Artificial Intelligence for Engineering Design, Analysis and Manufacturing (2009), *23*, 357–371. Printed in the USA.

doi:10.1017/S0890060409990059

Workflow agents versus expert systems: Problem solving methods in work systems design

WILLIAM J. CLANCEY,[1,2] MAARTEN SIERHUIS,[3] AND CHIN SEAH[4]

[1]NASA Ames Research Center, Moffett Field, California, USA
[2]Florida Institute for Human and Machine Cognition, Pensacola, Florida, USA
[3]Carnegie Mellon University Silicon Valley, NASA Ames Research Center, Moffett Field, California, USA
[4]Stinger Ghaffarian Technologies, NASA Ames Research Center, Moffett Field, California, USA

(RECEIVED December 1, 2008; ACCEPTED January 15, 2009)

Abstract

During the 1980s, a community of artificial intelligence researchers became interested in formalizing problem solving methods (PSMs) as part of an effort called "second-generation expert systems." We provide an example of how we are applying second-generation expert systems concepts in an agent-based system for space flight operations, the orbital communications adapter mirroring system (OCAMS), which was developed in the Brahms multiagent framework. Brahms modeling language provides an ontology for simulating work practices, including groups, agents, activities, communications, movements, and geographic areas. *Activities* are a behavioral unit of analysis to be contrasted with *tasks*, a functional unit of analysis. Problem solving occurs in the context of activities in the service of tasks; appropriate PSMs depend on the context: which people/roles are participating, what tools are available, how the results will be evaluated, and so forth. A work practice simulation facilitates designing workflow tools that appropriately interact with the physical and organizational context in which work occurs. OCAMS was developed using a simulation-to-implementation methodology, in which a prototype workflow tool was embedded in a Brahms simulation of how people would use the tool. The reusable components in a workflow system like OCAMS include entire "problem solvers" (e.g., a planning subsystem), interoperability frameworks, and agents that inspect and change the world. Thus, a tool kit for building workflow tools requires more than a library of PSMs, which play a relatively small role in the overall multiagent, systems-integration architecture. Our research concern has shifted to situations that may arise that are outside the OCAMS' capability. In practical decision making, people must reflect on the validity of their models. As programs becoming actors in the workplace, we need to develop systems that help people to understand the limitations of the models that drive the automated operations, which means in part detecting when the formalizations in the system are inadequate.

Keywords: Model-Based Automation; Problem Solving Agent; Situated Cognition; Work Practice Simulation; Work Systems Design

1. INTRODUCTION

> We can foresee two beneficial effects. The first and most obvious will be the development of knowledge systems that replicate and autonomously apply human expertise. . . . As an inevitable side effect, knowledge engineering will catalyze a global effort to collect, codify, exchange, and exploit applicable forms of human knowledge. (Hayes-Roth et al., 1983, p. xii)

During the 1980s, a community of computer scientists working in the area of artificial intelligence (AI) became interested in formalizing problem solving methods (PSMs) in an effort called "second-generation expert systems" (2nd GES; David et al., 1993). The idea was in the air as soon as the first collection of expert systems was in hand, representing competing frameworks at the San Diego Workshop on Building Expert Systems in 1980. The question was posed, "(Can we) identify the subcomponents of these tools that would provide a tool-building tool-kit"? (Hayes-Roth et al., 1983, p. 345).

The 2nd GES effort initially involved analyzing the expert systems built in the 1970s to abstract their design and operation. These efforts included general reasoning-based formalisms (e.g., Representation Language Language; Hayes-Roth et al., 1983, p. 314) and toolkits called "skeletal systems" (p. 286). The motivations included producing higher level frameworks that would make building expert systems (called "knowledge

Reprint requests to: William J. Clancey, Intelligent Systems Division, M/S 269-3, NASA Ames Research Center, Moffett Field, CA 94035, USA. E-mail: william.j.clancey@nasa.gov

acquisition") more efficient, making the programs more robust and powerful (by incorporating general principles rather than many isolated facts and heuristics), facilitating explanation of reasoning to users (especially in instructional applications), and facilitating reuse of the constructs in developing other expert systems (by developing modeling languages).

The 2nd GES analytic effort was also influenced by and occurring as part of the simulation of human problem solving in psychology (Newell & Simon, 1972), which showed the power of a production rule formalism for automating reasoning, that is, for programmatically manipulating a model of systems and processes in the world. Analysis of human problem solving protocols showed that there were patterns, called "methods," by which people applied and configured finer-grained "operators" (e.g., legal moves on a game board) in a process called "search in a problem space." Analyzing dozens of programs in different domains, ranging from medicine to physics, and spanning a variety of system-manipulation tasks (e.g., troubleshooting, design, control), 2nd GES researchers identified these additional patterns:

- All expert systems are "model based": symbolic representations of AI programming, specifically in expert systems, introduce a new modeling method to science and engineering, namely, a means of *modeling processes qualitatively* (Clancey, 1985, 1986, 1989) in contrast to purely numeric programming.
- Domain models (taxonomies, causal networks, definitions) should be represented separately from the procedures that manipulate them, thus facilitating explanation and reuse (Clancey & Letsinger, 1981).[1]
- Solving particular problems (e.g., diagnosing a patient) involves creating situation-specific[2] models, a process that was explicated particularly well in the blackboard architecture (Hayes-Roth et al., 1983, p. 308; Nii, 1986*a*, 1986*b*) by the explicit posting and comparison of model components (Clancey, 1992).
- Processes for manipulating models can be abstracted on different levels, ranging from graph operators to entire frameworks for doing diagnosis, design, and so forth (Clancey, 1985; Clancey & Barbanson, 1993).

Researchers summarized the conclusions of 2nd GES research in different ways, partly because we worked in different domains, with different purposes, and preferred different formalisms. Consolidating our work was complicated because we were generalizing from a variety of representational formalisms for modeling objects and processes in different domains (e.g., symptom–treatment rules in medicine vs. function–structure diagrams in electronics), as well as a variety of procedural methods (aka "control structures") for manipulating models for different modeling purposes (e.g., design, diagnosis, control). Alternative frameworks for organizing modeling languages and methods are provided by Chandrasekaran and Johnson (1993) and Clancey (1992).

Depending on theoretical and practical objectives, different analytic perspectives will be preferred and useful. However, the motivations of 2nd GES were broadly shared and not much in dispute, and the problem of completely modeling all of the entities and processes of a chosen domain (e.g., in classification and causal models) seemed tractable. Instead, the debate and ambiguities concerned how to abstract and describe the representation languages and model manipulation procedures in the expert system, that is, to identify, describe, and formalize the PSMs.

1.1. Abstraction in software engineering

More than two decades later, software engineering has not been transformed into a process of assembling PSMs from a library, as some had imagined. Has there been a failure to appreciate the benefits of a PSM library or was the vision (see opening quotation) of autonomous expert systems and global knowledge bases incomplete?

Software libraries are important and useful today. Useful modeling abstractions (e.g., XBRL) require less metadescription than we expected after seeing the advantages of systems like Representation Language Language. However, the built-in display, interactivity, and networking constructs available (e.g., for programming on mobile phones) were only a glimmer in our minds when we first used window–menu–mouse systems in the 1970s.

Nevertheless, building complex programs remains a craft; as researchers we are still aspiring to create libraries of modeling components, as a Holy Grail of system building in given domains, such as spaceflight operations. The appropriate "grain size" for PSMs (or software reuse more generally) is perhaps more variable and larger than we expected. Some programs use a specialized modeling language (e.g., a science database); some carry out high-level modeling tasks (e.g., spacecraft scheduling tools); others mediate between such programs (e.g., "middleware" that relates spacecraft telemetry to an interpretive model). In some respects, our experience confirms the perspective of Newell and Simon (1972), restated by McDermott (1988), that the reusable component or method is an encompassing *computational process*, a self-contained package that today we call a "software agent" (e.g., see Bonasso et al., 1997). For example, Choo and Skura (2004, p. 4) describe a toolkit of such relatively high-level components for developing simulations of space operations:

> SciBox uses the data analysis components from [the System Independent] layer to build data analysis packages specific to space operation simulations but not specific to any

[1] Medical AI researchers were also influenced in the 1970s by the efforts of the medical community to develop a Systematized Nomenclature of Medicine, an ontology project dating to the mid-1960s.

[2] "Situation-specific" refers to a particular case, setting, or scenario. It should not be confused with "situated cognition," which refers to how people are conceiving and thus coordinating their identity, values, and activities in an ongoing process enabled by higher order consciousness (Clancey, 1997*a*, 1999).

https://doi.org/10.1017/S0890060409990059 Published online by Cambridge University Press

particular space mission. Examples of SciBox software components are common mathematical algorithms used in celestial mechanics and astronomy, map projection, coordinate transformation, and scheduling and commanding.

1.2. A systemic view of problems and methods

In contrast, a more complicated story can be told that relates the role of PSMs in software engineering to how people developing practical tools for the workplace now interpret the phrase "problem solving method." In short, the nature of what constitutes a problem, how problems are solved in practice, and how methods are articulated and shared has advanced remarkably since the social scientists became involved in this research in the 1980s (e.g., Greenbaum & Kyng, 1991; Shalin, 2005). In a workplace, PSMs are now understood to be frequently interactive (e.g., getting assistance) rather than purely mental operations.[3] Work "tasks" (e.g., medical diagnosis; Vicente, 1997) are role defined and integrated in physically and organizationally located activities[4] (Clancey, 2002). Available resources and urgencies determine who is allowed to participate in handling a situation (i.e., *whose knowledge* is employed), and consequently, what methods are used (e.g., compare the diagnostic responsibilities of a nurse to a physician in a clinic, contrasted with the tools used by a medic on a battlefield). Furthermore, although much routine work can be formalized and automated, dealing with problematic situations often involves a degree of improvisation, in which policies and procedures are reinterpreted or worked around (Wynn, 1991; Dourish & Button, 1998). Together, these considerations mean that if an expert system is to be autonomous in the sense of operating without negotiating interpretations and methods with people (see the opening quote), then it must deal only with highly routine tasks[5] or be designed as a tool that can understand and adapt to the flow, rhythm, and changing priorities of human activities (Clancey, 2005).[6]

What are the implications of this practice-based perspective on problem solving for PSM research? Are PSMs (exemplified by the 2nd GES effort) relegated to a relatively minor role in software engineering? Or might PSMs be reconceived to incorporate the constraints of interactivity and improvisation imposed by practical operational environments? We explore these questions in this paper, examining our experience in developing spaceflight operations systems for NASA.

1.3. Challenges to explicating the new perspective

Explicating how ideas about problem solving have changed in cognitive science over the past 30 years is difficult for several reasons. First, pivotal assumptions that were tacit in the AI community in the 1970s might seem bizarre to younger researchers today, notably the idea that expert systems would be standalone ("autonomous") problem solvers (see opening quotation) or the idea that knowledge engineers would systematically collect and catalog global knowledge (e.g., CYC). We did not anticipate in the 1970s how personal computers and networked operations would provide different opportunities for automation. For example, a NASA (1980) report prepared by AI scientists, in a study group chaired by Carl Sagan and Raj Reddy, details roles for machine intelligence in the planned space station. However, obviously biased by the vision of interplanetary probes and laboratories, the report wrongly concluded that data distribution, science data collection, monitoring, and control sequencing would be irrelevant to the space station (p. 11). The report also omits "file management" from all mission categories, yet this is a central concern today, whether one is operating a rover on Mars or assisting a crew on the Station. Networking, physically and consequently socially, has radically changed how applications interoperate, how people interact with computers and each other, and how difficult problems are solved (e.g., scientist networks; Gewin, 2008). In particular, files (including programs, logs, photographs, and spreadsheets) are more important than streamed telemetry for transmitting data among instruments, applications, and people.

Second, subcommunities of "knowledge system researchers" have always had different motivations and methods, so we must speak from own experience and knowledge and be wary of generalizing about the field. For example, in the 1980s, Intelligent Tutoring System research emphasized cognitive modeling; today that is out of favor and researchers emphasize interactivity (e.g., emotional tutors; tutors for groups).

Third, although some AI researchers have experienced a paradigm shift in their understanding of knowledge engineering

[3] The description of the oil spill scenario in Hayes-Roth et al. (1983) is replete with examples of interactive methods, involving different roles and reporting relationships. Although several knowledge engineering teams addressed the coordination problem, all focused on tracing the source of the oil spill, fitting the dominant view at the time that diagnostic modeling is an exemplary aspect of expertise.

[4] The distinction between a task (a function) and an activity (a behavior) is crucial for understanding the distinction between a conventional business process model and a work practice model. Tasks and activities are different units of analysis, different perspectives for modeling human behavior. For a full exposition, see Clancey (2002).

[5] By "routine" tasks, we mean tasks that are repeated in particular contexts (e.g., uplinking a file to the Space Station) and for which the vast majority of situations that will occur (e.g., file uplink requests) can be handled automatically by a program that applies a general world model and rules or procedures (perhaps heuristic).

[6] A reviewer commented: "The fact that people naturally behave in disastrously stupid ways is not very good support for the claim that we ought to build automated tools that support the ways people actually work." By supporting the way people actually work, we do not mean reinforcing bad practices, such as failures to coordinate with others, check data validity, document decisions, and so forth. We mean that tools need to fit and respect how the work gets done in terms of interactive flows of data, information, and work products, and even the movements of people in the workplace. Understanding workflows includes identifying the source of data and how it is transformed, cooperative interactions with coworkers in different media (phone, e-mail, meetings), how job order priorities and schedules are set and revised, and so forth. The case study of the Orbital Communications Adapter Mirroring System (OCAMS) in this article illustrates how a workflow tool is designed to *change current practices*, and how we ensure through observation, partnership, and computer simulation that the tool will not disturb aspects of practice we wish to retain or facilitate.

https://doi.org/10.1017/S0890060409990059 Published online by Cambridge University Press

(e.g., Hendler, 2009), many if not most AI academics continue to focus more narrowly on knowledge representation, reasoning, search, uncertainty, and so forth, that is, inventing formalisms for well-defined "problems" rather than building practical systems (for which PSMs were intended).

In short, a given story about "what happened to PSMs" might seem to some people as inaccurate, idiosyncratic, or irrelevant (and probably incomprehensible). If this is your reaction, and you are interested in building practical systems, you are encouraged to consult books that explain in detail the situated cognition perspective about work, problem solving, and tool design (e.g., Greenbaum & Kyng, 1991; Hutchins, 1995; Clancey, 1997*a*, 2006, 2008; Wallace et al., 2007; Robbins & Aydede, 2008).

1.4. Content organization

This paper provides a concrete example of the situated cognition perspective by explaining the architecture and methodology we have applied at NASA in building a practical system for Mission Control. Our work is based on a tool for modeling *work practice*, called Brahms (Clancey et al., 1998; Sierhuis, 2001; Sierhuis et al., 2003, 2007; Clancey, Sierhuis, Damer, et al., 2005; Seah et al., 2005). Brahms provides a language (see Appendix A) for representing work in terms of agents and activities, constituting an analytic shift from building *expert systems* to modeling and designing *work systems*, and a methodological shift from interviewing experts to observing how groups identify problematic situations and negotiate practical methods for handling them. Using Brahms, our concepts for analyzing, modeling, and automating NASA operations include Agent (or Actor), Activity, Problematic Situation, Work System, and Workflow. Our work as "knowledge engineers" has been transformed by three related shifts in perspective: theoretical (how tools are used in practice), architectural (how tools are integrated in real-time, interactive systems), and methodological (how tools are designed and developed).

We begin our story by providing an example of how Brahms has been used to design and implement a workflow tool called OCAMS for communications between ground support in NASA's Mission Control Center (MCC) in Houston and the astronaut crew of the International Space Station (ISS; Clancey et al., 2008). We analyze the use of abstraction in this workflow tool, thus explicating some of the principles in the Brahms architecture, and relate the design of the tool to 2nd GES terminology.

We conclude that for us the motivations of 2nd GES and PSM abstraction in particular have been transformed in our projects from *configuring a problem solver* to a higher level problem of *designing agents in a work system.* From our perspective in developing tools for spaceflight operations, the reusable components in work systems design include entire "problem solvers" (e.g., a planning subsystem), interoperability frameworks (relating hardware and software on different platforms), and interactive systems ("workflow agents") that use and revise models dynamically in a network of people and computer systems. Consequently, the tasks, problems that arise, methods are much broader than how they were conceived in formalizing PSMs originally.

We use the OCAMS example to illustrate our present understanding of human problem solving, examining the challenges for developing automation that eliminates a flight controller position. Our inability to fully automate the work suggests that, although a large part of the original job can be formalized in PSMs, the remaining aspects of human capabilities lie outside of what can be achieved using the present-day model-based reasoning approach.[7] On reflection, we find that what we view as "problems" today are very often aspects of the work that cannot be easily automated, because they lie outside what a single person or expertise in a single domain can handle. Problem solving in spaceflight operations occurs at the group level (often international), across domains of expertise, with formal lines of authority and crosschecking of work. This elevates our notion of "PSMs" to the means of communicating and transforming work products among software agents and human actors, broadening from methods that only take place in the head (mental operations) to include methods for coordinating and cooperating that take place in the world (physical operations). With this shift from designing an expert system to designing a work system, our toolkit ontology moves up a level to include groups, agents, activities, locations, objects, communications, time lines, and so forth. At the lowest levels one finds model-based inference, where aspects of the original PSMs can be found.

2. EXAMPLE: THE OCAMS WORKFLOW AUTOMATION TOOL

We illustrate and develop our perspective on problem solving and tools by analyzing a mission operations workflow tool called OCAMS, developed at NASA using the Brahms modeling and simulation tool (see Appendix A). We describe the context and design constraints, the methodology, the use of abstraction in the solution, and the practical implications for the "library of methods" approach.

2.1. Objectives, requirements, and methodology

The objective of the OCAMS project is to automate to the greatest practical extent the file management operations between ground support groups and the astronauts onboard the ISS. A flight controller called the OCA Officer performs this work.[8] The broader organizational objective is to improve efficiency of mission operations by reducing personnel costs for supporting the Station by 30% from the year 2005 baseline. A secondary objective is to bring NASA's research results into practical application by establishing partnerships

[7] By this same analysis, model-based robots assisting astronauts may *cooperate* with people, but are incapable of *collaborating* on projects (Clancey, 2004).

[8] The Orbital Communications Adapter (OCA) is a PC card that enables a personal computer to FTP files using a satellite network.

https://doi.org/10.1017/S0890060409990059 Published online by Cambridge University Press

between research and operations organizations. Third, from the researchers' perspective, demonstrating practical applications of agent-based systems integration in ground flight operations will promote the use of such tools in lunar surface operations (which are being prototyped in the field experiments in "Mobile Agents," a system using the same architecture; Clancey, Sierhuis, Alena, et al., 2005).

The design requirements for the OCAMS project include the following:

1. automate routine operations of the OCA Officer: mirroring,[9] archiving, uplink/downlink to the ISS, notification[10];
2. enable flight controllers to retain responsibility and authority by allowing for manual overrides of all system operations;
3. enable flight controllers to make routine system modifications without programming (a practical issue of technology sustainability);
4. be sensitive to the current practices for workflow (e.g., how new jobs are received and results transmitted), timing, communications, and authority for variances in routines;
5. respect the work practices of shift handovers that involve restarting software tools, recording file management statistics in a handover log, retaining records of incomplete work, or unresolved problems, and so forth; and
6. respect local software and network practices (e.g., local contractors install software patches; networked computers use firewall protocols).

We used the "simulation to implementation" methodology to design, test, and deploy the OCAMS system (Fig. 1). Collaborating with operations personnel, we first studied how the OCA work was accomplished and what opportunities for automation were most salient and/or pressing. This led to focusing on the mirroring activity. We then collaboratively created two simulations: *current operations* (in which mirroring is done manually) and *future operations* (in which mirroring is done with a distributed multiagent workflow tool). The future operations simulation embeds the prototype OCAMS tool used by a simulated OCA Officer. The simulation also includes a prototype graphical user interface (GUI) by which an OCA Officer can control the simulation to understand what is happening. Both the current and future operations simulations model work shifts, handovers between OCA Officers, and maintaining handover logs. Both simulations were run using 1 month of previously recorded data about file management, allowing quantitative comparisons of the OCA Officer's work with and without the tool (Clancey et al., 2008; Sierhuis et al., 2009).

After validating the completeness and plausibility of the future simulation with the OCA Officers, we extracted the agents comprising the embedded tool prototype (i.e., OCAMS) from the simulation and reconfigured them as a run-time system that communicates between multiple computers.[11] This overall approach of transforming a current simulation into a future simulation and then a tool is called "simulation to implementation," and represents an important example of how system components can be used for both design and implementation in a given application project.

By comparing metrics derived from the current and future simulations, we can use the simulation-to-implementation methodology to quantify the benefit of automation. Simulating how a tool will be used in future operations, including the environment with which the tool will interact (e.g., OCA Officers and applications, such as FTP and spreadsheets) demonstrates how work systems design puts expertise and the original notion of a "consultation dialog" in context.[12] We will elucidate this shift in perspective further when we discuss the challenges of fully automating the OCA position. However, we will begin with a much simpler and narrower analysis, considering to what extent OCAMS can be described in terms of PSMs. First, we provide an overview of the program's design, and then we consider how the design relates to generic tasks and PSMs.

2.2. Analysis of the file management process

This section examines how OCAMS manages files so that we can better understand the role of PSMs (either actual or potential) in its construction. Table 1 shows a schema by which ISS file management is abstracted into a file type taxonomy and file handling rules. The principles for doing the three file management operations considered here (mirroring, archiving, and notifying) are not based on the file contents (e.g., a text document vs. a JPG image), but the functional relation of the file to the mission (operational plans/procedures and software, private data, and exceptions to these). Files are classified into 31 types identified by acronyms assigned by OCA Officers (e.g., NFH is "news from home," such as online

[9] Mirroring involves reconstructing on a local area network (Mirror LAN), a subset of the file operations performed on the ISS file system.

[10] Notification includes speaking on the "voice loop" (a programmable network of intercoms), modifying a flight note (a message posted in a workflow tool), sending e-mail, calling on a telephone, and broadcasting a remark out loud in the room).

[11] More specifically, the definition of certain agents were copied from the Brahms future simulation and pasted into a new Brahms system that runs in real time, with different agents configured to run on different computers. Effectively, each computer has a different Brahms system with its own Brahms virtual machine (engine), and the agents communicate within and between machines over the local network.

[12] The case study of the "emergency management of inland oil and hazard chemical spills" problem (Hayes-Roth et al., 1983) reveals awareness of broad design issues (e.g., the stakeholders will evaluate the program differently, pp. 273–274). However, the relation to the "users" work is undeveloped: "the spills program is intended to augment experts, not to replace them" (p. 273), and yet it must perform "as an expert" (p. 275) in providing assistance to a "possibly inexperienced, off-hours security officer" (p. 334). Brahms provides an ideal framework for simulating the physical flows of spills and how they are managed by an organization, including agents with different capabilities.

https://doi.org/10.1017/S0890060409990059 Published online by Cambridge University Press

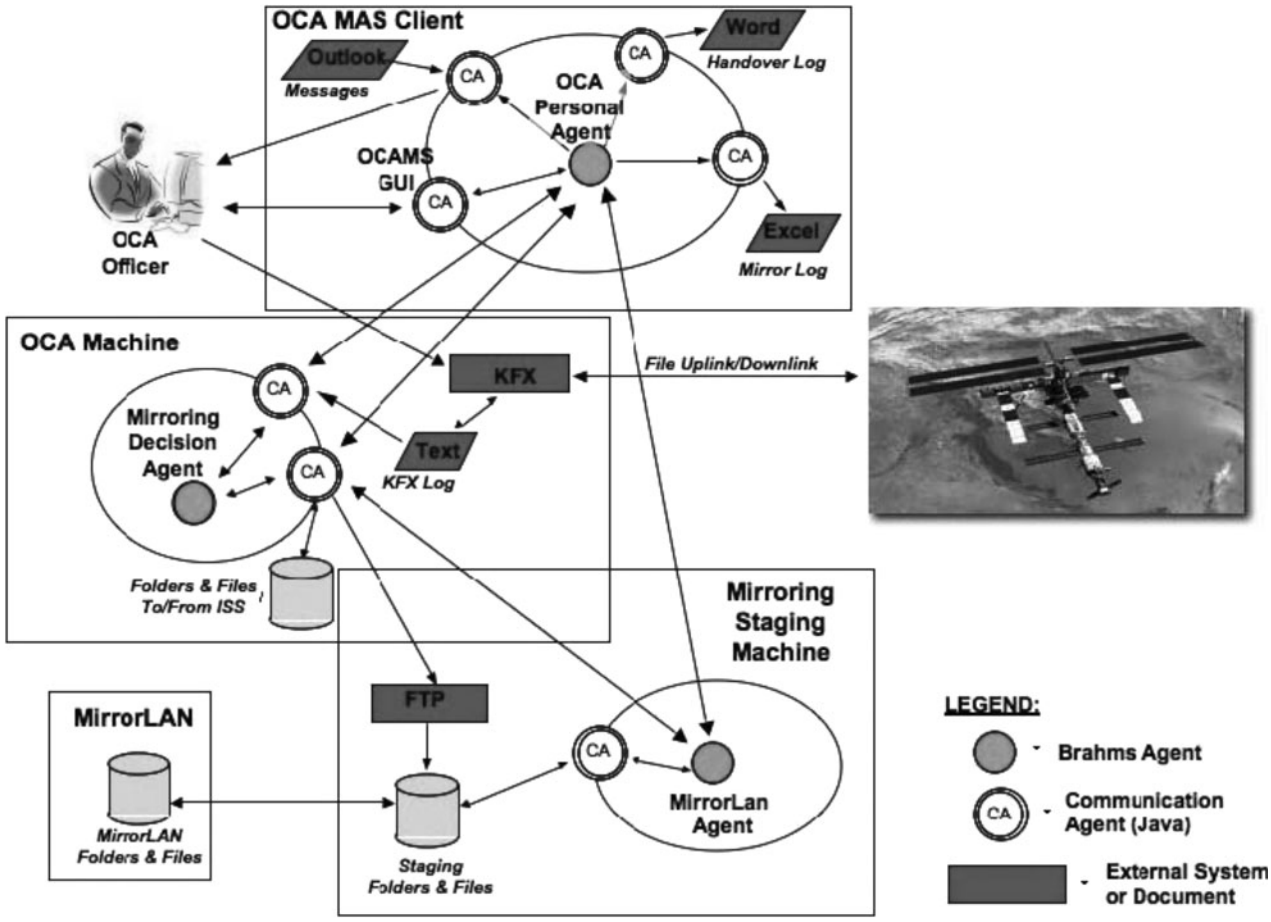

Fig. 1. The Orbital Communications Adapter Mirroring System (OCAMS) using model-based systems integration to automate some of the file management between ground support and the International Space Station (ISS). Brahms agents run on three computers, each with a Brahms Virtual Machine. Agents communicate via messages, providing requests and status. The communication agent commands and exchanges data with external applications and devices via an application-programming interface.

news articles). Files may be transferred "up" to the ISS (uplink), down to earth (downlink), or both.[13]

After collaboratively assembling and reflecting on the file taxonomy, the OCA Officers summarized the classification by the following four principles:

1. Operational data is mirrored and archived. Medical or personal data is neither mirrored nor archived.
2. Most downlinked items are not mirrored. Exceptions: files that stay onboard after downlink; changes the crew has made to the onboard organizational software that must also be implemented on the Mirror LAN.
3. Exception to archiving: keep a rolling archive of imagery because of the volume.
4. Items deleted onboard are also deleted on the Mirror LAN.

[13] The input to OCAMS is a log of operations carried out by the OCA Officer in transferring files between the ground and ISS. OCAMS infers the file type from the file path and name (e.g., a file name of the form "DOUG/flights . . . pkg" is file type DOUG, which is a file type that should be mirrored).

Clancey (1992, pp. 15–16) claimed that the relations required in a domain model depend on the modeling purposes (e.g., diagnosis, teaching, knowledge acquisition). We find in OCAMS this same process-specific, conditional character: for each interactive function added (mirroring, archiving, notifying) the classification becomes more branched and specific.

In particular, the exceptions and variations in notification cause file handling to be highly dependent on the type of file and customer. The simple summary might suggest that the 60 possible combinations of {Transfer × Mirror × Archive × Notify} outcomes would only require four file-handling categories to cover the 31 file types. However, notification must take into account how the file was delivered and whether the customer is on the voice loop system. We find some principles (e.g., modify the flight note if any). However, when we add further exceptions (e.g., is this a shuttle flight or a "stage" in the ISS expedition?), we end up with seven categories to cover exceptions to the four principles. As expected, the OCA Officers' manual provides one procedure per file type, rather than a set of principles (e.g., the manual

https://doi.org/10.1017/S0890060409990059 Published online by Cambridge University Press

Table 1. *Schema for OCAMS taxonomy of file types and handling procedures*

File Types	Type of Data	Transfer Direction	Mirror?	Archive?	Notify?
<Symbolic Name>	{Operational Plans & Procedures | Operational Software & Schedule Changes | Personal or Medical | Exceptions}	{UP | DOWN | BOTH}	{YES | NO}	{YES | NO}	{FlightNote | VoiceLoop | Email | Phone | Outloud}

Thirty-one file types are categorized by the function of the data and/or customer providing or using the data. Transfers are to (UP) or from (DOWN) the International Space Station (ISS). A mirrored file is copied to a ground-based subset of the ISS file system. An archived file is saved in a dated folder indicating its source. Ground support customers are notified in different ways, including a workflow "flight note" system, a speaker–headset intercom ("voice loop"), email, telephone, or by speaking out loud to someone across the room at another workstation.

devotes a section to handling the NFH file type instead of a procedure for "what to do with an up-linked Web page").

This analysis of OCAMS' file types is relevant to understanding the degree to which file management operations are general, and hence, likely to be automatically adapted to future, unanticipated situations. Following the four principles listed above, we could write rules to infer whether a new file type provided by a customer is mirrored, archived, and how the customer is notified. Yet the nature of exceptions indicates a person would need to approve any rule changes. Consequently, we developed an editor by which an OCA Officer can add new file types to the system and modify how file types are handled by filling out a simple form like Table 1. This sustaining cost becomes part of the analysis when justifying development costs and choosing areas for future automation. Put another way, understanding why and how frequently models used by PSMs will be inadequate, thus requiring human intervention, becomes part of the tool-building process.

3. TOWARD A WORKFLOW TOOL LIBRARY: COMPONENTS REUSED IN OCAMS

Can we relate the design of OCAMS to the generic tasks and PSMs of 2nd GES? A first step is to ask what components are reused in Brahms models and workflow tools. To begin, from the perspective of engineers in Mission Control, we have solved a *systems integration problem* by developing OCAMS. Accordingly, in thinking about building another system "like OCAMS" we would consider the components that facilitate interoperability among networks, machines, and applications. In terms of the Brahms language (see Appendix A), these reusable systems integration components include the following:

1. *Agents that communicate with external systems* (*Comm Agents*); these inherit behaviors from a Brahms group (AbstractCommunicationAgent) that handles memory management and supports both simulation and real-time modes.
2. *An FTP client library* derived from previous Mobile Agents configurations.[14]
3. Components in the Brahms "base library":
   a. *basic file operations* (copy, delete, checksum verification, etc.) represented as a Brahms "Input/Output Group" with file manipulation functions written in Java, which other Brahms agents can inherit
   b. *a Brahms "Communicator Group"* with activities to create and read Communicative Acts (inspired by Searle's 1969 speech act theory and based on the FIPA standard agent communication language for multiagent systems)
   c. *a Brahms "JavaUtility Group"* with activities to manipulate Java objects, read Java object values, and manage properties.

In modeling operations, we generally use a table-driven method to represent work schedules, that is, the relation of roles, activities, and time. When an OCAMS simulation begins, a spreadsheet is interpreted to initialize agent beliefs about file types and handling rules[15] and what activities are done when (i.e., the schedule time line; Seah et al., 2005). In addition to adopting a table-driven architecture for simulating operations of different NASA missions, we have reused Brahms workframes and thoughtframes that relate to schedules.

After building a variety of mission operations simulations, it seems clear that the simulating and automating workflow operations requires agents to maintain beliefs about the work in process, represented as sets of objects (e.g., the files being uplinked and downlinked), constituting a central part of the agent's "situation-specific model" of the work system. We expect to reuse such constructs in other applications.

In summary, the Brahms language enables formalizing and reusing model constructs exemplifying the 2nd GES concept of representational tools with components more specific than "inference rule" and "schema." In OCAMS, these components include descriptions of groups (including inheritable beliefs and activities), particular agents, and schedule-related inference, and situation-action rules (called *Workframes*).

[14] One of the first examples of an "intelligent agent" was a program that managed files using FTP (Anderson & Gillogly, 1976). A favorite joke was that if the agent were told to move a directory in the most efficient manner possible, it might first delete all the files.

[15] An example thoughtframe initialized from a spreadsheet: "If a file with the extension'.cfg' was uplinked to the folder 'doug/config,' then its file type is DOUG_Config_File_Type and it should be mirrored."

https://doi.org/10.1017/S0890060409990059 Published online by Cambridge University Press

These constructs are useful for automating workflow that involves systems integration. However, 2nd GES research especially focused on abstraction of methods for manipulating models, as in medical diagnosis. What are the tasks in OCAMS? What models are contained in OCAMS?

4. APPLYING THE SYSTEM–TASK–OPERATOR FRAMEWORK TO OCAMS: FROM EXPERT SYSTEMS TO WORKFLOW SYSTEMS

Representing a file taxonomy and file handling operations as agents (mirroring, monitoring, archiving) derives directly from the 2nd GES approach of developing a domain model and functional (task-specific) operators. Here is how OCAMS fits the System–Task–Operator framework (Clancey & Barbanson, 1991; Clancey, 1992):

- The system being modeled is the ISS file system, including workstations, directory structure, and types of files. These file types are related to types of customers (e.g., physicians, mission planners) and two broad functions in which the files play a part (operational and medical/personal).
- With respect to the ISS file system, *the task is configuration:* assembling/maintaining another file system with certain properties (mainly mirroring the ISS file system minus medical/personal files and images). In other words, the configuration task here is to replicate a subset of a given structure (the ISS file system) on a "mirror" server, in which the structure of the secondary system (the Mirror LAN) is subject to certain general constraints (namely, what file types are mirrored), which constitute "configuration rules."
- *The PSM is simple classification*: the file name defines the file type, and this defines how the file is to be configured in the Mirror LAN (the options are Copy, Unzip and monitor for errors, Delete, and Do nothing).

An obvious reaction to presenting OCAMS as an example for appraising the value of identifying PSMs is that the mirroring and archiving operations are based on definitions, so the most trivial method, simple classification, suffices. Heuristic decision making will be required subsequently for executing file transfer operations (Uplink/Downlink) within the variable time periods available for communicating with the ISS. Methods previously developed for scheduling problems are likely to be useful.[16]

However, just considering mirroring and archiving, the file management task actually being accomplished by OCAMS is already more complex than it might first appear:

1. OCAMS is actually *building a physical system* (the Mirror LAN), not just modeling the design of a file system. By the System–Task–Operator framework, the modeling purpose (task) with respect to the system in the world being reasoned about (the file system) is "Build."
2. OCAMS is applying the general model (file type taxonomy and handling rules) to plan and execute file management actions (copying, moving, renaming, deleting, opening, etc.) that coordinate three situation-specific system models (SSMs):
   a. SSM-CONFIG represents the *desired configuration:* files that need to be mirrored, archived, and monitored at this time (OCAMS' plan).
   b. SSM-MIRROR and SSM-ISS represent the *current state of the world*, which are the respective state of the Mirror LAN and the state of the ISS file system (including status of OCAMS actions that are pending confirmation).

Besides using simple classification for creating the SSM-CONFIG from the SSM-ISS, OCAMS uses the methods of queuing, handshake protocol, retry iteration, and synchronization to maintain the Mirror LAN system (modeled by SSM-MIRROR).

In other words, OCAMS is not just a reasoning system. OCAMS is an interactive system, an actor in the world,[17] which uses model-based methods to plan its actions (SSM-CONFIG) and keep track of the work to be done (SSM-MIRROR). In some respects, it is as if Mycin were charged not just with interpreting culture results, but actually treating a patient. More prosaically, this is the difference between an expert system and a workflow system.

Consequently, the software engineering problems in designing OCAMS are complex and incorporate the problem solving framework only as a means for a larger problem. In particular, OCAMS agents run as distributed processes on six or more workstations without a central controlling system. These agents coordinate the physical manipulation of materials (files) in the world, the layout of displays and logs, files in queues, and of course file directories. The agents also run other programs and interpret error messages. Model-based inference is used especially for planning and tracking actions (e.g., deciding how to handle a particular file and what to do when errors occur). In this respect, we can view OCAMS as being like a robot that inhabits the network in a distributed fashion.

As is plainly visible in the Building Expert Systems case study, the consultation paradigm was the dominant way of thinking about model-based tools in 1980. We imagined a single "user" who is interviewed by the program; the program

[16] One can also imagine a more complex program with "diagnosis and repair" capabilities. Application and networking errors can occur that put the file system (and agent system) into an uncertain state. Instead, failure handling is automated in OCAMS by an "administration agent" that simply restarts the agent processes and redoes the operations in the current "batch" of files. When more is required, a person usually needs to do something that is out of the scope of automation (e.g., deciding how to diplomatically handle an ISS crew member's overgrown mail file).

[17] For an analysis of "nonhuman actants," in particular, how technology and people codetermine work practices, see Latour (1991).

https://doi.org/10.1017/S0890060409990059 Published online by Cambridge University Press

asks questions, makes inferences, and then prints an interpretation and recommendations (e.g., a plan) for human interventions. Of course, many other different types of systems were developed in the 1980s, with automated process control programs being most similar to OCAMS.[18] Opportunities and challenges of embedding model-based programs in real-time systems changed substantially with the advent of distributed computing, the Internet, security concerns, multimodal interfaces, multiple vendor platforms, and so forth. As illustrated by the list of reusable components in OCAMS (see "Toward a Workflow Library"), a library of PSMs is just one part of what is required in a practical toolkit for building model-based systems today. To further understand the requirements, we will consider requirements for extending OCAMS' functionality to automate more of the OCA Officers' work.

5. A BROADER VIEW: THE MISSION OPERATIONS WORK SYSTEM

Viewed comprehensively, automating the entire role of the OCA Officer would involve the following additional tasks: sustaining communications between computers on different networks owned by different organizations, preserving security of mission systems and private data using secure communication, customizing file management for special requests; verifying and notifying that customer requests are complete, and keeping records in handover logs and ongoing mission documentation. As we described in OCAMS, some aspects of these responsibilities can be accomplished by PSMs. More broadly, the problem at hand is to build this entire system: developing a system of agents that together automate an operations role that involves a great deal of coordination with people.

As we broaden OCAMS' original mirroring and archiving functions to cover most of the work done by the OCA Officer, our perspective of "the system" being reasoned about and manipulated changes yet again, and the focus on maintaining proper interactions with other players (people and tools) becomes more central. For example, OCAMS will need to interpret and modify flight notes, messages in a kind of discussion forum used by flight controllers. When we include the customers who are delivering files for uplink and receiving downlinked files, we see that the work system involves people performing other roles in the Mission Operations Directorate (MOD),[19] astronaut family members, and flight controllers in other countries in support of three to thirteen astronauts (assuming a full shuttle flight of seven astronauts will occur with a full contingent of six onboard the ISS). This is a *work system*, a distributed collaboration among people using diverse representations and tools: physicians, aeronautics engineers, planners, robotics engineers, power and propulsion flight controllers, family members, and so forth.

From this perspective, an extended OCAMS that automates all of the work of the OCA Officer must be designed as an actor (participating agent) in a work system. This agent would be responsible for retrieving files from different locations (file servers, hard drives), interpreting documents, controlling different subsystems (e.g., software programs such as FTP), creating structured documents (logs), and communicating with people (via a GUI, e-mail, flight notes, and perhaps someday by speaking on the voice loop). The comprehensive process would require classifying the work beyond file types to include roles and authority structures in operations, daily schedules, and mission phases (e.g., protocols and priorities change when the Space Shuttle is docked to the ISS).

File management correspondingly becomes a subproblem within the larger task of configuring the more-encompassing work system, such as prioritizing file transfers for different customers, given limited bandwidth, and fragmented communication windows between the ground and ISS. In effect, the responsibility of OCAMS shifts from merely focusing on and juggling files to focusing as well on what people are trying to accomplish and the support required by interacting teams (e.g., a team of physicians on the ground and the astronauts on the ISS). For example, the OCA Officers' Handbook states:

> OCA operators should keep their OPSPLAN well informed of file transfers and [satellite network] availability. . . . The OPSPLAN should always know what is on board and what still needs to be uplinked. The OPSPLAN will coordinate crew email operations with CAPCOM and the crew. Prior to uplink, the OPSPLAN and OCA operator should agree on the priority/uplink order of the files to be transferred. This is because many times [network] availability is sparse and time to uplink can be pressed.

In summary, extending OCAMS' responsibilities requires considerably broadening what aspects of spaceflight are modeled (e.g., network availability), as well as adding agents to prioritize and coordinate what will then become "low level" file operations (e.g., mirroring). In particular, more sophisticated planning and perhaps "what if" analysis would be useful. To this end, we would probably couple OCAMS to a constraint-based tool, such as SPIFe (McCurdy et al., 2006), rather than represent a planning capability in Brahms.[20] This example suggests, like the functional-location decomposition of OCAMS into agents, that the preferred software engineering approach today is not construction of single

[18] For example, Fagan's 1980 dissertation project, the Ventilator Management system (Rutledge et al., 1992), was coupled to real-time data, obviating manual entry, and thus enabled ongoing interpretation and alerts.

[19] MOD is the organization within the NASA Johnson Space Center that operates the MCC, usually associated with a room with three large monitors called the Flight Control Center. The OCA Officers work within MCC (a secure building), but in another room, one of several "backrooms" where people support the flight controllers (such as OPSPLAN, the "operations planner") in the Flight Control Center, whom they can hear and speak to via the voice loop. Fortunately, simple PSMs are sufficient for understanding and using acronyms.

[20] Indeed, some of the files managed by OCAMS are maintained by a model-based scheduler (OSTPV; Frank et al., 2008). Thus, in effect, by following simple rules OCAMS is already helping coordinate work products between people and other automated systems.

https://doi.org/10.1017/S0890060409990059 Published online by Cambridge University Press

programs from components, but integration of modules, often running on different platforms, that can flexibly communicate in real-time settings.[21]

Thus, we again see that reusability of components is important; in particular, components that manipulate models (e.g., a scheduler) are direct descendents of PSMs formalized in the 1980s. However, the systems engineering problem now includes integration and interoperability of *tools* or *services*, not just copying a formalism used in one program into another. The Brahms run-time environment is specifically designed to handle this systems-integration problem, also called "interoperability." This integrative, run-time capability was not part of Brahms' initial design as a work practice simulation framework. However, the possibility of integrating actual tools within a work practice simulation was realized early on (e.g., to simulate how people used an expert system). Well before developing the Mobile Agents system in 2001, we also realized and documented the advantages of the Brahms language and engine for implementing distributed, communicating processes containing independent models of the world engaging in different, but coordinated activities, namely, "personal agents" (e.g., Anderson, 1977; see also US Patent No. 6,216,098, Clancey et al., 2001).

To recap, the simulation-to-implementation methodology using Brahms is particularly suitable to the challenges of automating a flight controller position. The shift from focusing on tasks to activities and thus work practices (how the work is actually done, vs. idealized procedures detailing functional transformations of work products) corresponds to a shift from replicating or facilitating reasoning to supporting *work systems*. This shift from a product-centered model to a behavioral-practice model involves a shift to modeling and simulating transactions (customer–flight-controller–product relations). Accordingly, we have become concerned with the dynamic behaviors of an agent in a work system involving *maintenance of transactions*, such as negotiation of requirements and resources, which must be tracked, confirmed, communicated, crosschecked, and sometimes renegotiated.

Going further to automate nonroutine aspects of the OCA Officers' work involves modeling how problems arise and are recognized, formulated, and resolved. In general, participating as an actor in spaceflight operations requires nuanced, context-sensitive conceptualization of what ground support and the astronauts (both individually and as teams) are trying to do, what incoming data and events mean, what different players are doing now on the mission, what resources (tools, timing, personnel, supplies) have available, how they are progressing in their endeavor, and so forth (Feltovich et al., 2007).

Relating Brahms and OCAMS to the conception of PSMs, we can say the glass is half-empty; our view of human problem solving is much broader and the capabilities of a model-manipulation process appear relatively limited compared to what groups of people accomplish. Alternatively, we can say that the glass is half-full; OCAMS can be abstractly described using the System–Task–Operator framework. However (half-empty again), this is only an analytical perspective: the salient architecture is characterized in terms of agents interacting with each other and systems in the world, and it is only within this more complex system that we find the situation-specific models and inference operators of "problem solving." Perhaps as the analysis in this section implies, OCAMS agents will eventually model and act within the broader spaceflight operations work systems, incorporating familiar PSMs at a different level of abstraction. However, the possibility and practicality of doing this remains to be seen.

6. CONCLUSIONS AND PERSPECTIVE

In applying 2nd GES methods, we have developed Brahms, a tool for modeling work practice. By enabling designers to model and simulate alternative work system designs in different scenarios, a Brahms simulation model has familiar science and engineering purposes. It enables better understanding of causal processes (e.g., relations between roles, schedules, procedures, and workplace automation), measuring work systems flows (e.g., productivity), identifying bottlenecks, predicting deadlocks and gaps, and evaluating hypothesized improvements.

The Brahms modeling framework constitutes a schema for simulating work practice, very much in the spirit of the 2nd GES effort to develop domain-general abstractions, but using an ontology relevant to modeling agent behaviors: observing, moving, communicating, manipulating objects in the real world. In particular, problem solving is modeled as an activity because it occurs in human behavior, often involving tools (e.g., getting procedures from manuals) and interacting with other agents. Modeling facilities, organizational roles, communication methods, and actual movements is important for understanding how an automated system will fit within or require changes to people's activities. For example, flight controllers in the same room as the OCA Officers speak over partitions to ask questions about the status of the work to be done; they also walk over to examine files on the Mirror LAN server. Should OCAMS broadcast information on a loudspeaker? How would it know that the person being addressed is not too busy to hear?

Just as everyday work involves much more than reasoning, a library of PSMs is not sufficient for building workflow tools. Nevertheless, the Brahms modeling framework is derived from the architecture of expert systems, where an agent is like a knowledge-based system, but it is located and operating within a broader *work system schema* (group, agent, activity, detectable, communication act, area, movement). The Brahms engine, replicated on each computer platform containing Brahms agents, coordinates real-time, parallel interactions among the agents and other systems in their environment.

[21] For example, in problem solving research the notion of "memory" originally focused on efficient matching. For an agent in an operational environment, the practical issue broadens to storing facts to engage in discourse about events that occurred days or months ago. In MOD, a flight controller might ask his/her personal agent, "Have we uplinked files like this to crew members on previous flights?"

https://doi.org/10.1017/S0890060409990059 Published online by Cambridge University Press

In using the simulation-to-implementation methodology we have found that a work practice simulation has a range of purposes over time:

- formalizing a particular aspect of practice to produce metrics useful for improving how the work is done,
- modeling and simulating agents that automate aspects of the work,
- simulating how a workflow tool would be used in practice (again with metrics) and deploying an agent-based workflow tool on distributed platforms, and
- improving the Brahms language and engine for modeling and relating people and technology in work systems.

The vision behind 2nd GES research, that abstraction of system components could make building future systems easier, has certainly been central in our methodology. The abstractions that have guided the development of OCAMS combine concepts and methods from software engineering and from our own multiagent systems: a layered architecture; functional decomposition of services into agents; providing a "personal agent" for interacting with the person using the tool; distributed implementation capability; abstraction of domain relations into a separate domain model, enabling, for example, a table-driven process; general methods for systems integration (using JAVA to write a "communication agent" that mediates between an application-programming interface and other Brahms agents); handshake communication protocols for tracking the status of subsystems; and categorizing agent messages (e.g., request, information, subscription, proposal) in the Brahms Communication Library.

We conclude that the 2nd GES effort was a reasonable, well-grounded engineering phase of research that aimed to analyze expert systems, abstract methods, and potentially make system building more efficient through tools with libraries of PSMs. We believe that such libraries never became widely used in software engineering for multiple reasons:

1. An important challenge in developing a new workplace tool is proper integration with a complex, distributed work system involving people and other tools.
2. The most obvious automation opportunities can be handled algorithmically, but people need to be responsible for interpreting and approving variations to the routine (e.g., handling an astronaut's request to uplink a new kind of file on a daily basis for 6 months). Such variations require value-based judgments involving diplomacy and negotiation with other stakeholders.
3. When components are "reused" they are usually large programs (e.g., a planner) or hardware (e.g., camera) integrated into a larger workflow system, and such integration is specialized because it involves representational mapping between ontologies (e.g., integrating a camera with email and a database; Clancey, Sierhuis, Alena, et al., 2005).

More broadly, referring to situated cognition research, we emphasize that the *methods* of work practice are interactive, employing reasoning for and through action in the real world. Interactive methods relate internal system models to actions in the world in a manner that carries out the agent's responsibilities in a sustained way over time, including direct observation, communicating with people and other tools, coping with failure (retrying; reconciling models and reality), and detecting when assistance is required. The twist is that a problem solver must not just model the world to reason about it, but actually uses such models to keep the world in order on an ongoing basis.

In conclusion, the analytic thrust of 2nd GES research was appropriate and still makes sense; however, the concept of developing practical systems from "skeletal tools" containing PSM primitives was incomplete. Automation systems are not "autonomous." Instead, like people in a workplace, applications that serve to automate entire roles are better conceived as *agents*, which interact frequently with people and other systems, to categorize, negotiate, and communicate their requests and contributions in the work environment. This cooperative endeavor can be viewed as a higher order "modeling problem" of configuration, diagnosis, planning, and so forth.

The problem solving paradigm that strongly influenced expert systems work focused on reasoning about a problem by manipulating one or more models of systems or processes in the world. In viewing problem solving more contextually, our research interest shifts to the meta-level: *problems* arise because of *discrepancies between models* (beliefs, plans, procedures, theories) *and the world*, that is, "tear" in the model (Burton & Brown, 1979, p. 95). PSMs include ways of interacting with people and other systems to reinterpret policies and adapt procedures to new contexts (e.g., changed resources, revised priorities defined by authorities). For example, the OCA Officer may seek supervisory assistance for dealing with a file transfer request that does not fit defined categories; negotiate with peers across disciplines or shifts about who will take responsibility to resolve aberrant situations; and discuss economic and international political perspectives by which policies will be evaluated by other agencies.

These concepts (assistance or permission, responsibility, and nontechnical perspectives) move work practice into the realm characterized by Simon (1973) as "ill-structured problems." Complex events can call into question the validity of models and policies. Such discrepancies are handled by people by creating new categories, giving new interpretive twists to rules and procedures by blending otherwise conflicting values, and other methods of deferring, reassigning, or even defining away the problematic situation.[22] Such adaptation

[22] Here we are reminded of the Columbia disaster, in which a simulation model was interpreted to argue that foam could not damage the Space Shuttle; hence, photographs of possible damage (taken from Earth) would not be necessary (Columbia Accident Investigation Board, 2003). This decision was influenced by a confusing presentation of the model in PowerPoint (Tufte, 2006), the organizational status of the presenters, and the possible delays imposed on future missions by categorizing the event as "out of family" (i.e., as being a discrepancy).

can be difficult when different analytic perspectives (e.g., scientific, ethical, economic, political world views) are at cross purposes (Schön, 1987) or because of organizational affiliation participation (hence knowledge) is discounted. Here, in saying that cognition is situated we mean that spaceflight operations expertise is inherently distributed, not all technical, and dynamically constructed in an ongoing social process. The reflective problem solver asks: are my models adequate? Have I interpreted them appropriately? Do I need to work harder to prove my proposed actions are valid? As software engineers move into this realm, with programs becoming actors in the workplace, we are challenged to be aware of the difference between a model-based mechanism and human conceptualization (Clancey, 1997*a*, 1999; Wallace & Ross, 2006). One approach is to design workflow agents to *facilitate human responsibility* when automating routine tasks by helping people to understand the limitations of the models and to detect when they are wrong.

ACKNOWLEDGMENTS

OCAMS was developed in partnership with the OCA Officers in Mission Operations Directorate of NASA Johnson Space Center, particularly Chris Buckley, Deborah Hood, Skip Moore, Fisher Reynolds, Tyson Tucker, and Karen Wells. We are grateful for the vision and support of Tim Hall and Brian Anderson at NASA Johnson Space Center and Mike Shafto at NASA Ames. Mike Scott and Ron van Hoof (Stinger Ghaffarian Technologies, NASA Ames) played key roles in implementing Brahms and OCAMS. This work was partially supported by funding from NASA's Constellation Program. *AI EDAM* reviewers provided many useful comments.

REFERENCES

Acquisti, A., Sierhuis, M., Clancey, W.J., & Bradshaw, J.M. (2002). Agent-based modeling of collaboration and work practices onboard the International Space Station. *Proc. 11th Computer-Generated Forces and Behavior Representation Conf.*, pp. 181–188.

Anderson, R.H. (1977, June). The use of production systems in RITA to construct personal computer "agents." *SIGART Newsletter 63*, 23–28.

Anderson, R.H., & Gillogly, J.J. (1976). *Rand Intelligent Terminal Agent (RITA): Design Philosophy*. RAND Report R-1809-ARPA. Washington, DC: Rand Corporation.

Bonasso, P., Firby, J.R., Gat, E., Kortenkamp, D., Miller, D., & Slack, M. (1997). Experiences with an architecture for intelligent, reactive agents. *Journal of Experimental Theory of Artificial Intelligence 9*, 237–256.

Bond, A.H., & Gasser, L. (1988). *Readings in Distributed Artificial Intelligence*. San Mateo, CA: Morgan Kaufmann.

Brooks, R.A. (1991). How to build complete creatures rather than isolated cognitive simulators. In *Architectures for Intelligence* (VanLehn, K., Ed.), pp. 225–239. Hillsdale, NJ: Erlbaum.

Burton, R.R., & Brown, J.S. (1979). An investigation of computer coaching for informal learning activities. *International Journal of Man–Machine Studies 11(1)*, 5–24.

Carley, K. (1990). Group stability: a socio-cognitive approach. In *Advances in Group Processes, Advances in Group Processes: Theory and Research* (Lawler, E.J., Markovsky, B., Ridgeway, C., & Walker, H.A., Eds.), Vol. 7, pp. 1–44. Greenwich, CT: JAI Press.

Chandrasekaran, B., & Johnson, T.R. (1993). Generic tasks and task structures: history, critique and new directions. In *Second Generation Expert Systems* (David, J.M., Krivine, J.P., & Simmons, R., Eds.), pp. 239–280. New York: Springer–Verlag.

Choo, T.H., & Skura, J.P. (2004). SciBox: a software library for rapid development of science operation simulation, planning, and command tools. *Johns Hopkins APL Technical Digest 25(2)*, 154–162.

Clancey, W.J. (1984). Methodology for building an intelligent tutoring system. In *Method and Tactics in Cognitive Science* (Kintsch, W., Miller, J.R., & Polson, P.G., Eds.), pp. 51–83. Hillsdale, NJ: Erlbaum.

Clancey, W.J. (1985). Heuristic classification. *Artificial Intelligence 27*, 289–350.

Clancey, W.J. (1986). Qualitative student models. In *Annual Review of Computer Science*, pp. 381–450. Palo Alto: Annual Reviews.

Clancey, W.J. (1989). Viewing knowledge bases as qualitative models. *IEEE Expert: Intelligent Systems and Their Applications 4(2)*, 9–15, 18–23.

Clancey, W.J. (1992). Model construction operators. *Artificial Intelligence 53(1)*, 1–124.

Clancey, W.J. (1997*a*). *Situated Cognition: On Human Knowledge and Computer Representations*. New York: Cambridge University Press.

Clancey, W.J. (1997*b*). The conceptual nature of knowledge, situations, and activity. In *Human and Machine Expertise in Context* (Feltovich, P., Ford, K., & Hoffman, R., Eds.), pp. 247–291. Menlo Park, CA: AAAI Press.

Clancey, W.J. (1999). *Conceptual Coordination: How the Mind Orders Experience in Time*. Hillsdale, NJ: Erlbaum.

Clancey, W.J. (2002). Simulating activities: relating motives, deliberation, and attentive coordination. *Cognitive Systems Research 3(3)*, 471–499.

Clancey, W.J. (2004). Roles for agent assistants in field science: personal projects and collaboration. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews 34(2)*, 125–137.

Clancey, W.J. (2005). Towards on-line services based on a holistic analysis of human activities. In *Towards the Learning GRID: Advances in Human Learning Services* (Ritrovato, P., Allison, C., Cerri, S.A., Dimitrakos, T., Gaeta, M., & Salerno, S., Eds.), pp. 3–11. Frontiers in Artificial Intelligence and Applications, Amsterdam: IOS Press.

Clancey, W.J. (2006). Observation of work practices in natural settings. In *Cambridge Handbook on Expertise and Expert Performance* (Ericsson, A., Charness, N., Feltovich, P., & Hoffman, R., Eds.), pp. 127–145. New York: Cambridge University Press.

Clancey, W.J. (2008). Scientific antecedents of situated cognition. In *Cambridge Handbook of Situated Cognition* (Robbins, P., & Aydede, M., Eds.), pp. 11–34. New York: Cambridge University Press.

Clancey, W.J., & Barbanson, M. (1991). TOPO: implications of the system–model–operator metaphor for knowledge acquisition. *IEEE Expert 6(5)*, 61–65.

Clancey, W.J., & Letsinger, R. (1981). NEOMYCIN: reconfiguring a rule-based expert system for application to teaching. *Proc. 7th IJCAI*, pp. 829–826.

Clancey, W.J., Sachs, P., Sierhuis, M., & van Hoof, R. (1998). Brahms: simulating practice for work systems design. *International Journal of Human–Computer Studies 49*, 831–865.

Clancey, W.J., Sierhuis, M., Alena, R., Berrios, D., Dowding, J., Graham, J.S., Tyree, K.S., Hirsh, R.L., Garry, W.B., Semple, A., Buckingham Shum, S.J., Shadbolt, N., & Rupert, S. (2005). Automating CapCom using mobile agents and robotic assistants. *American Institute of Aeronautics and Astronautics 1st Space Exploration Conf.* NASA Report TP 2007-214554. Accessed at http://ntrs.nasa.gov

Clancey, W.J., Sierhuis, M., Damer, B., & Brodsky, B. (2005). The cognitive modeling of social behavior. In *Cognitive Modeling and Multi-Agent Interaction* (Sun, R., Ed.), pp. 151–184. New York: Cambridge University Press.

Clancey, W.J., Sierhuis, M., Seah, C., Buckley, C., Reynolds, F., Hall, T., & Scott, M. (2008). Multi-agent simulation to implementation: a practical engineering methodology for designing space flight operations. In *Engineering Societies in the Agents' World VIII. Lecture Notes in Artificial Intelligence* (Artikis, A., O'Hare, G., Stathis, K., & Vouros, G., Eds.), Vol. 4995, pp. 108–123. Heidelberg: Springer.

Clancey, W.J., Torok, D.M., Sierhuis, M., Hoof, R.J.J.V., & Sachs, P. (2001). *Simulating work behavior.* US Patent 6,216,098.

Cohen, P.R., Greenberg, M.L., Hart, D.M., & Howe, A.E. (1989). Trial by fire: understanding the design requirements for agents in complex environments. *AI Magazine 10(3)*, 34–48.

Columbia Accident Investigation Board (2003). *CAIB Report*, Vol. 1. NASA. Accessed at http://www.caib.us/news/report/volume1/default.html

David, J.M., Krivine, J.P., & Simmons, R., Eds. (1993). *Second Generation Expert Systems*. New York: Springer–Verlag.

Dourish, P., & Button, G. (1998). On "technomethodology": foundational relationships between ethnomethodology and system design. *Human–Computer Interaction 13*, 395–432.
Feltovich, P.J., Bradshaw, J.M., Clancey, W.J., Johnson, M., & Bunch, L. (2007). Progress appraisal as a challenging element of coordination in human and machine joint activity. *Proc. ESAW 2007*, pp. 124–141.
Frank, J., Morris, P.H., Green, J., & Hall, T. (2008). The challenge of evolving mission operations tools for manned spaceflight. *Proc. 9th iSAIRAS 2008*.
Gewin, V. (2008). The new networking nexus. *Nature 451*, 1024–1025.
Gilbert, N., & Doran, J. (1993). *Simulating Societies: The Computer Simulation of Social Phenomena*. London: UCL Press.
Greenbaum, J., & Kyng, M., Eds. (1991). *Design at Work: Cooperative Design of Computer Systems*. Hillsdale, NJ: Erlbaum.
Hayes-Roth, F., Waterman, D.A., & Lenat, D., Eds. (1983). *Building Expert Systems*. Reading, MA: Addison–Wesley.
Hendler, J. (2009). The Semantic Web from the bottom up. In *Switching Codes: "Ontology, Induction, and Semantic Web"* (Bartscherer, T., & Coover, R., Eds.). Chicago: University of Chicago Press.
Hutchins, E. (1995). *Cognition in the Wild*. Cambridge: MIT Press.
Latour, B. (1991). Technology is society made durable. In *A Sociology of Monsters: Essays on Power, Technology, and Domination* (Law, J., Ed.), pp. 103–131. New York: Routledge.
McCurdy, M., Pyrzak, G., Ratterman, C., & Vera, A. (2006). The design of efficient ground software tools. *Proc. 2nd IEEE Int. Conf. Space Mission Challenges for Information Technology*, p. 257.
McDermott, J. (1988). Preliminary steps toward a taxonomy of problem-solving methods. In *Automating Knowledge Acquisition for Expert Systems* (Marcus, S., Ed.), pp. 225–256. Boston: Kluwer Academic.
NASA (1980). *Machine Intelligence and Robotics: Report of the NASA Study Group*. Office of Aeronautics and Space Technology. Accessed at http://www.ntis.gov/ and http://www.rr.cs.cmu.edu/NASA.pdf
Newell, A., & Simon, H.A. (1972). *Human Problem Solving*. Englewood Cliffs, NJ: Prentice–Hall.
Nii, H.P. (1986*a*). Blackboard systems. *AI Magazine 7(2)*, 38–53.
Nii, H.P. (1986*b*). Blackboard systems. *AI Magazine 7(3)*, 82–106.
Robbins, P., & Aydede, M., Eds. (2008). *The Cambridge Handbook of Situated Cognition*. New York: Cambridge University Press.
Rutledge, G.W., Thomsen, G.E., Farr, B.R., Tovar, M.A., Polaschek, J.X., Beinlich, I.A., Sheiner, L.B., & Fagan, L.M. (1992). *The Design and Implementation of a Ventilator-Management Advisor*. Stanford Knowledge Systems Laboratory. Accessed at ftp://ksl.stanford.edu/pub/KSL_Reports/./KSL-92-11.ps.gz
Schön, D.A. (1987). *Educating the Reflective Practitioner: Toward a New Design for Teaching and Learning in Professions*. San Francisco, CA: Jossey–Bass.
Schreiber, A.T., Wielinga, B.J., de Hoog, R., Akkermans, J.M., & Van de Velde, W. (1994). CommonKADS: a comprehensive methodology for KBS development. *IEEE Expert 9(6)*, 28–37.
Seah, C., Sierhuis, M., & Clancey, W.J. (2005). Multi-agent modeling and simulation approach for design and analysis of MER mission operations. *Proc. Int. Conf. Human–Computer Interface Advances for Modeling and Simulation*, pp. 73–78.
Searle, R. (1969). *Speech Acts: An Essay in Philosophy of Language*. New York: Cambridge University Press.
Shalin, V.L. (2005). The roles of humans and computers in distributed planning for dynamic domains. *Cognition, Technology, and Work 7(3)*, 198–211.
Sierhuis, M. (2001). *Modeling and simulating work practice*. PhD Thesis, University of Amsterdam.
Sierhuis, M., Clancey, W.J., Seah, C., Trimble, J., & Sims, M.H. (2003). Modeling and simulation for mission operations work systems design. *Journal of Management Information Systems 19(4)*, 85–128.
Sierhuis, M., Clancey, W.J., & van Hoof, R. (2007). Brahms: a multiagent modeling environment for simulating work practice in organizations. *International Journal for Simulation and Process Modeling 3(3)*, 134–152.
Sierhuis, M., Clancey, W.J., & van Hoof, R. (2009). Brahms: an agent-oriented language for work practice simulation and multi-agent systems development. In *Multi-Agent Programming: Languages, Tools and Applications* (Bordini, R.H., Dastani, M., Dix, J., & El Fallah Seghrouchni, A., Eds.). New York: Springer.
Simon, H.A. (1973). The structure of ill-structured problems. *Artificial Intelligence 4(3)*, 181–202.
Tufte, E. (2006). *Beautiful Evidence*. Cheshire, CT: Graphics Press.
Vicente, K.J. (1999). *Cognitive Work Analysis: Toward Safe, Productive, and Healthy Computer-Based Work*. Mahwah, NJ: Erlbaum.
Wallace, B., & Ross, A. (2006). *Beyond Human Error: Taxonomies and Safety Science*. Boca Raton, FL: CRC Press.
Wallace, B., Ross, A., Davies, J.B., & Anderson, T., Eds. (2007). *The Mind, the Body and the World: Psychology After Cognitivism*. London: Imprint Academic.
Wynn, E. (1991). Taking practice seriously. In *Design at Work: Cooperative Design of Computer Systems* (Greenbaum, J., & Kyng, M., Eds.), pp. 45–64. Hillsdale, NJ: Erlbaum.

William J. Clancey works at the NASA Ames Research Center and Florida Institute of Human and Machine Cognition. He is Chief Scientist for human-centered computing in the Intelligent Systems Division at Ames. He received a BA in mathematical sciences from Rice University (1974) and a PhD in computer science from Stanford University (1979). Dr. Clancey has extensive experience in medical, educational, and financial software and was a founding member of the Institute for Research on Learning. He is especially interested in relating social science and neuropsychology to descriptive (symbolic) models of cognition to understand the nature of consciousness.

Maarten Sierhuis is a Senior Research Scientist and Lead of the Autonomy and Decision Support Group at Carnegie Mellon University Silicon Valley, which is located at NASA Ames Research Center. This work was performed when he was affiliated with RIACS/USRA, also at NASA Ames. He received a PhD in social science informatics from the University of Amsterdam and holds an engineering degree in informatics from the Polytechnic University in The Hague, The Netherlands. Dr. Sierhuis is a Co-Principal Investigator for the Brahms Project, working in the Work Systems Design and Evaluation Group in the Collaborative and Assistant Systems area within the Intelligent Sciences Division at NASA Ames Research Center. He previously worked at NYNEX Science & Technology.

Chin Seah is a Computer Scientist at Stinger Ghaffarian Technologies (SGT), assigned to the Brahms Project at NASA Ames Research Center. He has a BS in computer engineering from Santa Clara University and an MS in computer information science from the University of Pennsylvania. Mr. Seah has applied the Brahms work system design and modeling approach to the Mars Exploration Rover and ISS mission operations. Before joining the Brahms team, he worked as a Business Process Management Consultant at Andersen Consulting and as a Knowledge Engineer at Mindbox, Inc., implementing rule-based and case-based expert systems.

APPENDIX A

A.1. Brahms: Modeling and facilitating human activities

In late 1992 NYNEX and the Institute for Research on Learning formed a partnership, with a primary objective of developing a

https://doi.org/10.1017/S0890060409990059 Published online by Cambridge University Press

work systems design simulation tool that would facilitate work practice analysis, ethnography, and participatory design (Clancey et al., 1998). In developing the tool, which became known as Brahms, it was apparent from early on that the modeling language must enable representing interactions between people doing activities, objects having structure and behaviors, and geographic areas in which people and objects were located and moved. Although the AI members of the group did not initially understand how the social scientists' concept of "activities" related to "tasks" of expert systems (Clancey, 1997*b*), it was possible to develop a modeling language and simulation architecture by viewing the problem as simulating chronological behaviors of interacting agents in a simulated world.

In modeling work practice the analyst's perspective view of the *domain* broadens: a work practice simulation models the structure and behavior of *human organizations*. Often this includes modeling the structure and behavior of *objects* (e.g., an electronic circuit) that people reason about. Objects simulated in Brahms include tools (e.g., a camera) and documents. Brahms' language provides special constructs for modeling communications, areas, and agent movements, with which one can model building facilities, object layouts in space, vehicles, communication devices, written procedures, and so forth. Because a simulation of work must show how assigned tasks are performed, a work practice simulation also models how people solve problems. Importantly, it also enables modeling how problems are discovered, defined, and handled (or not).

Modeling and simulating work practice requires representing details of interpersonal coordination in the workplace that business process models usually omit (e.g., using fax machines to pass jobs from one office to another). A work practice model goes beyond individual reasoning to simulate interactions among groups (e.g., office workers). The essence is always to understand and model how work actually gets done, not just what is supposed to happen (idealized procedures).

Another key idea is that human activities are conceptually occurring simultaneously on different organizational and temporal levels, in a form of conceptual subsumption and blending: living and working in California, working for NASA while affiliated with the Institute for Human and Machine Cognition, being a coprincipal investigator of the OCAMS Project, writing a paper, working at home. Personal activities (e.g., being a parent) are dynamically blended with work activities (e.g., how a phone call from a spouse is handled may depend on the ongoing work activity or may override work concerns).

Three existing computational ideas were merged in designing the Brahms language and simulation architecture circa 1992:

1. Neomycin's "metacognitive" architecture was adapted for its flexibility for organizing and controlling high-level processes (Clancey, 1984). Neomycin's representation of strategic methods called "tasks"[23] was adapted for simulating *Activities* in Brahms; Metarules became *Workframes*; "end-conditions" became *Detectables*.
2. Activities are activated and "running" in a *subsumption architecture* (Brooks, 1991) instead of being invoked like functions (a major change from how Neomycin's "tasks" were interpreted by its engine).
3. Following the "Distributed AI" approach (Bond & Gasser, 1988), agents and objects interact in a modeled environment: blending ideas from Cohen et al.'s (1989) simulation of fire-fighting, SimLife's simulation of animals (a game by Maxis), and the then nascent work on "simulating societies" (Gilbert & Doran, 1993).[24]

In summary, the Brahms language constitutes a particular type of *multiagent system*, based on modeling human behavior over time in a geographic space using physical tools while communicating with other people. The following summarizes how work practices are simulated in Brahms, referring to the key language constructs:

- *Groups* of *Agents* with individual *Beliefs* interact while doing personal and inherited group Activities.
- Behaviors in Activities represented as conditional actions (*Workframes*), which are sequences or alternative ways of doing something; Activities can be aborted, interrupted, and resumed.
- Reasoning (modeled by the application of inference rules called *Thoughtframes*) is contextual; that is, *Thoughtframes* are associated with *Activities*.
- Perceiving is an experience while acting (*Detectables* are associated with *Workframes*).
- *Activities* occur in parallel, in a subsumption hierarchy, modeling human behavior as a conceptual, contextual nesting of "what the agent is doing now."
- *World Facts* (the modeler's God's eye view of the environment) are distinguished from agent *Beliefs* about the world (e.g., the simulation may represent that an object is in a location with a state, but an agent may have arbitrary beliefs about the object).
- *Conceptual Objects* represent mental constructs about Agents, Groups, and Activities (e.g., jobs, phases in an activity: preparation for, during, and after journey of the Space Shuttle to the ISS); objects may be an instance of a class.
- *Areas* represent geographic places (e.g., a floor of a habitat), in which Agents (e.g., people, robots) and Objects (e.g., furniture, computers, global positioning system devices) are located; an area may be an instance of an area class, Part Of another area or connected by a Path.

Brahms is a natural extension of the knowledge-based systems concept, applied to modeling people at work. In our original inspiration, each agent in Brahms is like one knowledge-based system, but not all agents are people: Some devices with sensors and complex behaviors are modeled as agents (e.g., robots); simpler objects (or systems modeled as simple objects) can also have behaviors (e.g., an e-mail program).

In 1998, Brahms development shifted from IRL/NYNEX to NASA Ames Research Center. Sierhuis (2001) simulated aspects of Apollo lunar operations, followed by simulations of mission operations on

[23] In Clancey's (1992) reformulation, Neomycin's "tasks" were renamed "methods."

[24] Brahms was first presented at the Second International Conference on Multiagent Systems in 1996. The ideas of "multiagent systems" and "agent-based modeling" were in the air when the architecture was invented in early 1993. For example, Carley (1990) presented a "sociocognitive model of the interface between self and society," combining social and cognitive model constructs. However, her formalism did not have the construct of an "agent" with simulated behaviors in a simulated environment. Individuals only interacted in an abstract sense, which caused "exchange of information."

the ISS (Acquisti et al., 2002), a Mars analog habitat (Clancey, Sierhuis, Damer, et al., 2005), and planning operations for controlling the Mars Exploration Rover (Seah et al., 2005). The notion of geography shifted from Manhattan to the moon and Mars. Modeled objects now included robots. Our justification for simulating work shifted from simply providing insights to a work systems design team to showing lack of connectivity in operations designs and how breakdowns in flows (e.g., missing steps in procedures) were detected and handled in practice. Then in developing OCAMS we constrained the simulation to generating metrics that provided information for answering practical work system design questions.

We began the simulation-to-implementation approach in the Mobile Agents Project (2001–2006), where we used the Brahms architecture as a run-time system to develop a series of distributed workflow tools (Clancey, Sierhuis, Damer, et al., 2005) leading to OCAMS. In the run-time configuration, one or more agents are located on a given computer platform and communicate in real time with each other and to get data from and control external devices and software systems. Thus, run-time agents are interacting *processes* that interpret data, communicate, and take action in the world and may cause their platform to move (e.g., a robot) or be moved about in the world (e.g., a computer in a backpack). Generally, each person using Mobile Agents has a "personal agent" with which he or she communicates by voice and/or a GUI. The "world facts" of the Brahms simulation are replaced by the world itself, which must be inspected, instrumented, and manipulated by the agents to get information.

In the Brahms run-time configuration, an "agent" is a subsystem within a larger environment of agents. We agree with Schreiber et al. (1994, p. 29), "A KBS [knowledge-based system] is only one agent among many—human and nonhuman—and carries out only a fraction of the organization's tasks," but we would add that a workflow tool might consist of many agents, each of which includes a KBS. Correspondingly, a workflow tool constructed from Brahms does not consist of a set of modules such as "Inference," "Communication," and "Domain Knowledge Base," the common components of an expert system, but has a higher level physical and functional architecture (e.g., the rover's agents include a "navigation agent," "panoramic camera agent," and "a speech agent").

The Brahms Virtual Machine (the "engine"; Sierhuis et al., 2007) manages the simulation of each agent's behavior, physical state, and beliefs. The Brahms engine updates the state of the simulation according to each agent's inferences, communications with other agents, movements, perceptions, changes to the state of objects in the world, and so forth. In particular, depending on an agent's roles and location in the real world, including real-world systems to which it is coupled (e.g., a camera), each agent attends to different facts in the world, forms its own belief models, and carries out its own activities.

Diagrams of a Brahms multiagent system (see Fig. 1) show how the agents are distributed on platforms and their functional interactions; we also represent resulting behaviors in timelines (the AgentViewer, Sierhuis et al., 2007). In distributed, real-time systems, each computer platform with Brahms agents has a Brahms Virtual Machine, by which the agents on different platforms interact by Ask and Tell communications. Agents also interact indirectly through computer applications, physical sensors, and robotic effectors. See the references for details.