Workflow Agents vs. Expert Systems: Problem Solving Methods in Work Systems Design

William J. Clancey
NASA Ames Research Center &
Florida Institute for Human & Machine Cognition
Maarten Sierhuis
RIACS, NASA Ames Research Center
Chin Seah
QSS, NASA Ames Research Center

Prepared for special issue of *Artificial Intelligence for Engineering Design, Analysis, and Manufacturing* (AIEDAM) — "Problem Solving Methods: Past, Present, and Future," Spring 2008

Corresponding Author:
William.J.Clancey@NASA.gov
Intelligent Systems Division, M/S 269-3
NASA Ames Research Center
Moffett Field, CA 94035
650-604-2526, fax 4-4036

Short Title: "The Role of Method Abstraction in Work Systems Design"
Number of pages (excluding title, abstract, and biographies): 55
Number of figures: 2

Workflow Agents vs. Expert Systems:
Problem Solving Methods in Work Systems Design

Abstract

During the 1980s, a community of artificial intelligence researchers became interested in formalizing problem solving methods as part of an effort called "second generation expert systems" (2nd GES). How do the motivations and results of this research relate to building tools for the workplace today? We provide an historical review of how the theory of expertise has developed, a progress report on a tool for designing and implementing model-based automation (Brahms), and a concrete example how we apply 2nd GES concepts today in an agent-based system for space flight operations (OCAMS).

Brahms' incorporates an ontology for modeling work practices, what people are doing in the course of a day, characterized as "activities." OCAMS was developed using a simulation-to-implementation methodology, in which a prototype tool was embedded in a simulation of future work practices. OCAMS uses model-based methods to interactively plan its actions and keep track of the work to be done. The *problem solving methods* of practice are interactive, employing reasoning for and through action in the real world. Analogously, it is as if a medical expert system were charged not just with interpreting culture results, but actually interacting with a patient. Our perspective shifts from building a "problem solving" (expert) system to building an actor in the world.

The reusable components in work system designs include entire "problem solvers" (e.g., a planning subsystem), interoperability frameworks, and workflow agents that use and revise models dynamically in a network of people and tools. Consequently, the research focus shifts so "problem solving methods" include ways of knowing that models

do not fit the world, and ways of interacting with other agents and people to gain or verify information and (ultimately) adapt rules and procedures to resolve problematic situations.

Keywords: Work systems design, work practice simulation, model-based automation, problem solving agent, situated cognition

Introduction

During the 1980s, a community of computer scientists working with the area of artificial intelligence became interested in formalizing problem-solving methods (PSMs) in an effort called "second generation expert systems" (2nd GES; David et al., 1993). What motivated this abstraction process and what did it accomplish? How do *problem solving methods* relate to building tools for the workplace today? Indeed, what have we learned about problem solving since the 1980s? To answer these questions, we provide an historical review of how the theory of expertise has developed, a progress report on tools for designing and implementing model-based automation, and a concrete example how we apply 2nd GES concepts today.

The 2nd GES effort often involved analyzing the expert systems built in the 1970s to abstract their design and operation (e.g., Clancey & Letsinger, 1981). The motivations included: producing higher-level frameworks that would make building expert systems more efficient (called "knowledge acquisition"), 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 (through PSM libraries).

The 2nd GES analytic effort was part of the study of human problem solving (Newell & Simon, 1972), which showed that there were patterns, called "methods," by which people applied and configured finer-grained "operators" 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 kinds of problems (e.g., troubleshooting, design), 2nd GES researchers identified these additional patterns:

- All expert systems are "model-based"—the representation methods of AI programming, specifically in expert systems, introduces a new modeling method to science and engineering, namely a means of *modeling processes qualitatively* (Clancey, 1986; 1989) in contrast with purely numeric programming.
- Domain ontologies should be represented separately from the procedures that manipulate models, facilitating explanation and reuse (Clancey & Letsinger, 1981).
- Solving particular problems (e.g., diagnosing a patient) involves creating situation-specific models; this is formalized most clearly in the blackboard architecture (Nii, 1986), which makes explicit the posting and comparison of model components ("hypotheses") (Clancey, 1992).
- Processes for manipulating models can be abstracted on different levels ranging from graph operators to entire frameworks for doing diagnosis, design, etc. (Clancey, 1985; Clancey & Barbanson, 1993).

People summarized the conclusions of 2nd GES research in different ways, because models and procedures for manipulating models vary a great deal and have been represented in very different formalisms. Frameworks for organizing modeling languages and methods are provided by Chandrasekaran & 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 representation of domain entities and processes in classification and causal models seemed tractable. Instead, the debate and ambiguities concerned how to abstract and describe the procedures that manipulated models in the expert system (i.e., the PSMs).

Two decades later, software engineering has not been transformed into a process of assembling PSMs from a library, as 2nd GES researchers imagined. Has there been a failure to appreciate the benefits of a PSM library or was the vision incomplete? We argue that there is a parallel between the limitations of expert systems and limitations of the value of PSMs for software engineering. Evaluating the success of "the PSM movement" requires also evaluating the success of "the expert system movement." These limitations are founded in the narrow view of expertise that led 2nd GES researchers to believe that all software tools would be (or at least could contain) expert systems. The limitations in the "cognitivist" (Wallace, et al., 2007) view of knowledge, expertise, and problem solving—epitomized by the expert systems movement—explain why PSMs as conceived in David, et al. (1993) play only a small part in practical tools in the workplace.

On the other hand, the motivations for 2nd GES are just as important and useful today, and the effort to abstract, formalize, and reuse software components in program libraries (e.g., for C++, JAVA) has in some respects accomplished what we hoped in the 1980s. Yet writing complex systems remains a craft; it seems we are always striving for the imagined libraries of modeling components. Is a kind of 3rd GES effort required or is invention and tedious adaptation inherent in the problem of developing software tools?

To answer this question, we provide a broad review of problem-solving abstraction; the development of knowledge engineering tools; the shift in perspective about models, knowledge, and work from goals and reasoning to activities and interactive behavior; and the development of a tool for building work systems by modeling *practice*, called Brahms (Clancey, et al., 1998; 2005b; Seah, et al., 2005; Sierhuis, 2001; Sierhuis, et al., 2003). We provide an example of how Brahms has been used to design and implement a workflow tool for communications between NASA's Mission Control ground support and the astronaut crew of the International Space Station (ISS; Clancey, et al., in press). We analyze the use of abstraction in this workflow tool and relate its architecture and methods to 2nd GES terminology.

We conclude that the motivations of 2nd GES and PSM abstraction in particular has been transformed from *configuring a problem solver* to a higher-level problem of *designing agents in a work system.* 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 tools.

Historical Review of Problem Solving Methods

Problem Solving is Reasoning

The idea of problem solving methods is continuous with the long-term interest in mechanizing human reasoning (see for example, the review by Agre, 1997). The premises are simple and powerful: Good judgment is principled – knowledge is true belief — reasoning is mental. In the cognitivist paradigm, represented especially well by

Newell & Simon (1972), logical reasoning, formulated most notably by Whitehead & Russell (1910), is the essence of cognition, connecting perception and action. In early modeling frameworks, called the "Logic Theorist" and "General Problem Solver," theorem proving was the problem being studied. In subsequent frameworks (e.g., Green,1969), theorem proving then becomes a method for solving real-world problems: States in the world and goals are expressed in predicate calculus, and developing a plan of action to achieve a goal is reformulated as finding the states and actions (functions) that satisfies a theorem.

Thus, the "logicist" formulation became the foundation of *cognitivism* (Wallace, et al., 2007), the analytic framework in which human "knowledge" consists of facts and rules (axioms and theorems in predicate calculus), and "reasoning" is logical manipulation (e.g., resolution theorem proving). These researchers viewed problem solving as logical, mental manipulation of beliefs about representations.[1]

The Power of Generality: Heuristic Methods

Abstracting and generalizing is an essential aspect of human learning, in formulating a scientific model, a policy or law, or even an everyday principle for managing one's day. People naturally express models of the world and behavior as generalizations that in logical terms involve predicates and variables (e.g., "For every weekday, if 4 PM > Time

[1] For example, when Simon and Lea (1974) emphasized the role of what they called "information gathering" in problem solving, they didn't mean observing the world, but rather "the degree of similarity or difference between the expressions contained in a given knowledge state and the goal expression" (p. 333)—information about *mental constructs*, as a measure of progress in a theorem proving process.

> 7 PM, then avoid the freeways."). Thus, the vision of Newell and Simon in their magnum opus (1972) was that a single program that represented and manipulated generalizations (theorems) could be directly applied to solve many problems in different domains, hence the name "General Problem Solver." They demonstrated generality by applying the GPS method in cryptarithmetic and chess.

Most notably, Newell and Simon (1972) formulated the problem solving process as a variety of methods such as "generate and test" and "recognition." They define a method as "a collection of information processes that combine a series of means to attain an end" (p. 91). Their focus was not on the generality of the domain knowledge per se (that the theorems had variables and hence were general was essential for playing different games of chess, for example). Rather they focused on the generality of the "heuristic search" problem solving method (Newell & Simon, 1972, p. 101). This method was expressed as a program with steps such as "select-operator" and "decide-next-step." They emphasized that the formulation of heuristic search that they presented is an "encompassing scheme from which more methods can be derived," depending on the application. They also list some known alternative realizations of the undefined processes; for example, "decide-next-step" could be carried out by a fixed strategy of "always continue (one-level breadth-first search)."

In summary, it was clear from early on, at least a decade before the term "expert system" was coined, that problem solving programs could be constructed from a general problem solving method, consisting of mental operations, variously called "heuristics," "operators," and "methods" (cf. Newell & Simon, 1972, pps, 101-103, 416-417) that are general processes, instantiated and combined in different ways, depending on the task

environment. The overall program is referred to as a "heuristic;" thus for example, they refer to "the heuristic of means-ends analysis" and also "the basic system of heuristic of GPS." This wording, where "heuristic" means "discovery" or "exploratory searching"—hence in effect, "system of discovery"—sounds awkward today because a different interpretation emerged in applying the framework to real-world problems in the 1970s. Then "heuristic" shifted from meaning the model manipulation operators to meaning the domain relations that the operators manipulated, aka "domain knowledge."

Building Expert Systems: Heuristic Knowledge

Edward Feigenbaum, a student of Simon's, moved the mechanization of problem solving from logic and chess, which were than characterized as "game playing", to the broader realm of scientific human expertise, starting with chemistry (Feigenbaum, 1977). In 1965 Feigenbaum, et al. (1971) at Stanford started developing DENDRAL, a program for inferring molecular composition from spectral analysis, using the Plan-Generate-and-Test method, a variation of GPS. They concluded that real world problem solving, that is, problems that involved modeling empirical phenomena, was greatly aided by including more domain relationships, in particular uncertain syllogisms called "production rules." These rules were called "heuristics," emphasizing that they made the search process tractable.

After DENDRAL, a broader effort in domains of medicine and biology was initiated as the Heuristic Programming Project in 1970. The body of domain heuristics became known as a "knowledge base" in the early 1970s, and the HPP was renamed the Knowledge Systems Laboratory in 1982. Because computer scientists worked with experts to formulate these rules, the common view was that an expert's knowledge

consists of production rules stored in long-term memory, and thus problem solving involves instantiating rules into chains of inference relating facts to actions.[2]

At the same time, an alternative formulation based on the notion of "schemas" or "frames" developed most notably at MIT (Minsky, 1985) and Yale (Schank, 1982). In this framework, expert knowledge consists of concepts that describe patterns in objects and events through relations (or attributes). Problem solving instantiates schemas and relates them to interpret situations, construct designs, formulate plans, etc.

At the same time, the GPS formulation of operators and methods for applying operators was formalized in another framework called the "blackboard architecture" (Nii, 1986), a shared "working memory" in which operators post, relate, and refine alterative models of the world and actions. In related work, Pople (1977) formalized diagnosis as operators for constructing disease models that provided multiple and alternative explanations of or causal processes.

Researchers also recognized that generalizing heuristics made knowledge bases more concise and the generalizations might be useful across domains. For example, Davis (1980) formalized *metarules* that could heuristically control how more specific domain rules were applied. Also in the 1970s, researchers applying educational psychology to computer-aided instruction emphasized "strategic knowledge," "problem solving strategy," and "meta-cognition" as powerful ways of thinking that could be taught (Greeno, 1980).

---

[2] Representing concepts and their relations (attributes) in "semantic networks," which dominated in the 1960s (e.g., see Minsky, 1969), was useful for tasks such as question answering; problem solving required conditional and higher-order associations.

In summary, the idea of "problem solving methods" had different levels of emphasis—from heuristic processes that manipulated operators, to simply logical inference, to domain-general methods of analysis (e.g., Papert's (1972) "thinking like a mathematician"). Researchers agreed on the overall methodology: formalizing problems in some representational language, abstracting domain knowledge and the problem solving procedure. But the different domain representations, inference methods, and analytic perspectives led to a Babel of languages and terms. A workshop was held to relate the perspectives (Hayes-Roth, et al., 1983); in some respects its effect was to promote the 2nd GES analytic effort.

The Modeling Perspective

Neomycin (Clancey & Letsinger,1981), a 2nd GES, was one of the first efforts to bridge between different problem solving formulations, proving that a domain ontology and diagnostic procedure were implicitly represented in Mycin's production rules. Clancey (1984) formulated this procedure as a Hypothesize-Refine-Test method inspired by the terminology of GPS, implemented as a hierarchical set of "tasks" consisting of metarules for constructing a situation-specific model (e.g., a patient diagnosis) from the domain model. Clancey (1985; 1986; 1989; 1992) also claimed that all expert systems *necessarily* contained models and that all heuristic programs were necessarily constructing situation-specific models by applying operators that manipulated a general model of facts and associations.

The "model construction operators" framework claims that whether one views knowledge-based programs as modeling human knowledge or just as automation tools—and whether one views "knowledge acquisition" as extracting what was already stored in

the expert's brain or collaboratively developing new theories of a domain and expert behavior (Clancey, 1993a)—expert systems are computer programs that contain domain models and procedures for manipulating models to apply them in specific situations. The kinds of problems (the modeling purpose) and the kinds of modeling methods fall into general categories, leading to the System-Task-Operator formulation of problem solving:

1) Problem solving involves modeling one or more systems in the world.[3]
2) The task is the purpose, what one wants to do with the system, that is, "the problem": diagnosis, repair, configuration (design, modification, and/or plan), and control.[4]
3) Knowledge representation languages (e.g., production rules, schemas) provide a means of modeling processes occurring in the system (Clancey, 1989) using qualitative relations (e.g., type, cause, part-of, adjacency, co-occurrence).
4) Problem solving procedures (methods) consist of operators for manipulating models (e.g., chaining situation-specific models of different systems together in the Heuristic Classification Method).[5]

---

[3] A system could be a naturally occurring physical system (e.g., the human body), designed artifact (e.g., a computer system), formally defined (e.g., a chess game), or some combination (e.g., a work process involving human behavior and computer tools).

[4] Predicting the future state of a system is useful for many tasks, but not in itself a "problem."

[5] "The typology of problem tasks refers to *why* the system is being modeled; the typology of inference methods refers to *how* the model is developed" (cite AIJ retrospective). Generally speaking, when 2nd GES researchers referred to PSMs, they described what

As detailed later, the enduring value of these methods for developing sophisticated automation systems seems assured. However, by the late 1980s a different perspective on knowledge and expertise suggested that problem solving consisted of much more than manipulating models, and thus experts could not easily be replaced by expert systems, and fitting such tools into the work place required much more than reliable machines and good interfaces.

Theoretical Rebuttal: Reasoning is an Interactive Behavior

From the very start, the logicist view was controversial. For example, the philosopher-educator, Dewey (1896), argued that *inquiry* consisted of reasoning-in-action in an early critique of stimulus-response theory, and criticized Russell's equating reasoning with logic (Dewey, 1939). Wittgenstein (1953) rejected his own early support for Russell and like Dewey argued against a reductionist view of concepts and rule following (see also Wallace & Ross, 2006, p. 142). Damasio (1994) argued that emotion was essential to judgment, undermining the emotion vs. logic dichotomy that dominated the study of human intelligence. Many more trends of thought throughout the 20th century in ethology, cybernetics, anthropology, and systems theory built on a modeling framework often called "systems thinking." These fields, operating unknown to mainstream cognitive psychologists and AI researchers, developed a theory of cognition that placed

---

were variously called procedures, methods, or operators that combined how processes are modeled (the representation of the system being reasoned about) with how the models were manipulated. This is not necessarily wrong or unexpected, for operators for manipulating models would necessarily be stated in terms of the relations in the model (e.g., causality, subtype, temporality, adjacency).

human knowledge, memory, and reasoning within a complex system of biological, psychological, and social processes (Clancey, in press). This perspective eventually became known in the fields of AI and cognitive science as "situated cognition."

From the perspective of expert systems research, the essential claim of situated cognition is that human knowledge cannot be equated with models, or put another way conceptualizing does not consist of simply retrieving and instantiating stored relational networks and procedures. In effect, the nature of human conceptualization has never been properly understood or replicated because the nature of memory as a storage place is incorrect (Clancey, 1997a; 1999).

The implications for advancing theories of knowledge and action are immense. For example, the nature and role of consciousness becomes clearer: In attentively controlling behavior, a person is always conceiving "what I am doing now" (WIDN; Clancey, 1999), the present *activity*. This ongoing conception is effectively the construction of identity, a social-psychological construct. In particular, this conception must relate constraints of multiple, blended identities, involving different and perhaps conflicting obligations (accountability) and methods (what I might do now). This understanding of WIDN is dynamically and mutually developing within the conception of "the situation" (Clancey, 2004).

Knowledge systems developers had a great deal of difficulty at first understanding the situated cognition perspective because it violated the very assumptions of the information processing paradigm. "Situated" does not just mean located in the world (which is of course true) or "extracting information from the surrounding physical world" (Chandrasekaran & Johnson, 1993). Rather the person's perceiving, interpreting, and

acting is *conceptually organized* with respect to WIDN. Behavior is situated because the person is acting while (and by) dynamically conceiving what constitutes "the situation." Reflective feedback (Schön's [1987] "knowledge-in-action") is potentially fast (e.g., in dance or conversation) and often involves different conceptual organizers acting sequentially and/or simultaneously. Conceiving itself occurs in experience as a (necessarily conscious) behavior. You only know what you are going to say when you say it (even when you say it to yourself first). Action changes perception, and hence the conception of activity changes what constitutes information. (See Clancey [1997a, 1999] for discussion and references.)

Furthermore, not every human activity (the conception of WIDN) involves a problem to be solved (Clancey, 2002). Activity theory, another parallel school of thought dating from 50 years before Newell and Simon, provides a much broader view of motives, goals, and operations, including non-problematic goals (e.g., reading a magazine to relax), how conceptualization of the social setting affects the choice of methods (i.e., norms), and how a structured environment can provide an interactive scaffolding for guiding information gathering and reasoning (Hutching & Palen, 1997).

Situated cognition is not a behaviorist movement, as the cognitivists feared (Vera & Simon, 1993), but rather one that more radically turns from behaviorism than information processing was able—by emphasizing that *information is not given*, or simply "extracted," rather, the environment is both dynamically perceived in action and modified in action (dynamic interaction).[6] In contrast, cognition in the conventional information processing paradigm is more reactive, assuming a kind of given stimulus and a packaged

[6] See discussion of ecological psychology in Clancey (1997a).

response (a plan), such that human problem solving can be *replicated and improved upon* (not just modeled) by situation-action (stimulus-response) rules and stored schemas. Cognitivism thus narrowed the study of problem solving to the internal (mental) manipulation of models of the world and behavior, viewing the getting of information and subsequent action as the inputs and outputs of reasoning. As many have noted, this dichotomy between "mental processes" and "behavior" is just a continuation of Descartes' separation of mind and body (e.g., Damasio, 1994; Wallace, et al., 2007). If problem solving (reasoning) is actually a behavior, then the notion of "problem-solving method" can be greatly broadened and hence very different kinds of abstractions formulated for software engineering. In particular, as explained below, analyzing work in terms of *activities*—what people do and how they conceive of what they are doing—provides a context for how tasks are discovered, defined, and handled.

As has been shown in two decades of *Applications of AI* conferences, we can develop useful model-based tools. But these tools operate within a complex system involving other tools and human interactions. For example, Mycin's design assumed a culture has already been taken, presuming a certain medical setting with sophisticated caretakers, even if they are not antimicrobial experts. Developing tools that fit into a workplace involves a more sophisticated theory of problems and problem solving than was assumed in first developing expert systems.

Most importantly, the analytic perspective "technical rationality" (Schön, 1987) is reductionist because it presumes that gathering and interpreting information and decision making always has a routine character. Ethnomethodologists have shown how everyday work requires deciding how to categorize "situations" and deal with conflicts and

shortcomings in procedures (Clancey, 2006). Choosing among courses of action requires interpreting *how actions will be evaluated* in the current social-organizational context. For example, medical practitioners need to relate the choice of tests to the policies of the patient's insurance company. This involves judgmental reasoning to be sure, but moves beyond scientific models of the human body to a realm of negotiation and compromise (consider the work of patients themselves in attempting to overturn insurance decisions). Model-based automation can be very powerful for handling routine work, but there must be means for non-programmers to revise ontologies and rules, monitor the program, and modify operations on a case-by-case basis. This in turn requires new roles, work processes, and organizational policies, leading to an analytic framework called "work systems design"—a far broader problem than the view of automation expressed in the effort we called "building expert systems."

In summary, the limitations of expert systems stem from the limits of process models, procedures, and policies for detailing in advance how work must be (or could be) done. As conceived by people, the meanings of these formalizations (whether conceptual networks or texts) are not reducible to more models. People necessarily and opportunistically reconceive what categories and rules mean, and they often do this with other people. Even if one rejects the strong claim—that in principle human conceptualizations cannot be reduced to concept-relation networks[7]—and even allowing that meanings, justifications, sources etc. can be modeled so automation is more adaptive to circumstances—the automation cannot itself be left alone. For the seeable future at

[7] Despite the promise of neural net mechanisms (e.g., Elman, 2004), we have not yet figured out how to replicate human conceptualization (Clancey, 1999).

least, interactions with people are required so they can ensure that models, procedures, and policies embedded in automation tools are properly interpreted and adapted in practice. This interaction itself must be flexible and sensitive to contexts. The expert system must become an actor in a "web of practices" (Wallace & Ross, 2006), an *agent* that can interact with people and other tools in a dynamic physical-organizational work system. As Pollack (1991) said, "We want to build intelligent actors, not just intelligent thinkers. Indeed, it is not even clear how one could assess intelligence in a system that never acted – or, put otherwise, how a system could exhibit intelligence in the absence of action." Put another way, we need to formalize automation behaviors with respect to human activities.

Brahms: Modeling and Facilitating Human Activities

Motivation for Simulating Work Practice

In the early 1990s AI researchers at NYNEX Science & Technology Research Center in New York City were unsuccessful in fielding an expert system. An anthropologist was hired to study the knowledge and work relationships of line craftsmen in Manhattan. She brought to the AI group three areas of new expertise: 1) a *work practice analysis* approach that related tools to how the work was actually done, 2) an *ethnographic method* for gathering information about the workplace, and 3) a *participatory design* approach for bringing workers into the tool development process.

At the same time, the NYNEX Expert Systems group was competing with other systems analysts who espoused the "business process re-engineering" approach of modeling and optimizing workflows, using a business process modeling tool called Sparks (Clancey, et al., 1998; Sierhuis & Clancey, 1997). To be competitive, the team of

AI and social scientists needed to advocate their work system redesigns using a similar simulation that predicted timelines and costs. From the social scientists' perspective, the main requirement was to make social processes visible—put people on the screen so workers could participate in the modeling process and visualize how the graphics related to their own roles and situations.

The new team of AI expert systems developers and social scientists hired by Sachs used Sparks to model work practices as best they could, but were hampered by the lack of explicit modeling constructs for representing people, tools, documents, workplace layouts, communications, and movement of people and objects in geographic space. They were attempting to model not the idealized and abstracted "work flow" of business processes, but how artifacts and information were actually modified and conveyed by interactions among people and automated tools. For example, a work practice model would represent not just that a job order moved from one business functional unit to another (e.g., sales to provisioning to installation), but how the order was represented in a document and transmitted by fax, and how the manager of a particular business office would handle the incoming faxes.

Using Sparks, it was particularly difficult to explicitly represent how three or more people coordinated their actions (e.g., in testing a circuit across Manhattan) without jury-rigging the constructs in Sparks' manufacturing-inspired paradigm that centered on functional changes to the product as opposed to behaviors of the people. Multitasking, informal assistance (working on a task to which you were not assigned), dealing with breakdowns (e.g., inconsistent orders), interruption and resumption of activities (e.g., when answering a phone call) were all very difficult to express in Sparks' assembly-line

framework. In effect, business process modeling tools enabled representing how work flows through an organization, but not the work people were actually doing so that jobs and information actually moved along from one person or tool to the next (Wynn, 1991).

A work practice simulation has advantages over a model that only represents formal organizational roles and procedures:

- Reveals *informal practices*–what is not in the procedures but affects the quality of the work, including informal assistance, learning, sharing of information, workarounds, variations for efficiency, ways of satisfying the customer when the rules cannot be strictly followed, etc.
- Reveals *tacit assumptions* about work that a functional abstraction into tasks and methods ignores, e.g., who notices that an order fax arrives and how are questions about the order resolved?

When informal and tacit aspects are revealed (i.e., logistic issues and hidden side-benefits are articulated) then we can be more confident that workers' methods are not obstructed and are appropriately supported when new roles, procedures, schedules, tools, documents, etc. are introduced.

Modeling work practice requires representing details of workflow coordination that business process models usually omit (e.g., fax machines) and moving 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. The key constructs in a work practice model are:

- *Activities* (chronological behaviors of people), not just tasks (functional transformations of work products). Activities (conceptualizations of WIDN)

are effectively subsumed and simultaneous on different organizational and temporal levels: Living and working in New York, working for NYNEX, Installing a circuit for a customer on-site, Testing the circuit. Personal activities (e.g., Being a Parent) are dynamically blended with these work activities during the day (e.g., how a call from home is handled may depend on the ongoing work activity or may override work concerns).

- *Tools, Documents*, *Communications*, *Areas, Objects with behaviors, Movements*. Using these constructs, one models facilities, object layouts in space, vehicles, communication devices, etc.

In summary, a work practice simulation simulates behaviors of people, which includes simulating their reasoning about objects in a simulated environment. The emphasis is on *chronological behaviors* of people (how they organize their time, e.g., "reading email first thing in the morning") instead of only *functional behaviors* (transformations of work products, e.g., "filing out a purchase order") as in business process models, or just *reasoning* as in expert systems. Of course, some activities (e.g., constructing a plan, troubleshooting a device) are like expert system tasks. In effect, the nature of the *domain* broadens: A work practice simulation models the structure and behavior of *human organizations*, which includes modeling the structure and behavior of *objects* (e.g., an electronic circuit) that reasoning operates upon. Because a work system includes objects, models, procedures, and policies, and a simulation of work must show how tasks are performed, a work practice model contains models of problem solving within it.

Brahms: A Work Practice Modeling Approach

In late 1992 NYNEX and the Institute for Research on Learning formed a partnership, with a primary objective of developing a work systems design simulation tool that would facilitate work practice analysis, ethnography, and participatory design. 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 could not fully explain at the time how the social scientists' concept of "activities" related to "tasks" of expert systems (Clancey, 1997b), it was possible to develop an architecture that incorporated the desired constructs for modeling chronological behaviors.

Three existing computational ideas were merged in the Brahms architecture:

- Neomycin's "metacognitive" architecture was adapted for its flexibility for organizing and controlling high-level processes. Neomycin's strategic methods called "tasks"[8] became *Activities* in Brahms; Metarules became *Workframes*; "end-conditions" became *Detectables*).
- Activities are activated and "running" in a *subsumption architecture* (Brooks, 1991) instead of being invoked like functions (e.g., like Neomycin's "tasks").
- 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

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

game by Maxis), and the then nascent work on "simulating societies" (Gilbert & Doran, 1993)[9].

Key concepts in modeling human behavior are made explicit in the Brahms language to constitute a particular type of *multiagent system*:

- *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.
- Inference occurs within the context of Activities (*Thoughtframes*)
- Perceiving is an experience while acting (*Detectables within Workframes*)
- *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)

[9] 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) presents a "socio-cognitive model of the interface between self and society," combining social and cognitive model constructs. However, her formalism does not have the construct of an "agent" with simulated behaviors in a simulated the environment. Individuals only interact in an abstract sense, which causes "exchange of information."

- *Conceptual Objects* represent mental constructs about Agents, Groups, and Activities (e.g., jobs, phases in an activity: preparation for, during, and after journey of STS to ISS); objects may be an instance of a class.
- Agents and Objects are contained within *Areas*; 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 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 have behaviors, too (e.g., an email 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 the ISS (Acquisti, et al., 2002), a Mars analog habitat (Clancey, et al., 2005b), and planning operations for controlling the Mars Exploration Rover (Seah, et al., 2005). Amazingly, the notion of geography shifted from Manhattan to the moon and Mars. Modeled objects shifted from telephones to robots. Simulations of operations showed lack of connectivity and how breakdowns in flows (e.g., missing steps in procedures) were detected and handled in practice.

In summary, 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 shifting from a focus on modeling problem solving processes to modeling *work systems*. In effect, a model of work practice involves modeling how problems arise and are recognized, formulated, and resolved; the roles of different people

and the tools they use (e.g., the instruments that provide data input to expert systems), and the environment in which all this occurs. This notion of problem solving is much broader than is formalized in the model manipulation processes of PSMs.

Using Agents to Implement Automation Tools

Modeling problem solving as it occurs in the world, within *activities*, provides a context for defining automation, specifically model-based tools. In an elegant formulation, a prototype tool can be embedded in the Brahms work practice simulation, prompting the modeler to investigate and determine, for example, how information is gathered to use a tool and how tools are interactively related to the work of other people and tools, which might involve documents, communicating, networks, and so on. Thus a tool can be designed to fit the work practice, and then extracted from the simulation and deployed as an agent-based workflow system.

We began the simulation-to-implementation approach in the Mobile Agents Project (2001-2006), where we used the Brahms architecture as a runtime system to develop a series of distributed workflow tools (Clancey, et al., 2005b). In the runtime 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, runtime 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 on 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 in order to get information.

In the Brahms runtime configuration, an "agent" is a subsystem within a larger environment of agents. Comparing to 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," we would say that a workflow tool consists of many agents, each of which is a KBS. Correspondingly, a workflow tool constructed from Brahms doesn't 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"). Each agent has inferential, communication, and belief maintenance capabilities provided by the Brahms Virtual Machine (engine; Sierhuis, et al., 2007). In particular, depending on its roles and location in the real world, including other systems to which it is coupled, each agent attends to different data, forms its own beliefs, and carries out its own activities. Diagrams of a Brahms multiagent system (see Figure 1 for the OCAMS tool described subsequently) 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 summary, just as an expert system was not in general conceived as being embedded in a work system, a library of PSMs is not sufficient for building workflow tools. The expert system notion of an "executive" interpreting data and applying schemas

or rules applies most directly to the functions of the Brahms Virtual Machine.[10] The Agent-Thoughtframe-Belief framework is based on the architecture of expert systems, but is contained within a broader *work system schema* (group, agent, activity, detectable, communication act, area, movement) that enables simulating parallel, dynamic interactions among people and objects in their environment.

<< INSERT FIGURE 1 HERE >>

Practical Perspectives: Insights from Using Brahms in Practice

Distinguishing Brahms from other NASA tools (Freed, et al., 1998) and cognitive task analysis (Vicente, 1999) led us to better articulate the practical nature of activity models and use of simulation for work systems design. That is, we came to understand how to disentangle a number of theoretical and technical issues by recognizing how modeling, tools, and simulations are successfully configured in practice.

Perspective on Models (circa 2001)

- Activities and tasks are analytic abstractions (Clancey, 2002). There is no single, correct way to model and simulate work. The choice of analytic perspective(s) depends on the purpose of the model. Also, we can derive workflow diagrams from Brahms simulation runs; because these sequences are not necessarily built into the model, their emergence in particular cases can provide new information about the work system design.
- A Brahms model is not the actual knowledge, conceptualizations, situations, people, communications, etc. of practice—it is just a model. Because

[10] The Brahms Virtual Machine achieves a parallel, discrete simulation by managing communications, belief revision, agent movements, activity/workframe activations, scoping of detectables, and application of thoughtframes (Sierhuis, et al., 2007).

cognitivism equated human knowledge with models[11], the model was viewed not just as a process theory to be evaluated contextually, but as the very stuff of cognition itself—so it necessarily was either correct, incomplete, or wrong.

Perspective on Tools

- People in their everyday lives create models *as tools*, including models of other people's behavior and knowledge (Schön, 1987). Models are guides or, broadly speaking, maps that people interpret through conceptualizations.
- In predictable, patterned work settings with well-defined operations, models can be used to automate the work—to replicate routine human behavior. But when a model-based program is put into different value-laden and non-routine contexts, its operation may be interpreted as wrong and/or requiring a workaround. This means that at a minimum we must build into the tools and work practices means for adapting and/or circumventing the automation.
- Consequently, building *workplace tools* requires working with the people who will use them, not just the people who are being replaced (if any). Contrast

[11] Vera and Simon (1993) wrote: "Patterns of neurons and neuronal relations… bear a one to one relationship to the Category 4 [stored programs and data] symbol structures in the corresponding program" (p. 120). They provided no neuropsychological evidence of this isomorphism. But when Clancey (1993b) said that "Every act…is a new neurological coordination" citing neuropsychological models by Edelman and Freeman, as well as the psychological analyses of Dewey, Bartlett, Sacks, and Vygotsky, Vera and Simon replied, "He provides no evidence for these flat assertions" (p. 124).

building a medical expert system by working with nurses who will use the tool, not just interviewing physicians (e.g., Greenbaum & Kyng, 1991).

Perspective on Simulation

- In shifting from a model of mental processes to a model of work systems, the issue of building knowledge bases is replaced by designing work systems (e.g., including roles, facilities, operational procedures). We move from a tool for building tools to a tool for simulating tools and the context in which they will be used. We develop Brahms simulations to get insights about the workplace, particularly how problems are solved in practice, and thus guidance for knowing what tool to build.
- A practical simulation is not just descriptive, but makes predictions about timing, flows and bottlenecks, and costs. This resolves a scoping issue: What aspects of human life should we simulate if we are not (just) developing a model of the technical domain and reasoning? A good approach is to design the simulation to provide metrics that answer questions having a bearing on proposed changes to the work practices (e.g., schedules, automation, roles, product flows).
- A work practice model reveals unanticipated or missing interactions. By not directly modeling (building in) the workflows that form the backbone of a product-centered simulation, a work practice simulation enables evaluating more basic aspects of how the work comes together (e.g., whether work schedules of different roles interact to cause delays).

EXAMPLE: The OCAMS Workflow Automation Tool

In this section we illustrate and develop some of the points about problem solving and tools further by analyzing a mission operations workflow tool developed at NASA using the Brahms modeling and simulation tool. 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.

Objectives

The project is to automate some (and eventually perhaps all) of the file management operations between support groups and the astronauts onboard the International Space Station, as performed by a job position called the OCA Officer.[12] The broader organizational objective is to improve efficiency of mission operations by reducing personnel costs by 30% by 2012. A secondary objective is to bring NASA's research results into practical application by establishing partnerships between research and operations organizations. Demonstrating practical applications of agent-based systems integration in ground flight operations will promote the use of such tools in lunar surface operations (prototyped in the Mobile Agents field experiments).

Design Constraints

The project has the following design constraints:

- Automate routine operations of the OCA officer: mirroring,[13] archiving, up/downlink to the ISS, notification.[14]

[12] OCA = Orbital communications adapter, a card used that effectively enables a personal computer to FTP files on a satellite network. OCAMS = OCA Mirroring System.

[13] Mirroring involves replicating on a local network, called the Mirror LAN, the file operations performed on the ISS file system.

- Enable the OCA officer to retain responsibility and authority by allowing for manual overrides of all system operations.
- Enable OCA officers to modify how the system operates without programming (sustainability and adaptability in practice).
- Be sensitive to the current practices for workflow (e.g., receipt of new jobs and transmission of results), timing, communications, and authority for variances in routines.
- 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, etc.

Simulation to Implementation Methodology

We partnered with operations personnel to create 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 effectively includes the OCAMS tool used by a simulated OCA officer; it includes a prototype GUI by which someone can control the automation to understand what is happening. Both simulations model work shifts, handovers between OCA officers, and maintaining handover logs. The current and future simulations ran on one month of previously recorded data, allowing comparisons of the OCA Officer's work with and without the

[14] Notification includes speaking on the "voice loop" (a programmable network of intercoms), modifying a Flight Note (a message posted in a workflow tool), sending email, telephoning someone, and broadcasting a remark outloud in the room).

tool, based on actual data about the work products of an ISS mission (Clancey, et al., in press).

Subsequently the agents comprising the tool were extracted from the future simulation and reconfigured on multiple platforms. 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 models can be reused for design within a project (contrasted with the PSMs library idea of reuse across projects).

Analysis of File Management Process

Table 1 details how the work of ISS file management can be abstracted into an ontology of file types and handling methods. The principles for doing the three file management operations (mirroring, archiving, and notifying) are not based on the encoding of the file (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).[15] The 31 file types are acronyms assigned by OCA officers (e.g., JEDI for certain procedures; NAV for antivirus software; BME is medical; NFH for "news from home"). Files may be transferred up to the ISS (uplink), down to earth, or both.

[15] Not shown are file name templates used to recognize the file type. 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 and should be mirrored).

By examining the file type ontology, we can better understand the role of PSMs (either actual or potential) in the construction of OCAMS. The ontology can be summarized by the following four principles:

1) Operational data is mirrored and archived. Medical or personal data is neither mirrored nor archived.
2) Most down-linked items are not mirrored. Exceptions: a) Files that stay onboard after downlink; b) Changes the crew has made to the onboard 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.

<< INSERT TABLE 1 HERE >>

Without considering exceptions, the principles given above suggest these categories:

1. <Operational: UP: Mirror: Archive>
2. <Private: BOTH: Don't Mirror: Don't Archive>
3. <Operational: DOWN: Don't mirror: Archive)

But of the 60 possible combinations of {Transfer x Mirror x Archive x Notify}, 11 categories are actually required to cover the 31 file types. The exceptions and variations in notification cause file handling to be highly dependent on the type of file and customer. For example, 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). But 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 that are

exceptions to the rules. Not surprisingly, the OCA officers' manual provides one procedure per file type, rather than a set of principles (e.g., "what to do with a down-linked image file" or "how to handle operational software data").

Clancey (1992, pp 15-16) found that the relations required in an ontology depended on the modeling purposes (e.g., diagnosis, teaching, knowledge acquisition). We find in OCAMS this same process-specific, conditional character—for each function added (mirroring, archiving, notifying) the ontology becomes more branched and specific, such that 3 categories cover only 12/31 = 38% of the file types, and 8 more categories are required to cover the remaining 62%. Six categories include only one or two file types, and most are exceptions to the general rule of "mirror uplinks, don't mirror downlinks."[16]

Following the four principles listed above, we could probably write rules to infer whether a new file type provided by a customer is mirrored, archived, and how the customer is notified. But this much is obvious to the customer, too. Automation to infer file handling is not required, a customer could add new file types to the system by filling out a simple form like Table 1.

Further modeling and automation could eliminate the need for a flight controller to approve modifications to the OCAMS rules operation, but this would clearly fall into the realm of a more advanced tool, after OCAMS has been deployed and its methods and

[16] Every combination of {Direction x Mirror} occurs except BOTH/DOWN, DOWN/YES, and UP/NO, fitting the principle to mirror uplinks (hence the rule is BOTH/BOTH or BOTH/DOWN) and not to mirror downlinks (hence the rule is BOTH/UP and DOWN/NO). But there are exceptions, namely BOTH/BOTH for software configuration files (which violates both rules) and BOTH/YES (for email).

capability accepted and understood in practice. That is to say, the nature of the *file management work system*, which already has a practice for changing procedures, suggests a manual method for updating at first, rather than attempting to eliminate human checks and balances.

Towards a Work Systems Design Library: Components Reused in OCAMS

Can we relate the design of OCAMS to the generic tasks and problem solving methods of 2nd GES? A first step is to ask what components are reused in Brahms models and workflow tools, besides of course the work practice schema in the Brahms language. Here are examples from OCAMS that relate to systems integration:

- Agents that communicate with external systems (*Comm Agents*) inherit behaviors from a group (AbstractCommunicationAgent) that handles memory management and supports both simulation and real-time modes.
- An FTP client library has been reused in multiple Mobile Agents configurations.[17]
- The Brahms "base library" includes:
    - Basic file operations (copy, delete, checksum verification, etc) represented as a Brahms Input/Output group with file manipulation Java activities that other Brahms agents can inherit.

[17] 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.

- 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 multi-agent systems).
- A Brahms JavaUtility group with activities to manipulate Java objects, read Java object values, and manage properties.

We have used a table-driven method for different purposes in Brahms models. For example, in OCAMS and two previous simulations of operations, a spreadsheet representing a work schedule for one or more groups is interpreted to initialize agent beliefs about what activities are done when (i.e., the schedule timeline) (Seah, et al., 2005). Some of the specific workframe and thoughtframes that relate to schedules are reused. We expect that this method for modeling scheduled operations will play a role in most mission simulations, and more generally applicable for scheduling an agent in any domain.

In OCAMS the table-driven method is also adapted to generate thoughtframes and workframes about file types and handling rules (beliefs about attribute/values of file paths, file names, file extensions, etc.). That is, the spreadsheet serves as a knowledge-acquisition method by organizing information required from domain specialists, a means of presenting the model to others, and a means for changing the model (an agent's initial beliefs and activities).

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. Other aspects of the SSM include beliefs about the shift schedule (who is coming in to take over the role). We can expect these constructs to be adapted in the future.

In summary, the Brahms language enables formalizing and reusing model constructs, realizing the 2nd GES concept of representational tools with components more specific than "inference rule" and "schema." However, 2nd GES research especially focused on abstraction of methods for manipulating models. Of what value is abstracting tasks and methods in developing a system like OCAMS?

Applying the System-Task-Operator Framework to OCAMS: From Expert Systems to Workflow Systems

Representing an ontology of file types and different file handling operations as agents (mirroring, monitoring, archiving) derives directly from 2nd GES approach of developing a domain ontology and functional (task-specific) operators. Here is how OCAMS fits the System-Task-Operator framework:

- 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). Put another way, the configuration task here is to replicate a given structure (the ISS file structure) 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 *problem-solving method 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 2nd GES analysis is that OCAMS is not a heuristic program (yet), so the most trivial method—simple classification—suffices. One could argue that ISS file management is not the kind of systems modeling task addressed by the 2nd GES analysts.[18]

However, the file management problem actually being solved by OCAMS is more complex than it might first appear:

- OCAMS is actually *building a physical system* (the Mirror LAN), not just a representation of a design for the file system.
- OCAMS is coordinating the general model (ontology and file handling rules) with a situation-specific model of the desired configuration (SSM-CONFIG—what files need to be mirrored, archived, and monitored at this time) with a

[18] Errors do occur and put the file system (and agent system) into an uncertain state. But rather than modeling and reasoning in an expert system "diagnosis and repair" approach, 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).

situation-specific model of the current state of the world (SSM-MIRROR—the state of the Mirror LAN and SSM-ISS—the state of the ISS file system).

- 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 (according to SSM-MIRROR).

In other words, OCAMS is not just a reasoning system, a model manipulator. OCAMS is an interactive system, an actor in the world, 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 go well beyond the expert systems framework. In particular, coordinating the operations of OCAMS agents that run as distributed processes on six or more workstations is not trivial. File manipulation is the easy part. The dominant effort in requirements definition and system building involved: 1) Enabling communications between computers on different networks owned by different organizations, 2) Security of mission systems and private data using secure communication (SSL), 3) Customization of file management for special requests, 4) Verification and notification that customer requests are complete, and 5) Recordkeeping for handover logs and ongoing mission documentation. These are recurrent software engineering considerations for building office workflow tools.

Because of the advent of distributed computing, the internet, security concerns, multimodal interfaces, multiple vendor platforms, etc., having 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 how extending OCAMS' functionality involves much more than assembling additional PSMs or configuring them differently.

A Broader View: The Mission Operations Work System

As we broaden OCAMS' original mirroring function to cover other work done by the OCA officer, our perspective of "the system" being reasoned about and manipulated changes, and the focus on maintaining proper interactions with other players (people and tools) becomes more central. When we include the customers who are delivering files for uplink and receiving down-linked files, we see that the work system involves people performing other roles in the Mission Operations Directorate,[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 occurs 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, etc.

[19] MOD is the organization within the NASA Johnson Space Center that operates the Mission Control Center, usually associated with a room with three large monitors called the Flight Control Center (FCR). The OCA officers work within MCC (a secure building), but in another room, of one several "backrooms" where people support the flight controllers in the FCR, whom they can hear and speak to via the voice loop.

From this perspective, an extended OCAMS that automates all of the work of the OCA officer must be designed as an actor (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, email, and perhaps someday by speaking on the voice loop). The comprehensive process would require multiple ontologies: file types, roles in operations, mission phases.

File management correspondingly becomes a higher-order system configuration problem—such as prioritizing file transfers for different customers, given limited bandwidth and fragmented communication windows between the ground and ISS. If OCAMS were extended so it required such planning, we would probably couple it to a constraint-based tool, such as SPIFe (McCurdy, et al., 2006), rather than represent a planning capability in Brahms. Indeed, some of the files managed by OCAMS are maintained by a model-based scheduler (OSTPV; Frank, et al., in press). This example suggests, just like the functional-location decomposition of OCAMS into agents, that the preferred software engineering approach today is not construction of single programs from components, but integration of modules—often running on different platforms—that can flexibly communicate in real-time settings.[20]

[20] For example, in problem solving research the notion of "memory" focuses 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?"

So again, we see that reusability of components is important, but the systems engineering problem now includes integration and interoperability of *tools* or *services*, not just copying a formalism used in one program into another. Furthermore, just as the shift from a product-centered model to a behavioral-practice model constituted a shift from of detail from functions to transactions (client-server-product relations), this shift to an agent in a work system must focus on *maintenance of transactions*, which involves substantial negotiation of requirements and resources, whose status must be tracked, confirmed, communicated, and sometimes renegotiated.

Conclusions

In applying 2nd GES methods, we have developed Brahms, a tool for modeling work practice. Brahms is not a program that automates the design of work systems, which was the original expert systems vision. Rather, Brahms is a language and tool for expert designers.[21] By enabling designers to model and simulate alternative work system designs in different scenarios, Brahms serves like any simulation model in science and engineering. It enables better understanding causal processes (e.g., relations between roles, schedules, procedures, and workplace automation), measuring work systems flows (e.g., productivity), identifying bottlenecks, predicting how the work system might fail, and evaluating hypothesized improvements.

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

[21] Brahms is currently developed and maintained with the NASA Ames group called "Work Systems Design and Evaluation."

simulating agents that automate aspects of the work; simulating how a workflow tool would be used in practice (again with metrics); deploying an agent-based workflow tool on distributed platforms; and then by comparison with the tool in use, potentially improving the formalization of work systems and human behavior in the Brahms language and engine.

In developing OCAMS, we have shifted our perspective from building a "problem solving" (expert) system to building a workflow system, which inherently must become an actor in the world. Consequently, the library of reusable components is at a higher level than the model manipulation processes formalized in PSMs, involving integration with particular types of subsystems (e.g., handling high-volume telemetry), assisting people over time (e.g., managing a work plan), relating different representations (e.g., maps and databases), communicating in different modes (e.g., email, voice mail, conversations), and even methods for moving and behaving in location-dependent ways (e.g., robotic systems that avoid obstacles and follow people).

Nevertheless, the main idea behind 2nd GES research, that abstraction of systems, tasks, and methods 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: 1) a layered architecture, 2) functional decomposition of services into agents, 3) providing a "personal agent" for interacting with the person using the tool, 4) distributed implementation capability, 5) abstraction of domain relations into a separate domain model, enabling, for example, a table-driven process, 6) general methods for systems integration (namely, using JAVA to write a "comm agent" that mediates between an API

and other Brahms agents), 7) handshake communication protocols for tracking the status of subsystems, 8) categorizing agent messages (e.g., request, information, subscription, proposal, in the Brahms Communication Library).

Reflecting on the appropriate "grain size" for PSMs (or software reuse more generally), the trend appears to be towards programs that use specialized representations (e.g., a science database), carry out high-level modeling tasks (e.g., scheduling), or mediate between such programs (e.g., Brahms Comm Agents). This confirms the perspective of Newell and Simon (1972), restated by McDermott (1988), that the reusable component or method is the overall computational process, a self-contained package (such as a Brahms agent). Other reports of progress in developing software libraries affirm our experience that a significant opportunity for abstraction and reuse lies in higher-level components (Choo & Skura, 2004, p. 4):

> 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 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.

From yet another perspective, the Brahms work systems ontology (i.e., groups, agents, activities, workframes, etc.) for simulating practices remains fixed across domains and applications, providing an interesting twist on the idea of formalizing PSMs (Clancey, et al., 1998; Sierhuis, 2001). Brahms is analogous to the knowledge engineering tools used in the 1980s with their built-in abstractions for modeling and

reasoning about causal processes. However, Brahms' framework is in effect a language for modeling problem solving methods, but in a much broader sense intended in the 1980s, namely how models are created, manipulated, and used when detecting and solving problems in the real world—work practice.

We conclude with a claim and a hypothesis. Our claim is that work on 2nd GES 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. Our hypothesis is that such libraries never became widely used in software engineering for multiple reasons:

- The dominant challenge in developing a new workplace tool is proper integration into a complex, distributed work system involving people and other tools;
- The most obvious automation opportunities can be handled algorithmically because people need to be responsible for value-based judgments requiring diplomacy and sensitivity (e.g., how to handle a new astronaut's request to introduce a new procedure) and people usually need to be involved where physical reconfigurations are required (e.g., substituting a new computer);
- 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, et al., 2005a).

More broadly, by outlining the historical development of problem solving research, expert systems, and situated cognition, we argued that PSMs are not adequate for modeling how people discover, articulate and solve problems, that is, the practice of human problem solving. The issue is not so much that people might model the world differently (an issue central to developing instructional programs), but that simulating what people are doing in the course of a day is better characterized in terms of "activities," rather than only "solving problems." The *methods* of 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.

In conclusion, the analytic thrust of 2nd GES research was appropriate and still makes sense, however the belief that many practical tools would be constructed from PSM primitives alone was wrong. Automation tools are not standalone problem solvers—whether diagnosticians, therapists, or designers—but like human physicians and engineers, such tools are properly conceived as *agents*, which are frequently interacting 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 on. But the work itself is emergent, out of the control of any particular person or tool. Thus for an

"expert system" the most practical, reusable *methods* are ways of interacting with people and other systems to *handle discrepancies between models* (beliefs, plans, procedures, theories) *and the world*. In the expert systems formulation, the paradigm is applying a model to solve a problem. In practice, expertise includes knowing how to deal with models that don't apply, such as seeking supervisory assistance for dealing with a procedural category that doesn't fit the current situation, negotiating with peers across disciplines or shifts about who will take responsibility for certain problems, and understanding economic and political perspectives by which actions will be evaluated.

These concepts—assistance or permission, responsibility, and non-technical perspectives—move work practice into the realm characterized by Simon as "ill-structured problems" (1973). Complex events can call into question the validity of models and policies. "Tear" in models (Burton & Brown, 1979, p. 95) is sometimes handled 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 can be difficult when different analytic perspectives (e.g., scientific, ethical, economic, political world views) are at cross purposes (Schön, 1987). Here, in saying that cognition is situated we mean that expertise is inherently distributed, not all technical, and dynamically constructed in an ongoing social process. Ill-structured problems transcend

[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, and hence photographs of possible damage (taken from Earth) would not be necessary (Columbia Accident Investigation Board, 2003).

the ontology and library of methods. The 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, the challenge of designing problem solving agents is to *facilitate human responsibility*, not just by automating routine tasks, but by deferring to people when necessary, and revealing how the models might be wrong.

Acknowledgements

OCAMS has been 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, and Karen Wells. We are grateful for the vision and support of Tim Hall and Brian Anderson at NASA JSC and Mike Shafto at NASA Ames. Mike Scott and Ron van Hoof (QSS Group, NASA Ames) have played key roles in implementing Brahms and OCAMS. This work has been supported in part by funding from NASA's Constellation Program.

References

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

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

Anderson, R. H. (1977). The Use of Production Systems in RITA to Construct Personal Computer "Agents." *SIGART Newsletter, 63 (June),* 23-28.

Anderson, R. H. & Gillogly, J. J. (1976). *Rand intelligent terminal agent (RITA): Design philosophy.* RAND Report R-1809-ARPA, The Rand Corporation.

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: Lawrence Erlbaum Associates.

Burton, R. R., & Brown, J. S. (1979). An Investigation of Computer Coaching for Informal Learning Activities. *Int. J. 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. *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 Tech. Dig. 25(2),* 154–162.

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

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

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

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. (1993a). The knowledge-level reinterpreted: Modeling socio-technical systems. *Int. J. of Intelligent Systems, 8(1),* 33-49.

Clancey, W. J. (1993b). Situated action: A neuropsychological interpretation (Response to Vera and Simon). *Cognitive Science, 17(1),* 8-116.

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

Clancey, W. J. (1997b). 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: Lawrence 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 (Special Issue on Human-Robotic Interaction). 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. Series Frontiers in Artificial Intelligence and Applications* (Ritrovato, P., Allison, C., Cerri, S. A., Dimitrakos, T., Gaeta, M., Salerno, S., Eds.), pp. 3-11. 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. (in press). Scientific antecedents of situated cognition. To appear in *Cambridge Handbook of Situated Cognition* (Robbins, P., & Aydede, M., (Eds.). 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. of Seventh IJCAI,* pp. 829-826.

Clancey, W. J., Sachs, P., Sierhuis, M., & van Hoof, R. (1998). Brahms: Simulating practice for work systems design. *Int. J. 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.

and Rupert, S. (2005a). Automating CapCom using Mobile Agents and robotic assistants. *American Institute of Aeronautics and Astronautics 1st Space Exploration Conf*. NASA TP 2007-214554, Available: http://ntrs.nasa.gov.

Clancey, W. J., Sierhuis, M., Damer, B., and Brodsky, B. (2005b). 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., and Scott, M. (in press). "Multi-Agent Simulation to Implementation: A Practical Engineering Methodology for Designing Space Flight Operations." *Engineering Societies of Agents Workshop*, Athens, Greece.

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, Volume 1*. NASA. (Available: http://www.caib.us/news/report/volume1/default.html).

Damasio, A. (1994). *Descartes' error: Emotion, reason, and the human brain.* New York: G. P. Putnam's Sons.

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

Davis, R. (1980). Metarules: Reasoning about Control. *Artificial Intelligence, 15,* 179-222.

Dewey, J. (1896). The reflex arc concept in psychology. *Psychological Review, 3,* 357-370.

Dewey, J. (1939). Experience, Knowledge and Value: A Rejoinder. In *Philosophy of John Dewey* (Schilpp, P. A., & Hahn, L. E., Eds.), 3rd ed. La Salle: Open Court.

Feigenbaum, E. A. (1977). The Art of Artificial Intelligence: Themes and Case Studies of Knowledge Engineering. *Proc. IJCAI*, pp. 1014-1029

Elman, J. L. (2004). An alternative view of the mental lexicon. *Trends in Cognitive Science, 7,* 301-306.

Feigenbaum, E. A., Buchanan, B. G., & Lederberg, J. (1971). On generality and problem solving: A case study using the DENDRAL program. In *Machine Intelligence* (Meltzer, B., & D. Michie, D., Eds), Vol. 6, pp. 165-190. New York: American Elsevier.

Frank, J., Morris, P. H., Green, J., & Hall, T. (in press). The Challenge of Evolving Mission Operations Tools for Manned Spaceflight. *Proc. 9th iSAIRAS* 2008.

Freed, M., Shafto, M. G., Remington, R. W. (1998). Employing Simulation to Evaluate Designs: The APEX Approach. In *Engineering for Human-Computer Interaction* (Chatty, S., & Dewan, P., Eds.), pp. 207-223. Dordrecht: Kluwer Academic.

Gilbert, N. & Doran, J. (1993). *Simulating Societies: The computer simulation of social phenomena.* London: UCL Press.

Green, C. (1969). Applications of theorem proving to problem solving. *In Proc. of the Int. Joint Conf. on Artificial Intelligence* (Walker, D. E., & Norton, L. M., Eds.), pp. 219-239.

Greenbaum, J., & Kyng, M. (Eds.). (1991). *Design at work: Cooperative design of computer systems.* Hillsdale, NJ: Lawrence Erlbaum.

Greeno, J. G. (1980). Some examples of cognitive task analysis with instructional implications. In *Aptitude, learning, and instruction: Vol. 2. Cognitive process analyses of learning and problem solving* (Snow, R. E., Federico, P. A., & Montague W. E., Eds.), pp. 1-21. Hillsdale, NJ: Lawrence Erlbaum.

Hayes-Roth, F., Waterman, D. A., & Lenat, D., Eds. (1983). *Building Expert Systems*. Reading, MA: Addison-Wesley.

Hutchins, E., & Palen, L. (1997). Constructing meaning from space, gesture, and speech. In *Discourse, Tools and Reasoning: Essays on Situated Cognition* (Resnick, L., Säljö, R., Pontecorvo, C., & Burge, B., Eds.), pp. 23-40. Berlin: Springer-Verlag.

McCurdy, M., Pyrzak, G., Ratterman, C., & Vera, A. (2006). The Design of Efficient Ground Software Tools. In *Proc. of the 2nd IEEE Int. Conf. on 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 Publishers.

Minsky, M. (1969). *Semantic Information Processing*. Cambridge, MA: MIT Press.

Minsky, M. (1985). *The Society of Mind*. New York: Simon and Schuster.

Newell, A., & Simon, H. A. (1972). *Human problem solving*. Englewood Cliffs, NJ: Prentice-Hall.

Nii, H. P. (1986). Blackboard Systems. *AI Magazine, 7(2),* 38-53 and *7(3),* 82-106.

Papert, S. (1972). Teaching Children to Be Mathematicians vs. Teaching About Mathematics. *Int. J. of Mathematics Education and Science Technology, 3,* 249-262.

Pollack, M. (1992). The Uses of Plans. *Artificial Intelligence, 57,* 43-68.

Pople, H. E., (1977). The Formation of Composite Hypotheses in Diagnostic Problem Solving: An Exercise in Synthetic Reasoning. *Proc. IJCAI,* pp. 1030-1037. Cambridge, MA.

Schank, R. C. (1982). *Dynamic memory: A theory of reminding and learning in computers and people*. Hillsdale, NJ: Lawrence Erlbaum.

Schön, D. A. (1987). *Educating the reflective practitioner*: *Toward a new design for teaching and learning in professions.* San Francisco: 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. on Human-Computer Interface Advances for Modeling and Simulation*, pp. 73-78.

Searle R. (1969). *Speech Acts: An Essay in Philosophy of Language.* Cambridge University Press.

Sierhuis, M. (2001). *Modeling and simulating work practice.* Ph.D. thesis, Social Science and Informatics (SWI), University of Amsterdam, The Netherlands.

Sierhuis, M., & Clancey, W. J. (1997). Knowledge, Practice, Activities, and People. In *Proc. AAAI Spring Symposium on Artificial Intelligence in Knowledge Management* (Gaines, B., Ed.), pp. 142-148.

Sierhuis, M., W. J. Clancey, Seah, C., Trimble, J., & Sims, M. H. (2003). Modeling and simulation for mission operations work systems design. *J. of Management Information Systems, 19(4), 8*5-128.

Sierhuis, M., Clancey, W. J., & van Hoof, R. (2007). Brahms: A multiagent modeling environment for simulating work practice in organizations**.** *Int. J. for Simulation and Process Modeling, 3(3),* 134-152.

Simon, H. A. (1973). The structure of ill-structured problems. *Artificial Intelligence, 4(3),* 181-202.

Simon, H. & Lea, G. (1974). Problem solving and rule induction. *Models of Thought* (Simon, H. A., Ed.), pp. 329-346. New Haven: Yale University Press.

Vera, A., and Simon, H. (1993). Situated Action: Reply to William Clancey. *Cognitive Science, 17(1),* 117-135.

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.

Whitehead, A. N. & Russell, B. (1910/1913). *Principia Mathematica*. Cambridge, Cambridge University Press.

Wittgenstein, L. [1953] (1958). *Philosophical Investigations.* New York: Macmillan.

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: Lawrence Erlbaum.

Author Biographies

William J. Clancey works at the NASA Ames Research Center and Florida Institute of Human and Machine Cognition. He received a BA degree in Mathematical Sciences from Rice University (1974) and a PhD in Computer Science from Stanford University (1979). Chief Scientist for Human-Centered Computing, in the Intelligent Systems Division at Ames, he 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 RIACS/USRA, located at NASA Ames Research Center. He is a Co-Principal Investigator for the Brahms project, working in the Work Systems Design & Evaluation group in the Collaborative and Assistant Systems area within the Intelligent Sciences Division at NASA Ames Research Center. Previously, he worked at NYNEX Science & Technology. 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.

Chin Seah is a computer scientist at Science Applications International Corporation (SAIC), working at NASA Ames Research Center on the Brahms project. He has applied the Brahms work system design and modeling approach to the Mars Exploration Rover and International Space Station 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. He has a B.S. in computer engineering from Santa Clara University and an M.S. in computer information science from the University of Pennsylvania.

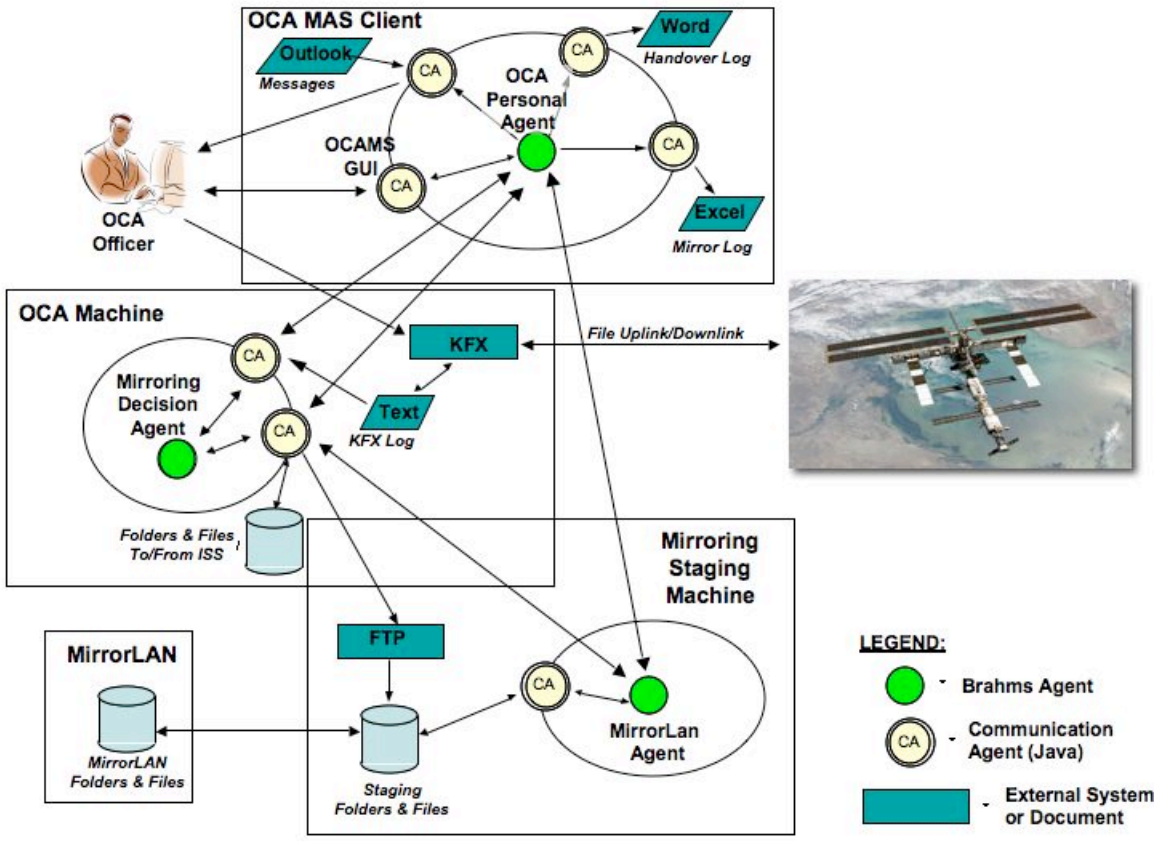

Figure 1. OCA Mirroring System (OCAMS) using model-based systems integration to automate some of the file management between ground support and the International Space Station (ISS).

Table 1. OCAMS Ontology of File Types and Handling Procedures. Thirty-one file types are categorized by the function of the data and/or customer providing or using the data. Transfer directions refer to "UP" to the International Space Station and "DOWN" from ISS to the Earth. A mirrored file is copied to a ground-based duplicate 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 outloud to someone across the room at another workstation.

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}