d., Hoog, R. d., Neeteson, M. J., Wester, P., and Wielinga, B. J. (1987). "Eindrapport Project Experimenteel Expertsysteem." Gemeentelijke Social Dienst Amsterdam, Amsterdam.
- van Harmelen, F., and Balder, J. R. (1992). "(ML)2: a formal language for KADS models of expertise." *Knowledge Acquisition*, 4(1)(Special issue: The KADS approach to knowledge engineering).
- Van Melle, W. (1979). "A domain-independent production rule system for consultation programs." *Proceedings of the Sixth International Joint Conference on Artificial Intelligence*, Cambridge, MA, pp. 923-925.
- Vygotsky, L. S. (1978). *Mind in Society: The Development of Higher Psychological Processes*, Harvard University Press, Cambridge, MA.
- Wall, S. D., and Ledbetter, K. W. (1991). *Design of Mission Operations Systems for Scientific Remote Sensing*, Taylor & Francis, London.
- Weisbord, M. R. (1987). *Productive Workplaces: Organizing and Managing for Dignity, Meaning, and Community*, Jossey-Bass Inc., Publishers, San Francisco, CA.
- Weiss, G. (1999). *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, The MIT Press, Cambridge, MA.
- Wenger, E. (1997). *Communities of Practice; Learning, meaning, and identity*, Cambridge University Press.
- Wilhelms, D. E. (1993). *To a Rocky Moon: A Geologist's History of Lunar Exploration*, The University of Arizona Press, Tucson.
- Winograd, T., and Flores, F. (1986). *Understanding Computers and Cognition*, Addison-Wesley Publishing Corporation, Menlo Park, CA.
- Wooldridge, M. (1992). "The logical Modelling of Computational Multi-Agent Systems," Ph.D. thesis, Department of Computation, UMIST, Manchester, UK.
- Wooldridge, M., and Jennings, N. R. (1995). "Intelligent Agents: Theory and Practice." *Knowledge Engineering Review*.
- Wynn, E. (1991). "Perspective, Modeling and Social Reality." *Critical Issues*, ACM.
- Yourdon, E. (1989). *Modern Structured Analysis*, Prentice Hall, Englewood Cliffs, NJ.
- Zeigler, B. P., H. Praehofer, and Kim, T. G. (2000). *Theory of Modeling and Simulation*, Academic Press, San Diego, CA.
- Zimmerman, D. E., and Muraski, M. L. (1995). *The Elements of Information Gathering*, The Oryx Press, Phoenix, AZ.

SUMMARY

Many organizations spend a lot of time and effort analyzing the knowledge people within the organization use to perform their work. This is often done to make the work process more efficient, or to manage the knowledge resources (man, machine, or other) better. To do this, the first thing that is often done is to model the current state of the work process, after which the process is redesigned to be more efficient. Modeling work processes is a very complex task that clearly needs to be supported by effective modeling tools. There are a number of modeling tools and approaches that are used today. However, none of these approaches allow for the representation of a work process at a *work practice level*—the level at which people within the process perform their work in the real world.

This dissertation presents a methodology and supporting tool for modeling and simulating the work practice. The main research question being answered is the question of representing people working together, collaborating and communicating, situated in the real world, using tools and creating artifacts, all the while being constraint by the environment they are in. The thesis first reviews existing modeling tools and models that, in one way or another, represent people as cooperating agents. It is shown that every tool being reviewed lacks, in some fundamental way, the ability to represent all the important aspects from the theory of modeling work practice. The work presented here is testing a new computer language and methodology, with supporting tools for modeling and simulating work practice. The software tool presented and tested—and proven to be a significant improvement over available modeling tools—is called Brahms. Specifying how the Brahms modeling language can be used to represent a work practice operationalizes the presented theory for modeling and simulating work practice. The second part of the thesis presents the application of Brahms in three real-world work practice case studies.

A work practice is defined as the collective activities of a group of collaborating people who communicate together, while performing synchronous and asynchronous activities. Most often, people view work merely as the process of transforming input to output. This thesis claims that the individual activities, that make up a work practice, not only have to do with the transformation of input to output, but more important with the collaboration between individuals in action, in pursuit of a goal. Imagine soccer players who collaborate their activities in pursuit of scoring a goal. Just focusing on the input and output of the activities of the players would not only be very difficult, if not impossible, but it would miss the opportunity to understand what is really going on.

This dissertation presents a different view, namely describing work as a practice, a collection of psychologically and socially situated collaborative activities of the members of a group. The purpose of modeling a work practice is to understand how, when, where, and why collaborative activities are performed, and to identify the effects of these activities. Besides this, it is also important to understand the reasons why these activities occur in the way they do. The central theme is to find a representation for modeling work practice. The thesis first defines what is meant by the term *work practice*, and how it relates to communities of practice, activities, collaboration, communication, artifacts, and geography. Then, the Brahms multiagent modeling language is presented. Brahms models can be simulated to show the effects of the activities of groups of people, their collaboration and communication, while situated in a geographical environment, using tools and artifacts to perform their collaborative work.

The Brahms software tool is applied, verified, and evaluated in a case study of the ALSEP Offload task performed during the Apollo 12 lunar mission. This first case study is a study about the use of Brahms for developing *descriptive* models. This case study shows a detailed simulation model of the work practice of the Apollo 12 astronauts offloading the ALSEP on the Moon. The second case study shows the application of Brahms and its associated methodology in the development of a *predictive* model. This subject of this case study is predicting astronaut behavior during error situations and the communication patterns during the ALSEP deployment activity on the Moon. This case study is based on the historical data from two work practices, namely the Heat Flow Experiment deployment during the Apollo 15 and 16 missions. The subject of the third and last case study is the design of a *prescriptive* Brahms model, for a work system design of mission operations for a future robotic mission to the Moon. The thesis ends with an evaluation of the use of Brahms in the case studies to answer the research question. Besides the main research question, a

secondary research question is answered. What is the added value of a model-based approach? The answer is given based on the modeling methodology used in the case studies.

The National Aeronautics and Space Administration, under the NASA Cross Enterprises Program, funded the research presented here.

SAMENVATTING

Er zijn veel organisaties die tijd spenderen aan het analyseren van de kennis die de mensen binnen een organisatie gebruiken, om hun werk te kunnen doen. Vaak wordt dit gedaan om het werk proces te verbeteren, of om de kennisbronnen (mens, machine of andere vormen) beter te kunnen beheren. Om dit te kunnen doen creëert men steeds vaker eerst een model van het werkproces zoals het op dat moment bestaat, waarna het proces wordt herontworpen om meer efficient te kunnen zijn. Het modelleren van werkprocessen is erg gecompliceerd en het is duidelijk dat voor deze taak ter ondersteuning effectieve modeleringsgereedschappen gebruikt zouden moeten worden. Vandaag de dag zijn er een aantal modeleringsgereedschappen die gebruikt kunnen worden. Het probleem is echter dat geen van deze gereedschappen het mogelijk maakt het werkproces te representeren op het ‘work practice level’—het werkniveau van de mensen binnen het proces zoals het in de praktijk ook werkelijk wordt uitgevoerd.

De dissertatie presenteert een methodologie en een bijbehorend gereedschap voor het modelleren en simuleren van ‘*work practice*’ (vertaling: werk zoals het in de praktijk wordt uitgevoerd). De voornaamste vraag die wordt gesteld en beantwoordt, is de vraag naar een computertaal voor het representeren van samenwerkende mensen, collaborerend en communicerend, gesitueerd in de wereld, gebruikmakend van gereedschappen voor het creëren van artifacten en terwijl gelimiteerd door hun werkomgeving. Als eerste worden er een aantal bestaande modellering tools en modellen besproken, die op een of andere manier mensen representeren als cooperatieve agenten. Aangetoond wordt dat deze tools geen van allen de mogelijkheid bieden om alle belangrijke aspecten uit de theorie van het modelleren van ‘*work practice*’ weer te geven. Het werk dat hier wordt gepresenteerd is een test van een nieuwe computertaal en methodologie, met bijbehorende software tools, voor het modelleren en simuleren van een ‘*work practice*’. De software tool die wordt gepresenteerd en getoetst in de werkelijkheid—en waarvan wordt aangetoond dat deze significant beter is dan de tools en modellen die zijn beschreven—is Brahms. De gepresenteerde theorie voor het modelleren en simuleren van ‘*work practice*’ wordt geoperationaliseerd door te specificeren hoe de Brahms taal kan worden gebuikt in het weergeven van een *work practice*. Het tweede deel van de thesis is de presentatie van het gebruik van Brahms in drie realistische ‘*work practice*’ case studies.

Een ‘*work practice*’ wordt gedefinieerd als de collectieve activiteiten van een groep samenwerkende en onderling communicerende mensen, tijdens het uitvoeren van synchrone en asynchrone activiteiten. Vaak wordt werk alleen gezien als een proces van transformeren van input tot output. In deze thesis wordt gesteld dat de individuele activiteiten, die een ‘*work practice*’ definiëren, niet alleen te maken hebben met de transformatie van input tot output, maar bovenal als de samenwerking tussen individuen in hun doelgerichte acties. Denk bijvoorbeeld aan een voetbaleftal waarin de spelers samenwerken in hun gezamelijke activiteiten om te kunnen scoren. Het focussen op de input en output van individuele activiteiten van de spelers, is niet alleen erg moeilijk, als het überhaupt mogelijk is, maar mist ook de grote mogelijkheid om te kunnen begrijpen wat er werkelijk aan de gang is.

In deze dissertatie wordt een andere opvatting van werk gepresenteerd, namelijk het beschrijven van werk als een ‘*practice*’, een collectie van psychologische en sociaal gesitueerde collaboratieve activiteiten van een groep van mensen. Het doel van het modelleren van een ‘*work practice*’ is om te kunnen begrijpen hoe, wanneer, waar en waarom collaboratieve activiteiten worden uitgevoerd en om de effecten van deze activiteiten te kunnen achterhalen. Daarnaast is het van belang om de manier waarop deze activiteiten worden uitgevoerd te achterhalen. Daarom is het centrale thema een goede representatie te vinden voor het modelleren van ‘*work practice*’. Als eerste wordt gedefinieerd wat er bedoeld wordt met de term ‘*work practice*’ en deze term wordt gerelateerd aan ‘*communities of practice*’, activiteiten, collaboratie, communicatie, artifacten en geografie. Daarna wordt de Brahms multiagent modelerings taal gepresenteerd. Brahms modellen kunnen worden gesimuleerd, om te kunnen laten zien wat de effecten zijn van activiteiten van groepen van mensen, hun samenwerking en onderlinge communicatie, gesitueerd in een geografische omgeving, gebruikmakend van gereedschappen en artifacten tijdens hun collaboratieve werkzaamheden.

De Brahms software tool is gebruikt, geverifieerd en geëvalueerd in een case studie van de ALSEP Offload taak tijdens de Apollo 12 lunar missie. Deze eerste case studie is een studie naar het gebruik van Brahms voor het ontwikkelen van *beschrijvende* modellen. De case studie beschrijft een gedetailleerde

simulatiemodel van de ‘work practice’ van de Apollo 12 astronauten, tijdens het afladen van de ALSEP instrumenten op de maan. In de tweede case studie, worden Brahms en de bijbehorende modelleringsmethodologie gebruikt voor het ontwikkelen van een voorspellingsmodel. Het onderwerp van deze case studie is het *voorspellen* van het gedrag van astronauten tijdens problematische situaties en de communicatiepatronen gedurende het opzetten van de ALSEP instrumenten op de maan. De case studie is gebaseerd op de historische data van twee “work practices”, de ‘Heat Flow Experiment deployment’ tijdens de Apollo 15 en 16 missies. Het onderwerp van de derde en laatste case studie is het ontwerp van een *voorschrijvend* Brahms model voor een ‘work system design’ van ‘mission operations’ voor een toekomstige robot missie naar de maan. De thesis eindigt met een evaluatie van het gebruik van Brahms in de case studies, als antwoord op de onderzoeksraag. Naast deze hoofdonderzoeksraag is er een tweede onderzoeksraag die wordt beantwoord. Wat is de toegevoegde waarde van een ‘model-based approach’? Het antwoord wordt gegeven aan de hand van de modelleringsmethodologie die is gebruikt in de case studies.

De National Aeronautics and Space Administration, als onderdeel van het NASA Cross Enterprises Programma, heeft het hier gepresenteerde onderzoek mogelijk gemaakt.