13 Organization and Work Systems Design and Engineering

From simulation to implementation of multi-agent systems

MAARTEN SIERHUIS[†‡] WILLIAM J. CLANCEY[§¶] CHIN H. SEAH[||]

13.1 INTRODUCTION

In this chapter we present a specific approach to analyzing and designing work systems we call *Work Systems Design* (WSD), that is based on a theory of modeling and simulating work practice. The theory is made explicit in an agent-based modeling and simulation environment called *Brahms* (see also Chapter 12). In the next section we start with explaining what work systems design is. We then give a short overview of the Brahms language. This overview is in addition to the Brahms description in Chapter 12. It gives a more detailed description of some of the language constructs for modeling work systems. Next, we describe the *"from simulation to implementation"* software engineering methodology. In this we turn an agent-based simulation of a work system into a distributed multi-agent system (MAS). It is here where Brahms differs from other agent simulation environments and where agent-based modeling and simulation meets agent-oriented software engineering (22). After the description of the methodology, we describe NASA's OCAMS project in which this approach was successfully applied. The OCAMS MAS has been in operation 24x7 for over six months as of the writing of this chapter, in NASA's Mission Control Center (MCC) for the International Space Station (ISS) in Houston, TX. In this project we first modeled and simulated the current work system in Brahms as an agent-based simulation of people, artifacts and environment. We then designed the OCAMS system as a MAS simulation interacting with a simulation of people agents. We end this chapter with some conclusions about the approach and experiences from the OCAMS project.

[†] Carnegie Mellon University Silicon Valley, NASA Ames Research Center, M/S 269-1 Moffett Field, CA 94035 USA; `Maarten.Sierhuis@nasa.gov`
[‡] MMI Group, Delft University of Technology, 2628 CD Delft, The Netherlands
[§] NASA Ames Research Center, M/S 269-1 Moffett Field, CA 94035. USA; `William.J.Clancey@nasa.gov`
[¶] Florida Institute for Human & Machine Cognition Pensacola, FL 32502
[||] Stinger Ghaffarian Technologies, NASA Ames Research Center, M/S 269-1 Moffett Field, CA 94035. USA; `Chin.H.Seah@nasa.gov`

13.2 WORK SYSTEMS DESIGN

A work system is a "natural" setting for those who work within it, because every setting is natural for those who frequent it. In this chapter we focus on workplaces for work systems. A workplace is where the work system comes alive, where the daily work is continuously being performed, based on familiar past performance as well as changes unknown until faced. In other words, work is like a symphony, well rehearsed, but always different. It is this "symphony" we are interested in composing (designing) changes, using an engineering method that allows us to predict the impact of designed change on the current system.

A work system covers more than the people working inside it, just like a whole that is bigger than the sum of its parts. From the perspective of designing and engineering a work system, we see the system as a socio-technical system of people, artifacts and (computer) systems, their (joint) activities, organization, and communication. Everything is interconnected (13). The design of a work system sometimes involves the completely new design of people's tasks, procedures and activities. However, often a change to a work system is created by the introduction of a new software system that changes the activities of people. It is this change that we want to understand upfront in the design of software systems so that we can predict how the system should operate within the work system and how it changes the work.

13.2.1 Existing work systems design methods

There are a number of existing methods for analyzing and/or designing work systems. Most of these methods are referred to as "soft system" methods. In these methods working with the people in the workplace is one of the big differences from more traditional (software) engineering methods. A number of methods were developed as by-products of the "Scandinavian approach" to systems design, called participatory design. Other approaches are by-products from tasks analysis mostly performed by cognitive psychologists. Below are four existing approaches from the participatory design realm:

- *Design at Work: Cooperative Design of Computer Systems* (17) gives detailed examples, theory, and methods for participatory design. It gives the Scandinavian perspective that defined observational studies of workplaces as a theoretically grounded activity in software engineering.
- *Contextual Design (CD)* (1) can be seen as a method for "contextual inquiry," including how to observe and work with customers, conduct interviews, and how to model work as organizational flow, task sequences, artifacts, culture, stakeholders, and physical environment; and how to redesign work.
- *Soft Systems Methodology* (SSM) (6) descibes an engineering method for understanding the "soft" issues of a system, through examples how different stakeholders use different "lenses" looking at the same problem. Checkland, himself a civil engineer, realized that engineered artifacts are part of a human

work system he called an activity system. The process of designing an activity system is a *holonic* modeling process that involves many parties (customers, users, designers, policy makers, etc.).

- *Cognitive Work Analysis* (CWA) (36) is another method for analyzing work systems and designing software systems, based on detailed mapping of information flows and tasks. CWA was developed by Ransmussen and colleagues (26): Work models are detailed cognitive models rooted in cognitive task analysis for tool design, and hence observation must be systematically organized to understand the domain.

What all these approaches have in common is that they are mostly ad hoc modeling approaches. Most of them are pen and paper based, and have no specific modeling and simulation tool that help guide what to do; although some of the approaches lend themselves to one or more modeling methods; such as system dynamics modeling (34) for SSM, and GOMS (5) for CWA.

13.2.2 A brief history of work systems design

WSD finds its roots in business anthropology, which started in the nineteen-eightees as the Scandinavian approach to system design (12; 17). The use of photography and video in studying workplaces was pioneered in cultural anthropology. Today, digital photography and video makes the use of capturing the practice of people in the workplace much easier. Descriptive textual notes of a setting are replaced by high-resolution digital photos, or a short videos recorded on ones digital camera. These recordings are extremely useful in the analysis of how people use space, interact with artifacts and communicate with each other.

Improvement of procedures and tasks through the design of new technology used in the workplace has taken place throughout the last fifty or so years. Studying schools and other natural settings by developmental psychologists (20), modeling workflow by management consultants and organizational analysts (30) has led to major fads in business process reengineering. Often it is not the academic studies that create change in the development of methods, but the application of academically rooted and developed methods to real-world problems that move a field forward. Similarly, this happened with the coming of expert systems and knowledge acquisition (3). Where computer scientists and business consultants failed, anthropologists, together with computer scientists, succeeded in developing a participatory software engineering practice (24; 19).

For a more complete history of how WSD came to be, we refer the reader to Clancey (9). At this point in time, we are applying modeling and simulation to the understanding of work practices—how people work. This is where social science is finally meeting systems engineering, and a truly holistic human-centered engineering approach—an approach in which people are at the center and people, environment and systems are understood as an interacting system—is created. This is the topic of the next section.

13.3 MODELING AND SIMULATION OF WORK SYSTEMS

In this chapter we argue that we need to move the design of human work systems from a seemingly mystical social-science art-form to systems engineering with, more or less, formal methods, tools and guidelines. To do this, we need to show how the social-science "methods," mentioned in the previous section, can be made useful as data-gathering methods, not for ethnographic analysis, but for the purpose of work systems design.

What is needed is a theory of analysis and design of human activity systems. This theory will then form the foundation upon which we can develop a systems engineering approach that has clear guidelines and procedures to follow; a sort of cookbook for analysis and design of work systems. We argue that modeling and especially simulation focuses the attention on system engineering methods. This is mostly because a computer simulation requires a formal description of the system in order to simulate changes to the system over time, based on operationalization of theories.

13.3.1 Designing work systems: What is the purpose and what can go wrong?

If we develop a methodology with appropriate methods for designing work systems, what is its purpose? How many new work systems get developed from scratch? The answer is, not many. There are situations, for example at NASA, where a complete new work systems gets designed and put in place for a short duration of say three months to ten years. Space mission operations organizations are such an example. However, in most cases design mostly entails changes to existing systems. More often than not there is a current system that exists and the objective is to change just a small part of the current system to make it more efficient or effective, or both. In the first, not so frequent case, the use of modeling and simulation is to design the future system as a computer model and then to simulate it in order to understand how the system will behave in the future. We use the methods to understand a system in the future before it is actually there. In the second, more frequent case, the objective is to understand the change trajectory of an existing system and understand how the designed change fits into the current system.

In both cases we are talking about predicting future behavior of people and machines. The danger of a modeling and simulation approach is the well-known "garbage-in-garbage-out" dilemma. In the nineteen-nineties this issue became well known in the process re-engineering fad (10). Expensive, external business consultants were hired to re-engineer a company's work processes to make them "leaner and meaner." On top of bringing in outside consultants, who knew very little of the company's work process, most of the time this was done just prior or during a company merger. The approach became synonymous with "cutting cost by cutting heads," while in the mean time management "forgot about the people" (11). The approach was not driven from understanding the work of the people within the system, but it was mainly driven from management's view of cutting cost. The result was often that the models and workflow simulation tools that were used were geared to showing high-level tasks and information flow, and the objective-function of the simulation effort was to cut-down on processing time and simplify information flows to cut down time as it was modeled.

Clancey, et al (7) have shown how business process-flow modeling approaches from the nineties lacked the capabilities of modeling people's work. Together, Clancey and Sierhuis have been working on an agent-based modeling approach for modeling people's work at the practice level—how work really gets done. This work practice modeling approach is the topic of the next section, and the main topic of this chapter. However, before we get to that we explain why it is that agent-based modeling and simulation is a powerful method for work system design and engineering, we turn briefly to the fact that management often does not like modeling and simulation of work systems, regardless of the fact that, if done right, it allows for an "objective" and "quantifiable" view of the work system's improvement.

13.3.2 The difficulty of convincing management

It seems contradictory to the modeler and engineer that management is uninterested in getting objective and quantifiable measures about their organization, in order to improve the work product. However, it is on purpose that we use quotations marks around the words objective and quantifiable at the end of the previous section.

First, it has to be understood that the modeler can never be completely objective. It is important to recognize that a model of a work system (any system for that matter) is build with a particular point of view, namely that of the modeler. It is extremely important for any modeling effort to have a well-defined objective for the modeling effort. This objective needs to be developed participatory with management and workers from the work system that is to be modeled. Management will and should drive the when, what and why of the modeling effort. Without management (or customer, i.e. the one who pays for the effort) approving and standing behind the when, what and why of a modeling and simulation effort, the project will not succeed. Besides the so called buy-in from management, it is as important to get buy-in from the workers whose work is going to be modeled, because they will be the ones that benefit or not from the outcome of the effort. Without their help the accuracy of the model will be compromised and the garbage-in-garbage-out phenomena will crop into the effort.

The second important aspect of getting buy-in is to convince management that the simulation can quantify important variables that are part of the why of the effort. Often the why's of a modeling and simulation effort of a work system is about increasing efficiency, which most often will translate into reducing cost. However, this is not always the case. The example we use in this paper is based on an actual NASA project in which the why of the modeling effort, as put forward by management (the customer), was indeed related to efficiency, however not to reduce cost (because no jobs were lost), but to increase efficiency of the work system by introduction of a designed agent-based workflow system to reduce manpower for one particular organizational task. The overall objective was to add this manpower to other, understaffed tasks.

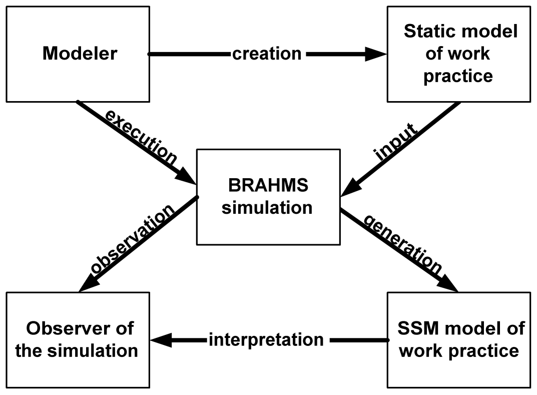

Modeler
creation
Static model of work practice
execution
input
BRAHMS simulation
observation
generation
Observer of the simulation
interpretation
SSM model of work practice

Fig. 13.1 Relation of a model of work to a description of the work practice

The key of convincing management of the benefits of modeling and simulation for work systems design is to (a) use it as a decision-making method for management and (b) use it as a design and engineering method to provide an implementation of the decision that is supported in (a). As mentioned before, the quality of the model of the work systems depends on the amount of participation one gets from the workers in the organization that is being modeled. This participation goes two ways; (c) without participation the quality and the outcome of the simulation should always be questioned, and (d) without participation the implementation of the resulting design will not be accepted by those who have to live with it.

13.4 WORK PRACTICE MODELING AND SIMULATION

Work practice modeling and simulation is a model-based method for developing a time-based representation of the practices of people. Whereas work practice modeling is a process of developing static representations of a work system at the practice level, simulation adds a time dimension to such static models allowing us to view the model at different moments in time, and thus providing a way to see how the work happens in practice. In this section we describe what is meant with work practice. The modeler in Figure 13.1 develops a model of the work using the representational power of the Brahms language (see section 13.5). Model creation is an elaborate process of data collection and work description that leads to a static model of the situated activities of the individuals involved. Using the Brahms simulator, the model is simulated and a dynamic behavioral model of the work (i.e. a situation specific model of the practice) is generated. The observer of the simulation model can observe the model during and after the simulation, interpreting the work practice model.

13.4.1 Practice vs. Process

Many researchers in the social sciences use the word practice as if it is a well-defined concept that everyone knows, described as "work as experienced by those who engage in it" (4). Practice is also called "lived work"—"what work consists of as it is lived as part of organizational life by those who do it" (4). In short, practice is doing in action (35). However, it is difficult to describe what a practice is. People notice when something is not a practice, and can often describe why—"we normally don't do things this way." It can be said that a group of people has developed a practice, but when asked to describe what it consists of, we find it difficult to describe in words. As such, practice is part of our tacit knowledge (25). An ad hoc definition of *practice* that we use is:

> *The collective performance of contextually situated activities of a group of people who coordinate, cooperate and collaborate while performing these activities synchronously or asynchronously, making use of knowledge previously gained through experiences in performing similar activities.*

Practice is to be contrasted with formal process specification of what work is to be done. In the workplace itself, processes are often idealized and constitute shared values. Narratives that people record or present to authorities cater to these policies or preferences and create an inherent conflict in the work system between what people do and what they say they do. Two fundamental concepts related to the practice versus process distinction are *behavior versus function* and *activity versus task*. Process models (e.g, workflow diagrams) are idealized functional representations of the tasks that people in certain roles are expected to do. In contrast, practice concerns chronological, located behaviors, in terms of everyday activities, for example, reading email, meeting with a client, and sorting through papers. Activities are how people "chunk" their day, how they would naturally describe "what I am doing now" (8; 9).

13.4.2 Modeling Work Practice

In the model of Technical Rationality, the notion of a practice is automatically associated with the application of scientific knowledge in "major" professions (31). Not only are we claiming that practical knowledge is an important category of knowledge, but the concept of work practice allows us to view practical knowledge within the scope of all kinds of practitioners (not only within those of "major" professions). Here we focus on creating a framework that allows us to investigate, collect data about, and model the work practices of any group of individuals from any type of profession. Even more so, we focus the attention on work situations where multiple individuals from different professional backgrounds are collaborating.

Work practice is constituted by the way people act and interact in their daily tasks as part of their job, socially and psychologically situated within their environment. It is situated action (35) described in terms of activities and their context. It is how people act and interact in order to accomplish what they have to do. In the next sections, we give definitions of the important elements: community of practice, activity, communication, artifacts, and geographical environment that are important elements in a theory on modeling work practice.

13.4.2.1 Community of practice People who are engaged in a work practice together belong to a community that has an identity (38). Together this group of people is engaged in choreographed activities, acting either together or on their own. For example, consider the interplay of activities of people working and dining in a restaurant. There are different roles that are played, the waiters, the chef, the dishwashers, the maître d'homme, etc. Even the dinner guests are part of the practice. They all engage in interplay, a kind of theatrical improvisation in real-time. An unwritten play, so to speak, unrehearsed, but still they never forget their lines. They seem to know what the play is about, reacting to each other, never stepping out of character. They all seem to know their parts. They react to and communicate with each other. They have all played their parts before they have ever met each other, because their actions are based on similar previous experiences working and eating in restaurants. This is what the activity of working in a restaurant, and going to eat in a restaurant is all about. It is a conceptual choreography. Everyone knows their roles, because they have done it so many times before. They are part of a community of practice that exists inside and outside the restaurant. This type of community of practice focuses on a group of people who produce something together.

> **(a. Community of practice)** *A community of practice is a group of individuals, each with different individual skills and knowledge, performing complementary activities while producing something together, that collectively can be seen as a unity within a practice.*

We define a second type of community of practice (b). The distinction between the first definition and the second is the type of people that belong to a community. The first definition (a) includes individuals playing different roles and performing different activities. The second definition includes people with similar skills and knowledge, playing the same role and performing similar activities. This type of community of practice includes the professional communities, such as the Java programmers at company X, the architects at company Y, or the group of waiters at a restaurant, etc. However, it does not by definition have to be a professional community. For example, we could also talk about the practice of the group of people meeting each other regularly at the water cooler. Such communities are more informal or social, and do not have to include people from the same professional background. The point is that this definition of community of practice focuses on people that play similar roles and perform similar activities.

> **(b. Community of practice)** *A community of practice is a group of individuals playing similar roles, each with similar skills and knowledge that allow them to perform the same activities, that collectively can be seen as a unity within a practice.*

Both definitions are useful and hold true at the same time. The reason for making a distinction is for the purpose of identifying these types of communities of practice, and the ability to talk about their practice as a whole. For purpose of modeling, it is useful to make a distinction in the practice of a community in terms of different groups of people in an organization performing different activities, or in terms of a group of people performing similar activities. By describing a community of practice as a group to which individuals belong, we can represent people's practice in terms of the sum of the communities (groups) they belong to.

13.4.2.2 Activity An important concept in modeling practice is that of an *activity*. In describing the practice of a group of people, we describe the individual and group activities over time, as a *day in the life* (DITL) model of the group. To understand activities we must first understand that human action is inherently social. The key is that "action" is meant in the broad sense of an "activity," and not in the narrow sense of altering the state of the world (15). Describing human activities as social means that the tools and materials we use, and how we conceive of what we are doing, are socially and culturally constructed (37; 21). Although an individual may be alone, as when reading a book, there is always some larger social activity in which he or she is engaged. For instance, the individual is reading the book, as relaxation, while on vacation. Engaging in the activity of "being on vacation," there is an even larger social activity that is being engaged in, namely while on vacation still "working for the company" and while working also "being a parent," and so on. The point is that we are always engaged in a social activity, which is to say that our activity, as human beings, is always shaped, constrained, and given meaning by our ongoing interactions within a business, family, and community. An activity is therefore not just something we do, but a manner of interacting. Viewing activities as a form of engagement emphasizes that the conception of activity constitutes a means of coordinating action, a means of deciding what task to do next, what goal to pursue, in other words, a manner of being engaged with other people and things in the environment. The idea of activity has been appropriately characterized in cognitive science as intentional, a mode of being. The social perspective adds the emphasis of time, rhythm, place, and a well-defined beginning and end.

Conceptually, we can view activities as the "what we are doing at each moment in time". Goals can be viewed as the "why we are doing what we are doing," while tasks can be viewed as the "how we are doing what we are doing." In other words, goals and tasks are being executed within activities, or better, activities at the meso-level are our social conception of goals and tasks at the micro problem-solving level. Viewing work as the collective activities of individuals (14) allows us to understand why a person is working on a particular task at a particular time, why certain tools are being used or not, and why others are participating or not. This contextual perspective helps us explain the quality of a task-oriented performance.

We can be in more than one activity at the same time. This is situated action, an activity that is not fully planned in detail, and can be interrupted and resumed (35); think about putting on your pants in the morning, and the phone rings. While there is not a control program that runs and controls our activities, a situation that suddenly comes up has to be dealt with, without articulated task knowledge. While switching context, the higher-level activity is still being engaged in. Therefore, it is such higher-level activities that constrain us from switching context from one lower-level activity to another lower-level activity and back.

People choose which activity they engage in, but cannot choose this for others. Therefore, when people suddenly enter our space to interact, we juggle the activities we engage in. We suspend the current activity, start a new one, stop a third one never to come back to it again, etc. We act in the situation and react to our environment. This is how the work practice of an organization is formed, and work happens or does not happen. If we are interrupted all the time during our work activities, we start acting a certain way, conscious or unconscious. We might hide, so that interruptions are minimized, or we might just do those activities that do not require a lot of time, or can be interrupted at any moment. In short, the situation and the environment determine our activities, which in turn form our work practice.

> **(Activity)** *An activity is a collection of actions performed by one individual, socially constructed, situated in the physical world, taking time, effort, and application of knowledge. An activity has a well-defined beginning and end, but can be interrupted.*

13.4.2.3 Communication In order for two or more people to collaborate they need to communicate. In Searle's Speech Act theory (28), the meaning and intent of *speech acts* are formalized in terms of sending and receiving communicative acts triggering response actions. A speech act has at least four distinct types of acts (28) that are all part of the same collaborative activity:

1. Uttering words is performing an *utterance act*. This is modeled with a time consuming communication activity the sender is engaging in.

2. Referring and predicating is performing a *propositional act*. This is modeled as the content of the message (beliefs of the sender) being communicated in the communication activity.

3. Stating, questioning, commanding, promising, etc, is performing an *illocutionary act*. This is modeled as the type of communication, defining the type of response expected from the receiver(s) of the communication.

4. The consequence or effect on actions, thoughts, and beliefs of the hearers is the *perlocutionary act*. This is modeled as the activity the receiver(s) engages in, because of receiving the communication. Together with the utterance act of the sender this defines the collaborative activity both sender and receiver(s) engage in.

Searle went as far as defining a taxonomy of types of speech acts in which he classified all types as embodying one of five illocutionary points: assertives, directives, commissives, expressives, and declarations (29). Speech Act theory analyzes communication in terms of its illocutionary point, -force and propositional content. Using this type of communication analysis, we can model the sequence of communications in a collaboration activity between sender and receiver, as well as the intention and meaning of the speech act. In our theory for modeling work practice sending and receiving information is minimally a coordination activity.

However, in analyzing the way collaboration occurs in practice, we also need to analyze communication in terms of how it actually happens in the real world, thereby modeling collaboration as it really occurs. Speech Act theory abstracts communication in terms of patterns of commitment entered into by the speaker and the hearer. While this is important, in modeling communication as it happens in practice we also need to take into account if a communication activity between two people actually happens, or does not happen. We need to include the communication tools used in the activity, because the type of tool has an impact on when and how the hearer receives the speech act. Today, communication is more and more efficient and certain communication tools are used globally. Phones, voice mail, e-mail, and fax are communication tools that are more and more taken for granted in the way that we use them. However, it should not be taken for granted that we all have created our own practice around the use of these tools in certain situations. We emphasize the point that collaboration is very much defined by our practice surrounding our communication tools, and that we, therefore, need to include the use of communication tools in modeling how people actually coordinate their collaboration in the real world. We need to include a model of the workings of communication tools (phone, e-mail, etc), and how they are used in practice.

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

13.4.2.4 Context Work is performed within a three-dimensional geographical environment. The restaurant we have dinner at, the office we work in, and the Moon crater Apollo astronauts explored, are all examples of places and spaces that enable, while at the same time constrain, our work. The artifacts we use in our work, such as communication- and information tools, are also located in a three-dimensional space. We are constrained to our three-dimensional world and it defines very much how we can perform our work. For example, when the phone rings, we cannot hear it if we are not in the same room as the telephone. We also cannot observe specific changes in a location when we are not there. For example, if someone turns off the light in a room and you are not there, you will not observe this and therefore will not be aware of the fact that the light in this room is now off. To show the effect of the environment on the practice, we have to include a model of the environment, including the people and artifacts in it, in a model of work practice.

> **(Environment)** *The environment is a description of the physical environment and the people and artifacts located located within it and performing activities.*

People use and/or create artifacts in almost all activities they engage in. When in the activity of hammering a nail, we use a hammer and a nail, and we end up with a nail in whatever artifact we have hammered it in. If we try to understand this activity in context of performing it in the real world, we cannot leave out the artifacts. The artifacts constrain the way we perform activities. It is part of our context, and we have no choice but to interact with the physical world in order to act. We need to include these artifacts into our model of work practice. Leaving them out would miss the opportunity to understand the reason for performing activities. In other words, the artifacts are as important in the work practice as the people are.

(Artifact) *An artifact is a physical or digital information object in the world.*

Important in modeling work practice is how the artifact is used and conceptually understood within the activity. George Mead's social-behaviorist notion of *instances of the universal* (23), as well as Heidegger's notion of *break down* and *readiness-at-hand*, explain the role of objects—artifacts—in an activity. Mead, as well as Heidegger, use the hammer and the activity of hammering as the example in which the hammer is the object that turns into a tool—as an extension of the hand. Mead's idea is that the concept "hammer" is the universal and the object used in the specific activity is the instance of the universal. Therefore, for Mead, the role of the hammer is *socially bound* to the activity, and is not a property of the object itself. If the person who is hammering uses a piece of wood to hammer in the nail, that piece of wood becomes the instance of the universal during its use in the activity, and thus plays the role of a hammer. In other words, the object is transformed into the tool used to hammer in the nail. Heidegger, in essence, says the same. Only he speaks to it through the understanding that objects and their properties are not inherent in the world, but arise only in an event of break down in which the object becomes present-at-hand. To the person hammering, the hammer as such does not exist. It is part of the readiness-at-hand, and it is taken for granted in the activity. Without the user's identification as an object It is only in the break down, for example when the person cannot find the hammer when he wants to hammer in the nail, that the object is present for the user. Whichever notion speaks to you, the issue that is important in modeling work practice is how artifacts are used, created and conceptually understood by people within an activity. Figure 13.2 shows this relationship.

It is the use of the artifact in the activity—its role—that transforms the artifact into a *tool* or *resource* in, or a *product* of the activity, used or created by the subject. Outside the activity the artifact is just an object in the world. To the observer, and to the modeler, the object is necessary for the activity to be performed.

(Tool/Resource) *When an artifact is being used in an activity, it becomes a tool or resource in the performance of the activity.*

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

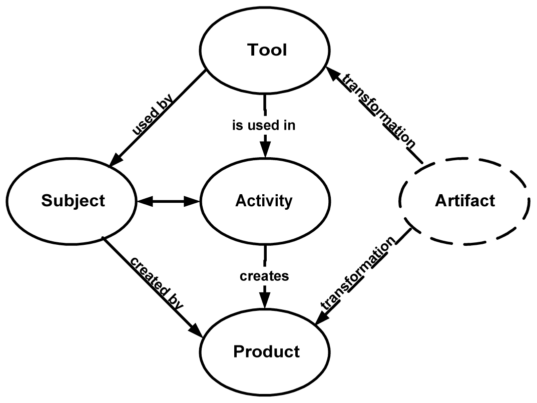


Fig. 13.2 Mediated relationship of artifacts in activities

13.5 THE BRAHMS LANGUAGE

The name *Brahms* refers to the agent-oriented language (AOL), the multiagent modeling and simulation environment and the MAS development and execution environment. The Brahms language was developed to incorporate and operationalize the theory of modeling work practice described in the previous section. A Brahms model or program can be written using a standard text editor, or using one of the two available interactive development environments for the Brahms language; the Composer (see Figure 13.11 or the Brahms Eclipse plugin. We refer the reader also to chapter 12, section 5 for a comparative description of Brahms with other agent simulation languages. Here we give some more detailed of the Brahms language and its syntax. For a more detailed description of the Brahms language we refer the user to (33).

13.5.1 Simulation or execution with Brahms

A Brahms model or program is executed by the Brahms virtual machine (BVM). All components of Brahms, including the compiler, the BVM and both IDEs are developed in Java, and thus run on most systems that run Java. The BVM can be run in one of three modes:

1. *Simulation mode*: In this mode the BVM is a multi-agent discrete-event simulator. Each agent and behavioral object are independent processes (Java threads) that process events using their own belief-based inference and activity subsumption engine (2). A Brahms agent is a *belief-desire-intention* (BDI) agent (32). Each agent has a private set of beliefs that are used to infer new beliefs (in *thoughtframes* and *workframes*) and perform activities (in *workframes*). All creation of new beliefs and facts, as well as the start and end of activities are new events for the agent that need to be scheduled on the agent's local event

queue. There is a centralized scheduler that schedules the distribution of events to and from agents (in case of agent-to-agent communication and world fact creation), based on a discrete simulation clock. Although the grain-size of the simulation clock can be set by the user, by default one clock-tick is assumed to be one second of simulated time. The centralized scheduler makes sure that all events generated by all agents are processed in the right order, making sure that each agent processes the correct events in each event-cycle. See Figure 12.6 in section 12.5.2 for a depiction of the central event scheduler and its interaction with a Brahms agent and the central world state.

2. *Real-Time or RT mode*: In this mode the BVM's central scheduler is bypassed and the simulation clock is turned off. Each agent and behavioral object are now executing in real-time. Since they are all independent Java-threads, they are all independently executed parallel processes. All incoming events are processed as fast as the CPU schedules the priority of each agent Java-thread in the Java VM. Since there is no centralized event scheduler, and no simulation clock, activity execution time is managed by the system clock.

3. *Distributed RT mode*: This mode is equal to the RT mode, with the additional capability of allowing multiple BVMs to connect to each other over a network. Agents and behavioral objects running in different BVMs can find each other using a distributed directory service. Communication between agents is managed using a hidden communication layer. Different communication protocols are supported; TCP/IP, UDP, SSL, SOAP, Corba, etc. This allows a Brahms program to be distributed over a network. For a more detailed description of the DRT model we refer the reader to (33).

To explain the Brahms language concepts, in the next subsections we use the example of a student whom gets hungry while studying and needs to go to an ATM machine to get cash to pay for lunch.

13.5.2 Modeling people and organization

Modeling of people and organizations is done with the language concepts *agent* and *group*. Imagine that all students, while they are studying, monitor how hungry they are. The level of "hungriness" is measured by a belief about a real-valued attribute. Let's assume that all students will study until their hungriness level rises to 21. At that moment, the student determines that he or she needs cash to go to lunch. This will make the student stop the "study" activity.

The program source code in Table 13.1 shows the Brahms code for a model of the above description. First, we see the group *Student* declared. The group has two *attributes*, *howHungry* and *needCash*; the first one being of type *double* and the second of type *boolean*. Then, there is one *relation* called *hasCash* defined. This relation makes it possible for a student to have cash on him or her. Cash is represented by a class-type *Cash* (see next subsection).

Next in Table 13.1, is the definition of every student's *initial beliefs*. For each agent that is a member of the group *Student*, initial beliefs are inherited from the group and are created at initialization time and added to the agent its local belief-set. In the source code below agent *Alex_Agent* is a member of the group *Student* and the belief *(Alex_Agent.needCash = false)* is created at initialization (i.e. sim time = 0). This means that at initialization time agent *Alex_Agent* will believe that it does not need any cash.

Next is the definition of the activity *study*. In Table 13.1 you can see that activity *study* is defined as a *primitive activity* with a *max duration* of 3000 clock ticks, or by default, five minutes of simulated time. This means that when an agent executes this activity, the simulation engine makes sure that the activity takes five minutes. How long the activity actually takes during a simulation is undefined, and is dependent on the priority of other active activities (see Chapter 12.5 for a more detailed discussion on activity scheduling and execution). Important to note is that a primitive activity only takes an amount of time. What happens while the agent is in a primitive activity is not further specified. In this case, the agent is simply studying for five minutes at a time.

The last, but most important piece of code in Table 13.1 is the definition of the *workframe* (WFR). WFR *wf_study* defines when the agent starts and stops executing the *study* activity. The *when-clause* defines the *preconditions* for the start of the WFR. In the example in Table 13.1 the preconditions state that the agent must belief that the time of the Campanile clock is less than 20, that it does not need any cash, and that the hungriness level is below 21. Only then will the WFR *wf_study* be executed and the *body* or *do-part* of the WFR performed. Here only the *study* activity will be performed. After the activity is complete the WFR is done. However, since the WFR *repeat variable* equals *true*, as long as the agent's beliefs will match that of the preconditions the agent will keep executing an instance of the WFR. Thus, the agent will repeatedly keep studying for five minutes, until at any moment in time the agent *detects* the *fact* that it needs cash. The *dt_veryHungry detectable* is active while the WFR is active, and at any moment the agent detects the fact that it needs cash, or gets the belief that it needs cash through some other means (e.g. reasoning, or communication with another agent), it will *abort* the current *study* activity. Because one of the preconditions is then also false—the agent now beliefs it needs cash—no new *wf_study* will get executed. This is shown in Figure 13.3 where one can see other WFRs interrupting the WFR *wf_study*.

The last line in Table 13.1 defines agent *Alex_Agent* as a member of the group *Student*, which means it inherits all concepts defined in that group. It should be noted that an agent can be a member of any number of groups, as well as that groups can be members of parent groups. In this way, we can model any organization by modeling the organizational-, functional- and social groups and what people do when they are a member of these groups. By using group inheritance, the practice in an organization can be modeled by modeling the community of practice as groups with attributes, beliefs, activities, workframes and thoughtframes. People that belong to a community of practice are modeled as agents that are a member of the corresponding group.

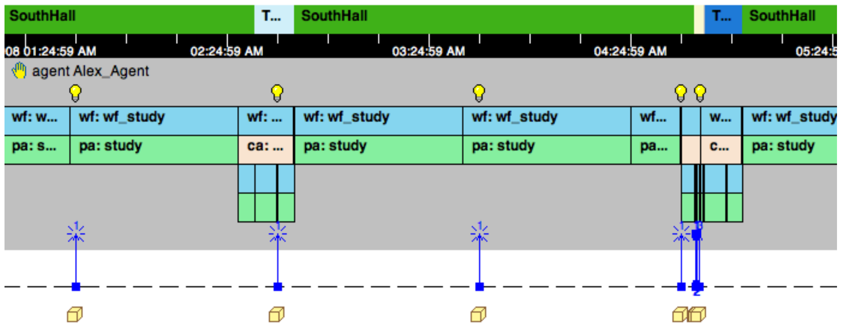


Fig. 13.3 AgentViewer screenshot of agent Alex_Agent execution

Figure 13.3 shows the execution of multiple *study* activities by agent *Alex_Agent*.

13.5.3 Modeling artifacts and data objects

As described in section 13.4.2.4 a model of work practice includes the artifacts and data objects used in the context of an activity. In the student model described in Table 13.1 we can see a number of artifacts being used; The object *Campanile_Clock*, which is the representation of the Campanile bell tower on the UC Berkley campus and the Cash object of the student, representing the money the student has in his or her pocket. In the complete model there are other objects well, such as the ATM machines of the bank, the student's ATM card as artifacts, as well as the student's bank account as a data object with a pin code, etc. Table 13.2 shows the source code of the classes for the ATM machines and and bank account, as well as some of the object instances. It shows that the Brahms language includes the concepts of *classes* and *objects*, besides that of agents and groups. Brahms is thus not only agent-based, but also object-oriented.

Important to note are the *initial_facts* defined for the *Alex_Account* object in Table 13.2. This object is a data object and the initial facts are created at object creation time. These represent the facts in the world that exist about student Alex' bank account. Agents can have beliefs about those attributes that differ from the facts in the world. This is how agent *Alex_Agent* can have a wrong belief about his bank account's pin code, and can lose his ATM card after typing in a wrong pin code three times at the ATM.

In Brahms, the world facts are separate from the agent's beliefs about those facts. There is only one world fact set. Agents can detect these facts using *detectables*, as the one shown in the WFR in Table 13.1.

13.5.4 Modeling communication

Communication in Brahms is modeled as speech act or communicative act activities (29; 16). The Brahms language includes default *communication* and *broadcast*

```
group Student {
    attributes:
        public double howHungry;
        public boolean needCash;

    relations:
        public Cash hasCash;

    initial_beliefs:
        (current.needCash = false);

    activities:
        primitive_activity study() {
            //equals 5 mins of simulated time
            max_duration: 3000;
        }//end study

    workframes:
        workframe wf_study {
            repeat: true;
            detectables:
                detectable dt_veryHungry {
                   when(whenever)
                     detect((current.needCash = true), dc:100)
                   then abort;
                }//end dt_veryHungry
            when (knownval(Campanile_Clock.time < 20) and
                knownval(current.needCash = false) and
                knownval(current.howHungry < 21))
            do {
                study();
            }//end do
        }//end wf_study
}//end Student

agent Alex_Agent memberof Student { }
```

Table 13.1 Partial source code for Student group and agents

```
class Atm {
    display: ''Atm'';
    cost: 0.0;
    resource: true;
    attributes:
        public int currentAccountPin;
        public boolean pinChecked;
        public boolean pinIsWrong;
        public boolean pinAsked;
        ...
    relations:
        public Bank ownedbyBank;
    ...
}//end Atm

object Boa_Atm instanceof Atm  {
    location: Telegraph_Av_113;
    initial_facts:
        (current ownedbyBank Boa_Bank);
}//end Boa_Atm

class Account {
    display: ''Account'';
    cost: 0.0;
    resource: true;
    attributes:
        public double balance;
        public string typeof;
        public int code;
        public int pin;
    relations:
        public Bank openedWithBank;
}//end Account

object Alex_Account instanceof Account {
    display: "Alex_Account";
    initial_facts:
        (current.balance = 20.00);
        (current.typeof = checking);
        (current.code = 1212);
        (current.pin = 1111);
        (current openedWithBank Boa_Bank);
}//end Alex_Account
```

Table 13.2 Partial source code for classes and objects

activities. In a *communication* activity, the performing agent specifies to which agents or objects it is communicating. In a *broadcast* activity, the performing agent does not have to specify the receivers of the communication. In that case all agents in the same location (see section 13.5.5) as the performing agent will receive the communication. What is being communicated is the beliefs specified in the, what is called, *transfer definitions* defined in the activity. It should be noted that an agent needs to have the beliefs in its belief-set for it to be able to communicate them to another agent or object. This is to represent that we, as people, cannot communicate that what we don't know.

Table 13.3 shows the definition of communication activity *communicatePIN* and use of this activity in the WFR *wf_communicatePIN*, all part of the *composite activity* called *useATM*. In this composite activity we model how the student uses an ATM. Figure 13.3 shows the communication of student *Alex_Agent* with the object *Boa_Atm*, showing the communication as lines from the agent to the object on the right in the figure. For a more detailed description of communications using the Brahms FIPA-based Communicator library, we refer the reader to Sierhuis, et al (33).

13.5.5 Modeling location and movement

In Brahms agents and objects can be situated in a model of the physical world. The world is represented independent of the capability of agents. An *areadefinition* is used for defining a class of *area* instances, used for representing geographical locations. Area definitions are similar to classes in their use. Examples of area definitions are "Building", and "City". An example of an area is "Berkeley". Areas can be decomposed into sub-areas. For example, a building can be decomposed into one or more floors. A floor can be decomposed into offices. The decomposition can be modeled using the PART-OF relationship. A *path* connects two areas and represents a route that can be taken by an agent or object to travel from one area to another. The modeler may specify distance as the time it takes to move from area1 to area2 via the path. The BVM automatically generates location facts and beliefs for agents and objects moving from one area to another.

Agents and objects can be located in an *initial location* (i.e. an area). Agents and objects can move to and from areas using a *move* activity. For example, Figure 13.3 shows the movement of agent *Alex_Agent* from the areas *SouthHall* to the areas of the restaurant and bank branch on Telegraph Avenue, and back to SouthHall. When agents and/or objects come into a location, the BVM automatically creates a location fact *(agent.location = <current-area>)*. Agents always know where they are and they notice other agents and objects. When agents come into a location, the BVM automatically gives the agent a belief about its new location (same as the location fact), and also gives the agent a location belief for all other agents and objects currently in that location. When an agent or object leaves a location, the location fact and beliefs are retracted from all agents that are in that location the moment the agent or object leaves. Agents and objects can carry (through the *containment* relation) other agents and objects. Contained agents and objects are not "noticed" until they are put into the area by the containing agent or object.

```
group Student {
    ...
    activities:
    ...
        composite_activity useATM() {
            activities:
                ...
                communicate communicatePIN(Atm at3, Account bka) {
                    max_duration: 20;
                    with: at3;
                    about: send(bka.pin = unknown);
                    when: end;
                }//end communicatePIN
                ...
            workframes:
                ...
                workframe wf_communicatePIN {
                    repeat: true;
                    variables:
                        forone(Account) bka;
                        forone(BankCard) bkc3;
                        forone(Atm) at3;
                        forone(Bank) ba3;
                        forone(Building) bd3;
                    when(knownval(current hasBankCard bkc3) and
                        not(current contains bkc3) and
                        knownval(current hasAccount bka) and
                        knownval(current.chosenBank = ba3) and
                        knownval(at3 ownedbyBank ba3) and
                        knownval(current.pinCommunicated = false) and
                        knownval(current.location = at3.location) and
                        knownval(at3 contains bkc3))
                    do {
                        communicatePIN(at3, bka);
                        conclude((current.pinCommunicated = true),
                        bc:100, fc:0);
                    }//end do
                }//end wf_communicatePIN
                ...
            }//end communicatePIN
            ...
        }//end useATM
        ...
}//end Student
```

Table 13.3 Partial source code for communication of beliefs

```
// Area defintions
areadef University extends BaseAreaDef { }
areadef UniversityHall extends Building { }
areadef BankBranch extends Building { }
areadef Restaurant extends Building { }

// ATM World
area AtmGeography instanceof World { }

// Berkeley
area Berkeley instanceof City partof AtmGeography { }

// inside Berkeley
area UCB instanceof University partof Berkeley { }
area SouthHall instanceof UniversityHall partof UCB { }
area Telegraph_Av_113 instanceof BankBranch partof Berkeley { }
area Telegraph_Av_2405 instanceof Restaurant partof Berkeley { }

// initial location
agent Alex_Agent memberof Student {
    location: SouthHall;
}//end Alex_Agent
```

Table 13.4 Brahms Geography Model

The geography model is a conceptual model—it does not represent the geography as a graphical three-dimentional model—representing geography as a hierarchical model of areas, with attributes representing facts about these areas and agents and objects as inhabitants of these areas. Areas can have attributes and relations, and define initial facts. Facts about areas can represent the state of a location, e.g. the temperature in an area. The BVM automatically generates facts about the 'partof' relationships in the geography. Agents can detect these facts and thus learn (i.e. get beliefs) about the areas in their environment.

The example geography model in Table 13.4, defines a simple geography for the University of Berkeley in Berkeley, CA. This model defines the university building SouthHall, where student Alex is initially located. Furthermore, the model defines a bank branch and a restaurant in the city of Berkeley.

13.5.6 Java Integration

Brahms currently has two ways of interfacing with Java using the Brahms JAPI:

- **Java activities** are primitive activities written in Java. To write a Java activity the modeler needs to define the Java activity in the Brahms model, and implement the activity by writing the activity using the Brahms JAPI. To do this you need to create a Java class that extends from the *AbstractExternalActivity* abstract class in the JAPI. The AbstractExternalActivity is an interface for external activities implemented in Java, called by Brahms Java activities. The external activity can perform any Java action. This abstract class provides access to parameters passed to Brahms Java activities, and allows for adding bindings to unbound variables passed to Brahms java activities through parameters. Most importantly, you need to define the *doActivity* method to execute the Java activity.

- **External agents** are Brahms agents written in the Java programming language. To write an external agent you will need to define the agent as an external agent in the Brahms model and then write the external agent in Java using the Brahms JAPI. To do this you need to create a Java class that extends from the *AbstractExternalAgent* abstract class in the JAPI. The AbstractExternalAgent is an interface for external agents implemented in Java, loaded into the virtual machine to participate in a Brahms simulation or real-time agent execution. The external agent can perform any Java action. This abstract implementation provides access to the concepts loaded in the virtual machine and the world state to allow for communications with these concepts and to allow for world state changes to be triggered by this agent.

- **Java objects** can be referenced using Java class types as Brahms attribute types. This allows referencing Java objects from within the Brahms language.

13.6 SYSTEMS ENGINEERING: FROM SIMULATION TO IMPLEMENTATION

As we gained experience with applying our theory of modeling work practice using the Brahms environment, we developed a world view and a set of practices for the development of software systems. We collectively named these set of practices *human-centered computing*, in the sense that the golden rule is that people in a work system should always be *at the center* in the development of new technology. Over the years, we have developed a set of practices that we are now starting to formulate as a MAS software development methodology. At the center of this methodology is our work practice modeling and simulation approach outlined in this chapter. The development of software is rooted in the analysis, modeling and simulation of the work system in which the new software system is to be introduced. Due to the nature of the Brahms environment, an agent-based simulation model can be relatively easy converted to a real-time MAS. This fact has made it possible for us to first design a MAS as an agent-based simulation in Brahms, and subsequently convert this model to a MAS that can be executed in by the BVM in real-time. Hence, we are speaking of a methodology that allows us to go *from simulation to implementation* of a MAS. In this section we describe this methodology in more detail.

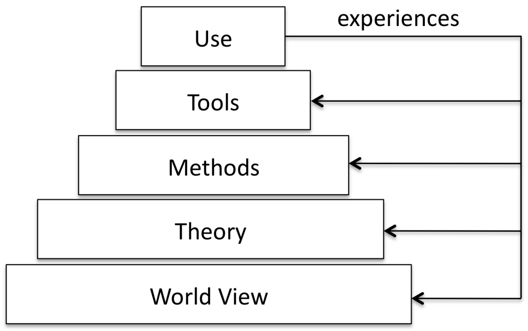


Fig. 13.4 The methodology pyramid

A methodology consists of a number of elements that build on each other like a pyramid (see Figure 13.4). Below we describe how each of these elements are filled in in our approach. As we apply our methodology to more software engineering projects, new experiences will surely make us revisit each of these elements and make changes to the approach. It is thus important to note that the description we provide here will not be the final version of our approach. One should see this as but the first attempt to formalize a set of practices we have developed over time, and applied with success at NASA. In section 13.7 we provide our recent experience with applying our own methodology in a project for NASA's Mission Control. Here we first provide a more formal description of our methodology, as applied.

- **World view:** Our world view is our human-centered slogan that people should alway be at the center of any technology development project. This sounds trivial, but is not the rule in most software development methodologies, or in any systems engineering approach for that matter. This human-centered slogan is the view on which our theory is based.
- **Theory:** The theory of our methodology is the combination of theories on which our theory is based, such as activity theory, situated cognition and participatory design, and of course our theory of modeling and simulating work practice described in this chapter.
- **Methods:** The methods we apply are those that come from anthropology and knowledge engineering (e.g. participant observation, video and photography, interviews and knowledge acquisition), as well as those coming from participatory design and cognitive psychology (e.g. story boarding and task analysis). Our main method is that of work practice modeling and simulation using the Brahms agent-based activity approach, and the way we develop different models over the course of the project. We start with a model of the *current work system*, then move to a participatory design of the MAS in Brahms and incorporated this in a *model of the future work system*. The future work system model is then turned into the MAS.

- **Tools:** Besides the obvious tools of digital photo and video cameras, recorders, etc, our main tool is the Brahms environment for developing agent-based models of the work practice and the MAS.

- **Use:** Our methodology has been used in different ways over the last ten years at NASA. Every time we use the methodology on a project, we gain new understanding and experiences which we use to update our theories, methods and tools.

13.6.1 A cyclic approach

The *from simulation to implementation* methodology uses a cyclic approach, as shown in Figure 13.5. The project starts with doing participatory observation of the current work system. This activity can take up to a couple of months, depending on the objective of the project. The next step is to model and computationally simulate the current work system in a *descriptive* Brahms agent-based model. Even though Figure 13.5 shows these two activities to be sequential, it should be understood that in reality these activities happen in parallel. It is the agent-based modeling of the work system that drives what parts of the work practice needs more observation.

After the simulation of the current work system is completed, the next activity is to design the work system change. This design activity needs to be participatory with the workers from the organizations in the work system. Together the MAS system is designed given the constraints of the possible changes. The new design is then implemented in a Brahms MAS model and simulated. This MAS is incorporated in a simulation of the future work system that is changed from the current work system model. This way the impact of the newly designed MAS on the future work system can be analyzed. Also, changes to the current work system can be identified and analyzed, and the same metrics as the ones generated by the current simulation can be compared with those of the future simulation.

After the future work system simulation is completed, the next activity is to implement the MAS in a real-time environment. Using the Brahms environment, the Brahms model of the designed MAS can be easily changed to a real-time executable MAS. It is in this activity that the “from simulation to implementation” transition takes place. Ethnographers, system designers and Brahms modelers, are replaced by Brahms and Java programmers.

13.6.1.1 What to include and when to stop The cycle is complete after implementation of the MAS in the work system. Continuous improvement to the work system is obtained by starting the next version of the cycle. The model of the future work system in the previous cycle now becomes the model of the current work system in the next cycle. However, because in the previous cycle the model was a *prescriptive model* of the future work system, new work practice observations and changes to the future model will result in a *descriptive model* of the current work system in the next cycle.

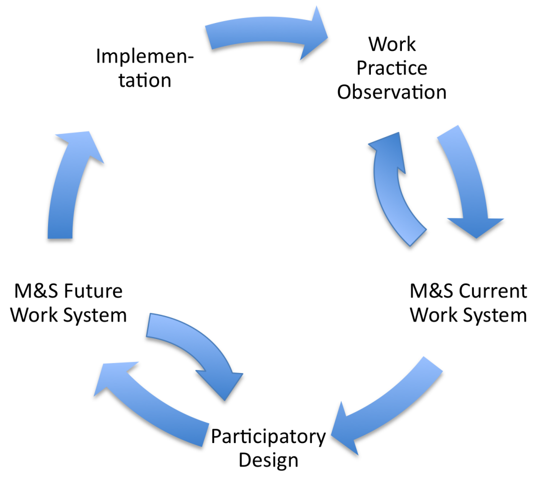


Fig. 13.5 Simulation to Implementation Cycle

13.6.2 Modeling current operations

As for any modeling and simulation project, it is very important to define the objectives of the modeling effort up front. This should be done in collaboration with management and end users. The end product of modeling and simulation of the *Current Work System* is an agent-based model of a "day in the life" of the people in the work system. The output of the simulation should be a detailed activity timeline of all relevant roles and people in the organization, as well as the generation of metrics and statistics that answer the specific questions that are being asked.

13.6.2.1 Developing a day-in-the-life model A day-in-the-life (DITL) model is a model of the chronological activities of people. It is developed based on observation in the workplace where one can see what is happening minute to minute as the workday progresses. The trick is to observe without making immediate assumptions. The only form of interpretation the observer does is to abstract moment by moment action into high-level activities that can be identified to have a defined beginning and end. For example, leaving home to go to work (the "going to work" activity), coming in to work and leaving work to go home (this is the highest level "working" activity), the start and end of a meeting, the reading of email, the answering of a telephone call, etc.

Once the observer has identified an abstract model of the chronological activities of the day of a person that plays a particular role, the next step is to model these activities as an activity model for the agent representing that person. It is important to observe not just one person playing that role, but observe a number of different people playing the same role, as to develop not a person specific model, but an activity model for that role in the organization. Once a role is modeled as one or more groups in Brahms, as many agents being members of these groups can be

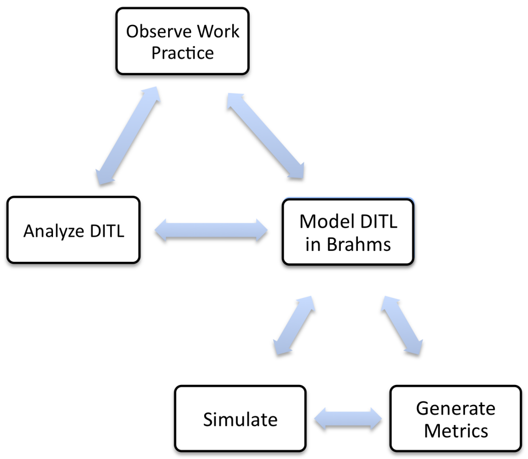


Fig. 13.6 Current operations modeling process

created in the simulation. Figure 13.6 shows the *observation-analyze-model* cycle of the development of a DITL model for the current operations of a work system.

Other important aspects to observe, analyze and model are communications between people and how this communication takes place, be it fact-to-face, using phone, e-mail, paper or electronic documents, etc. The use of space and artifacts in the workplace and the movement within the environment—e.g. using a USB stick to transfer files from a computer on ones desk to another computer in another place. All these objects, information creation and transfer need to be modeled in the DITL model, as well as movement of agents within geography model.

13.6.2.2 Simulating a day-in-the-life-model After a version of the DITL is created in Brahms, it is important to make sure the model can be simulated and results can be shown to the end-user. Statistics and metrics generation need to be incorporated in the model from the beginning, such that the why- and how- questions can be answered with the simulation. What statistics to keep track off is an important decision in any effort.

The bottom of Figure 13.6 shows this *model-simulate-generate* cycle. The top- and bottom cycles are working in concert, and the development of a current operations model is based on cyclical phases of more and more detail in the DITL. The issue of when to stop going through this cycle is an important one. At the start of any modeling and simulation effort the objective of the effort needs to be clearly defined. The objective should be formulated in terms of what needs to be learned from the output of the simulation, e.g. "What we are interested in is understanding how task X is performed by the workers in group Y in terms of who is doing what, when, what knowledge is used and how long it takes. From this we need to design an automated system that can do this task."

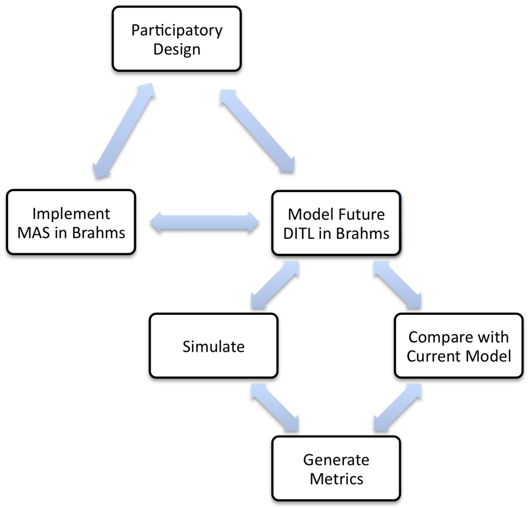


Fig. 13.7 Future operations modeling process

13.6.3 Modeling future operations

After the current operations modeling phase is finished, and the decision is made to change the current work system and/or create some automation that will be implemented in the current work system—thereby also changing the work system—the next phase in the simulation to implementation cycle is the modeling and simulation of the future operations.

Figure 13.7 depicts the future operations modeling process. We start with a participatory design task, where the new automation is designed in close participation with the end-users of the system. It cannot be stressed enough how important the participatory approach is to this effort. Participatory design in this context means that the design team includes workers from the organization in which the system is going to be used. Together system and user interface requirements should be developed.

When this is done, the design of the new system is implemented as a MAS in Brahms. This implementation is at first a simulation of the behavior of the system. The simulation of this future MAS system should use as input the same input as was used in the current operations simulation, and generate the same output. This is accomplished by developing a DITL simulation of the future operations that includes a simulation of the MAS, based on the simulation of the current system. Especially, the people—end users—in the future work system need to be included as agents in the future DITL simulation model. Those aspects of the practice that needs to change need to be modeled in this future model. Similar statistics as in the current operations simulation model will need to be generated by this future operations simulation model. This will allow later on for the comparison between the current and future operations model.

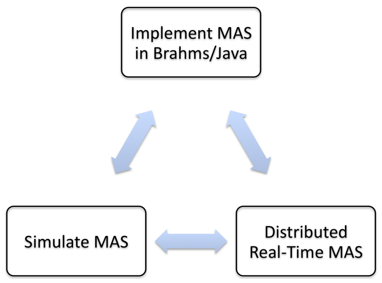


Fig. 13.8 Moving from simulation to implementation of the MAS

The trick in this phase is to model the future MAS as close to complete as possible. This will not only help in the transition from simulation to implementation of MAS in the next phase, it also helps in prototyping the system in the participatory design process. Using the capability in the Brahms language to develop Java agents, designed user interface agents can already be implemented as part of the future simulation. This will allows for a close to realistic simulation. The simulation needs to be verified and validated participatory with the workers form the organization.

13.6.4 MAS implementation

The last step in the from simulation to implementation cycle is the transformation of the MAS as a simulation in Brahms, to a distributed real-time MAS that implements the designed automation for the future work system. This transition process is depicted in Figure 13.8.

Depending on the needed interfaces of the system, this transition can be easy and fairly fast as long as the MAS simulation was done in detail. However, most of the time the MAS needs to be integrated with existing legacy systems and networks in the organization. The integration with existing external systems and networks will have been abstracted and simulated in the future operations model that includes a simulation of the MAS. The main effort of the transition from a MAS simulation to an actual executable MAS in the work system is the development of external system and network interfaces. For example in the future simulation of the OCAMS system, described in the next section, the FTP-ing of files performed by an FTP agent was modeled as a simulation of the FTP of file objects on a simulated file system. In the MAS simulation model the file system was modeled as Brahms areas where file objects were inhabitants of that area. File FTP was simulated as a movement of file objects from one area to another. When the MAS system was implemented, this FTP agent became a Java agent that implemented the actual FTP transfer of files on over the network system.

When using appropriate agent communication protocols in the simulation, a simulated agent can easily be replaced by a real-time agent, making the transition from simulation to implementation rather seamless. When done correctly, the simulation of the MAS will have allowed for unit and some integration testing depending on the number of simulated agents, leaving a more complete integration testing and final system testing. Thus, the simulation of the MAS and the future operations simulation not only help in the design validation of the system and determining the impact on the current work system, it also cuts down in the development and testing of the actual MAS.

The Brahms environment makes this easy transition from simulation to implementation possible, because of its ability to both run the BVM in simulation mode and in (distributed) real time mode. Besides this, the Brahms language is closely related to Java and it is very easy to implement agents or activities with the existing Java API. As far as we know this is an unique feature that no other agent simulation language or agent-oriented language possesses.

13.7 A CASE STUDY: THE OCA MIRRORING SYSTEM

The Lyndon B. Johnson Space Center ("JSC") near Houston, Texas, is the National Aeronautics and Space Administrations (NASA) center for human spaceflight activities. JSC houses the Shuttle and ISS Mission Control Centers (MCC), one of the most complex and best-known organizations for command and control of human space flight. The ISS MCC is charged with managing, commanding and controlling every aspect of the ISS, from docking with the Space Shuttle and Soyuz spacecrafts to uplinking and downlinking all information to and from the ISS. The ISS MCC, as most MCCs, is divided into a "front room"—the room where the main flight controllers are located—and a "back room"—where the support flight controllers are located. Each flight controller in the front room has several support flight controllers in the back room. Together the people in the front- and back room are organized in flight control groups for the different subsystems for the ISS, overall named the Flight Control Team (FCT).

Over a six-month period, computer scientists and ethnographers in the Work Systems Design & Evaluation Group of the Intelligent Systems Division at NASA Ames observed, studied and simulated the work practices of the Orbital Communications Adapter (OCA) Officer group to identify possible process improvements. Using statistics generated from a Brahms agent-based current operations model, the team, over the next three-month period, designed and simulated a Brahms agent-based workflow system that now automates the process of creating a ground-based replica of the ISS file system (the Mirror LAN). Future operations simulation statistics predicted a reduction in mirroring time from 6% to 0.4% of the OCA Officer's shift—a more than ten time reduction. Using the above described simulation to implementation methodology, agents were then converted, in a one-month period, into a distributed MAS tool called OCAMS. Using three distributed Brahms BVMs, these agents manage the workflow on multiple computers and servers using secure communications provided by the Brahms collaborative infrastructure (CI). The tool automatically writes large parts of the OCA Handover Log.

13.7.1 Mission Control as a socio-technical work system

The main task of the MCC is to manage human space flight, from liftoff to landing at the end of a mission. The different groups of flight controllers manage all aspects of the mission. In case of the ISS this means monitoring and commanding all aspects of the ISS, as well as planning, scheduling and managing the daily activities of the astronauts onboard the ISS. Besides the ISS MCC, the Shuttle MCC is housed by the same JSC organization (the Mission Operations Directored) in the same building complex. This large organization is divided into divisions, branches and groups with a total of around 3200 people. Most systems and subsystems of the ISS and Shuttle are managed by separate flight control groups. These groups consists of both front room and back room positions for a specific subsystem, such as "Guidance & Control." Besides having general MCC software tools available that every flight controller group uses, each group is responsible for developing their own software tools to manage their subsystem and training each other to use these tools. Each group is thus a separate organization of flight controllers, trainers and software developers. Most senior flight controllers do it all; they help train their younger colleagues, set requirements for tools and tool improvements, as well as "sit on console"—are in the flight control room working a shift as a flight controller.

There is a big difference between the Shuttle MCC and the ISS MCC; Whereas the Shuttle only flies a couple of times a year for a mission of around 14 days, the ISS is manned 24x7, continuously 12 months a year. To do this there are three shifts—called orbits—in the ISS MCC. Certified flight controllers are the only ones that can sit on console, and most who are certified will be on console one week for a particular shift. Then they are "off console" for a week, which means that they are in their offices working on their next certification level (i.e. training), or are working on developing their next generation tools and procedures.

While on console, the flight controller is solely responsible for his or her task for which he or she is certified. Depending on the stage of a mission—an ISS mission, called an Expedition, takes about six months—the work on console can be highly stressful, or very quiet. Besides the upfront known stage of a mission, a "day in life of a flight controller" can never be predicted with certainty, because at any moment in time something can happen that requires their utmost attention, flexibility, and inventiveness. It is this excitement that most flight controllers like. Because of this high-stress environment, most flight controllers are young engineers.

The MCC is both a work system that manages and controls two of the most complex systems in the world (both the ISS and the Shuttle), as well as a work system that deals with both the most advanced software- and computer systems (dedicated AI tools running on Linux systems) and the most old fashioned software- and computer systems (old mail servers running on Windows 2000 machines with 512KB memory). Indeed, the MCC is a hodgepodge of software- and computer systems. Flight controllers need to be able to live within this socio-technical work system, which often means that the young flight controllers are very inventive and flexible, seemingly working miracles with the tools and system that exist within their work environment.

13.7.2 The OCA officer's work system

The Orbital Communications Adapter (OCA) Officer is a back room flight controller and a member of the Operations Planner (OPSPLAN) group. The ISS OCA Officer is responsible for manually uplinking and downlinking all files to and from the ISS. These files include schedules, procedures, commands, email, photographs, health data, newspapers, etc.

13.7.3 Simulating the current OCA work system

As part of the current OCA work process, the OCA officer spends time after each file uplink/downlink activity mirroring the same files to the Mirror LAN. This cumbersome activity is both error prone, because of the manual "sneaker net" being used and time consuming, because many uplink/downlink activities have to be duplicated on the Mirror LAN using a laborious manual process. Figure 13.9 is a process flow depiction of the current mirroring process.

13.7.3.1 The mirroring process After the OCA Officer has uplinked to or downlinked files from the ISS he or she needs to create a mirror of the files on what is called the Mirror LAN. The Mirror LAN network is a ground-based duplicate of the network onboard the ISS. This mirroring activity starts with the OCA Officer—after the files have been FTP-ed from the machine on which the files are downlinked or uplinked to the archive server—deciding which of the files need to be mirrored. Not every file uplinked or downlinked needs to be mirrored to the Mirror LAN. After FTP-ing the file from the uplink/downlnk computer, the OCA Officer moves in his or her chair back to the MAS computer from which they copy the files from the archive server via their workstation to a USB stick. After the files are copied to the USB stick the OCA Officer moves in his or her chair to a space station computer (an IBM laptop) connected to the Mirror LAN. The OCA Officer copies the files from the USB stick to folders on the MIrror LAN. Some of the files are processed by special batch files that run on the Mirror LAN. For these files the OCA Officer monitors the batch process, which can take up to 15 minutes, to watch for possible errors that can occur. In case there are errors the error files need to be send to the flight controller responsible. This is done via e-mail on the MAS PC. After all files have been mirrored the OCA Officer logs all actions and problems in a Word document called the handover log.

13.7.3.2 Simulating the mirroring process The current OCA Mirroring work process was modeled in Brahms, based on our three month observation with two OCA work practice observers. We took pictures, videos and written notes. Our modeling data was based on timing of OCA Officer activities during their actual work.

Figure 13.10 shows the UML Class diagram generated from the Brahms model (the Brahms compiler can generate standard UML interface—umi—format from Brahms source code). The Brahms group inheritance in Figure 13.10 shows the

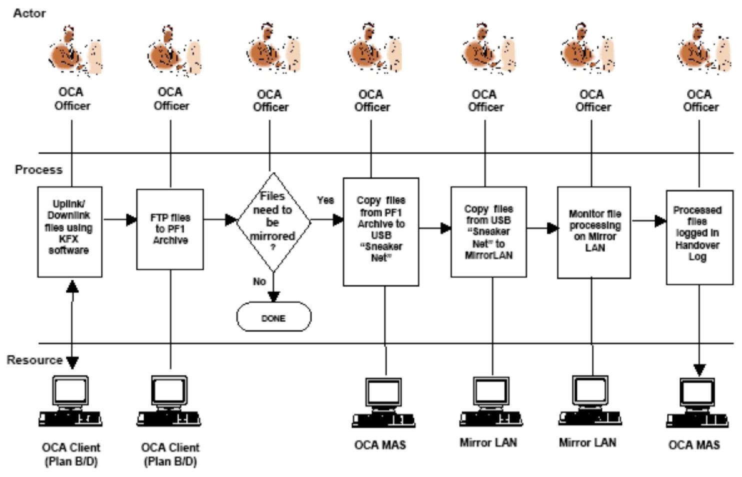

Actor
OCA Officer
OCA Officer
OCA Officer
OCA Officer
OCA Officer
OCA Officer
OCA Officer
Process
Uplink/ Downlink files using KFX software
FTP files to PF1 Archive
Files need to be mirrored ?
Yes
No
DONE
Copy files from PF1 Archive to USB "Sneaker Net"
Copy files from USB "Sneaker Net" to MirrorLAN
Monitor file processing on Mirror LAN
Processed files logged in Handover Log
Resource
OCA Client (Plan B/D)
OCA Client (Plan B/D)
OCA MAS
Mirror LAN
Mirror LAN
OCA MAS

Fig. 13.9 Current OCA Mirroring Work Process

organizational model of the MOD organization of the OCA Officers group. You can see that the OCA Officers group is part of the DO4 Flight Planning Branch, which in turn falls under the DO Operations Division.

Figure 13.10 also shows the activity-based groups that the OCA Officer group is a member of. Activity groups are abstract groups the modeler creates to define the common activities in the model. You can see the three activity groups in Figure 13.10 that are all members of the highest-level Schedule group. The OCA Mirroring group is the group in which the Mirroring Activity is modeled. Figure 13.10 shows the activities and attributes inherited from each group.

Figure 13.11 shows the sub-activities of the composite Mirroring Activity in the *Agent Model*. The OCA agents inherit this activity from the OCAMirroringGroup also shown in the UML Class diagram of Figure 13.10 . This activity describes what the OCA does to mirror a file. On the right side of Figure 13.12 one can see part of the *Geography Model*, while one the left hand side the *Object Model*. The Geography Model shows the ISS OCA room areas in the Planning MPSR room, where the OCA Officer and computers are located. The OCA Officer agents can move between room and room areas according to the paths specified. The Object Model on the left shows only a part of the object model. Together these three parts of the Brahms model constitute the OCA current operations model.

Figure 13.13 shows a part of the simulated OCA Officer activities timeline for Orbit 3 (3rd shift). On the right top (1) you can see agent OCA Orbit 3 executing the Mirroring activity with its subactivities. Below that (2), the timeline also shows the actual file object being moved (due to agent OCA Orbit 3's copying subactivities) from folder "V:" to a folder on the Mirror LAN, via a USB "stick" (the colored bars above the black timeline shows location movements of the OCA Officer agent (1) and file object (2)).

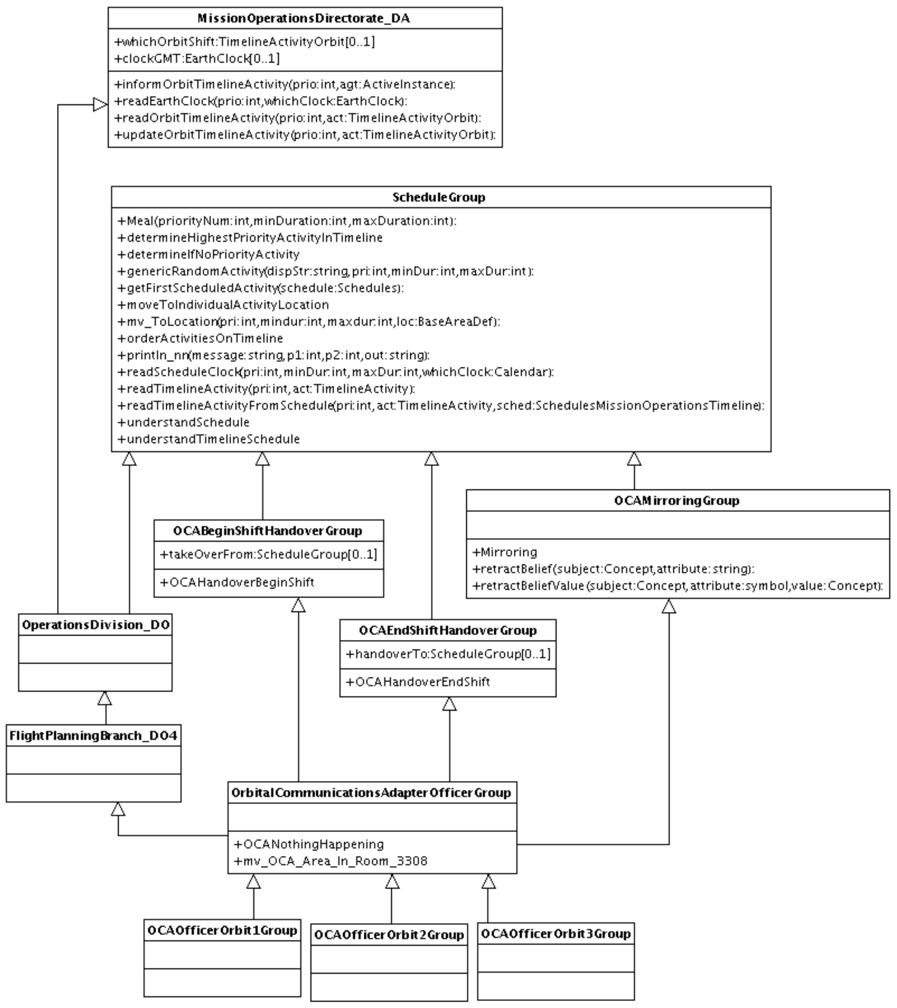
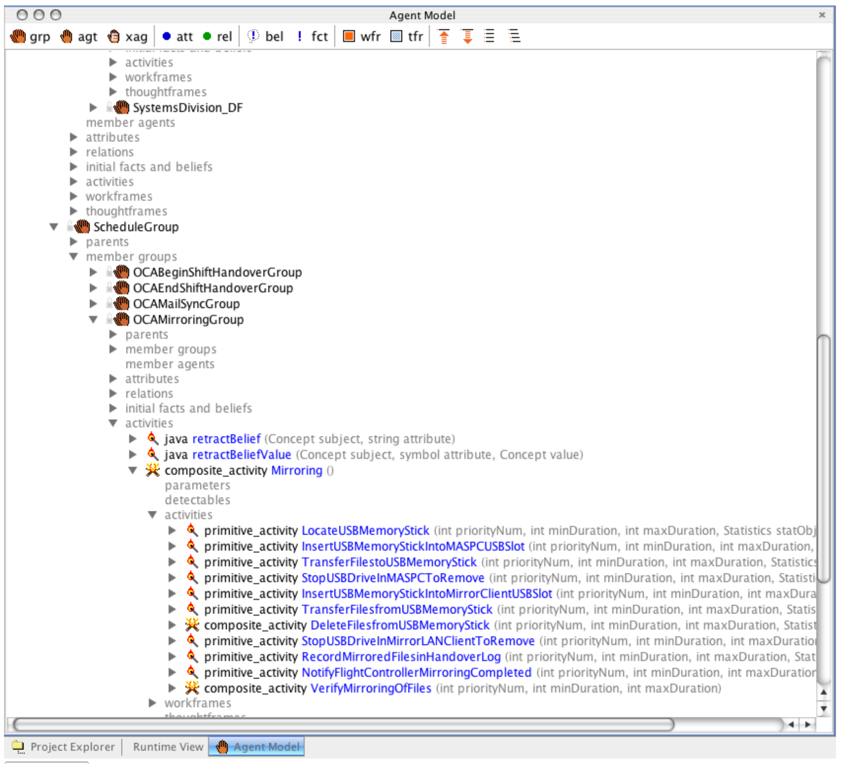
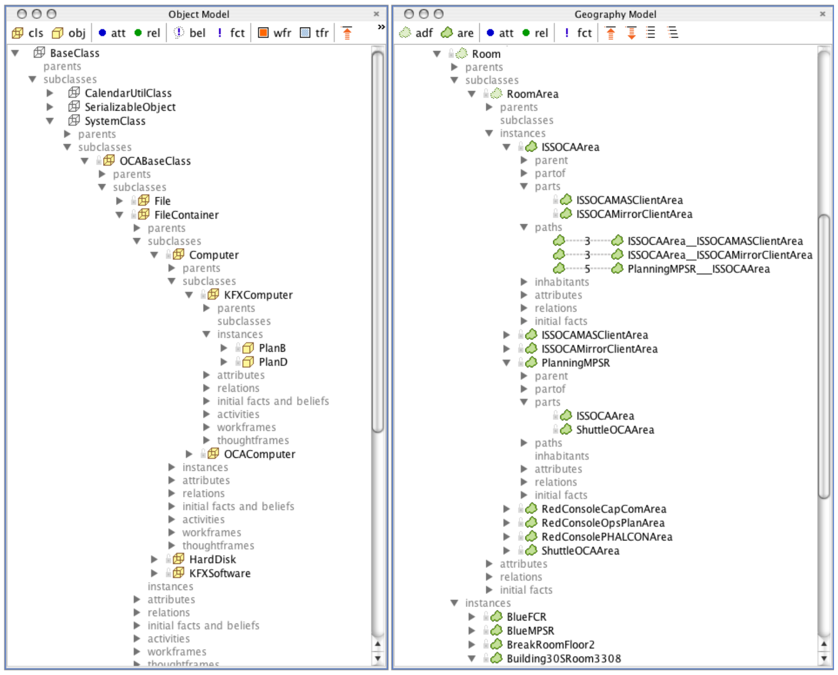

MissionOperationsDirectorate_DA
+whichOrbitShift:TimelineActivityOrbit[0..1]
+clockGMT:EarthClock[0..1]
+informOrbitTimelineActivity(prio:int,agt:ActiveInstance):
+readEarthClock(prio:int,whichClock:EarthClock):
+readOrbitTimelineActivity(prio:int,act:TimelineActivityOrbit):
+updateOrbitTimelineActivity(prio:int,act:TimelineActivityOrbit):
ScheduleGroup
+Meal(priorityNum:int,minDuration:int,maxDuration:int):
+determineHighestPriorityActivityInTimeline
+determineIfNoPriorityActivity
+genericRandomActivity(dispStr:string,pri:int,minDur:int,maxDur:int):
+getFirstScheduledActivity(schedule:Schedules):
+moveToIndividualActivityLocation
+mv_ToLocation(pri:int,mindur:int,maxdur:int,loc:BaseAreaDef):
+orderActivitiesOnTimeline
+println_nn(message:string,p1:int,p2:int,out:string):
+readScheduleClock(pri:int,minDur:int,maxDur:int,whichClock:Calendar):
+readTimelineActivity(pri:int,act:TimelineActivity):
+readTimelineActivityFromSchedule(pri:int,act:TimelineActivity,sched:SchedulesMissionOperationsTimeline):
+understandSchedule
+understandTimelineSchedule
OCAMirroringGroup
+Mirroring
+retractBelief(subject:Concept,attribute:string):
+retractBeliefValue(subject:Concept,attribute:symbol,value:Concept):
OCABeginShiftHandoverGroup
+takeOverFrom:ScheduleGroup[0..1]
+OCAHandoverBeginShift
OperationsDivision_DO
OCAEndShiftHandoverGroup
+handoverTo:ScheduleGroup[0..1]
+OCAHandoverEndShift
FlightPlanningBranch_DO4
OrbitalCommunicationsAdapterOfficerGroup
+OCANothingHappening
+mv_OCA_Area_In_Room_3308
OCAOfficerOrbit1Group
OCAOfficerOrbit2Group
OCAOfficerOrbit3Group

Fig. 13.10 Current operations sim statistics generated for each OCA Officer on every shift over a simulated month of work (time in minutes)



```
        ▶ activities
        ▶ workframes
        ▶ thoughtframes
      ▶ SystemsDivision_DF
    member agents
  ▶ attributes
  ▶ relations
  ▶ initial facts and beliefs
  ▶ activities
  ▶ workframes
  ▶ thoughtframes
▼ ScheduleGroup
  ▶ parents
  ▼ member groups
      ▶ OCABeginShiftHandoverGroup
      ▶ OCAEndShiftHandoverGroup
      ▶ OCAMailSyncGroup
      ▼ OCAMirroringGroup
        ▶ parents
        ▶ member groups
          member agents
        ▶ attributes
        ▶ relations
        ▶ initial facts and beliefs
        ▼ activities
            ▶ java retractBelief (Concept subject, string attribute)
            ▶ java retractBeliefValue (Concept subject, symbol attribute, Concept value)
            ▼ composite_activity Mirroring ()
                parameters
                detectables
              ▼ activities
                  ▶ primitive_activity LocateUSBMemoryStick (int priorityNum, int minDuration, int maxDuration, Statistics statObj
                  ▶ primitive_activity InsertUSBMemoryStickIntoMASPCUSBSlot (int priorityNum, int minDuration, int maxDuration,
                  ▶ primitive_activity TransferFilestoUSBMemoryStick (int priorityNum, int minDuration, int maxDuration, Statistics
                  ▶ primitive_activity StopUSBDriveInMASPCToRemove (int priorityNum, int minDuration, int maxDuration, Statisti
                  ▶ primitive_activity InsertUSBMemoryStickIntoMirrorClientUSBSlot (int priorityNum, int minDuration, int maxDura
                  ▶ primitive_activity TransferFilesfromUSBMemoryStick (int priorityNum, int minDuration, int maxDuration, Statis
                  ▶ composite_activity DeleteFilesfromUSBMemoryStick (int priorityNum, int minDuration, int maxDuration, Statist
                  ▶ primitive_activity StopUSBDriveInMirrorLANClientToRemove (int priorityNum, int minDuration, int maxDuratio
                  ▶ primitive_activity RecordMirroredFilesinHandoverLog (int priorityNum, int minDuration, int maxDuration, Stat
                  ▶ primitive_activity NotifyFlightControllerMirroringCompleted (int priorityNum, int minDuration, int maxDuratio
                  ▶ composite_activity VerifyMirroringOfFiles (int priorityNum, int minDuration, int maxDuration)
              ▶ workframes
```

Fig. 13.11 Agent Model of Current OCA operations Brahms Model



```
BaseClass
  parents
  subclasses
    CalendarUtilClass
    SerializableObject
    SystemClass
      parents
      subclasses
        OCABaseClass
          parents
          subclasses
            File
            FileContainer
              parents
              subclasses
                Computer
                  parents
                  subclasses
                    KFXComputer
                      parents
                      subclasses
                      instances
                        PlanB
                        PlanD
                      attributes
                      relations
                      initial facts and beliefs
                      activities
                      workframes
                      thoughtframes
                    OCAComputer
                  instances
                  attributes
                  relations
                  initial facts and beliefs
                  activities
                  workframes
                  thoughtframes
                HardDisk
                KFXSoftware
              instances
              attributes
              relations
              initial facts and beliefs
              activities
              workframes
              thoughtframes
```

```
Room
  parents
  subclasses
    RoomArea
      parents
      subclasses
      instances
        ISSOCAArea
          parent
          partof
          parts
            ISSOCAMASClientArea
            ISSOCAMirrorClientArea
          paths
            3 ISSOCAArea__ISSOCAMASClientArea
            3 ISSOCAArea__ISSOCAMirrorClientArea
            5 PlanningMPSR__ISSOCAArea
          inhabitants
          attributes
          relations
          initial facts
        ISSOCAMASClientArea
        ISSOCAMirrorClientArea
        PlanningMPSR
          parent
          partof
          parts
            ISSOCAArea
            ShuttleOCAArea
          paths
          inhabitants
          attributes
          relations
          initial facts
        RedConsoleCapComArea
        RedConsoleOpsPlanArea
        RedConsolePHALCONArea
        ShuttleOCAArea
      attributes
      relations
      initial facts
  instances
    BlueFCR
    BlueMPSR
    BreakRoomFloor2
    Building30SRoom3308
```

Fig. 13.12 Object and Geography Model of Current OCA operations Brahms Model

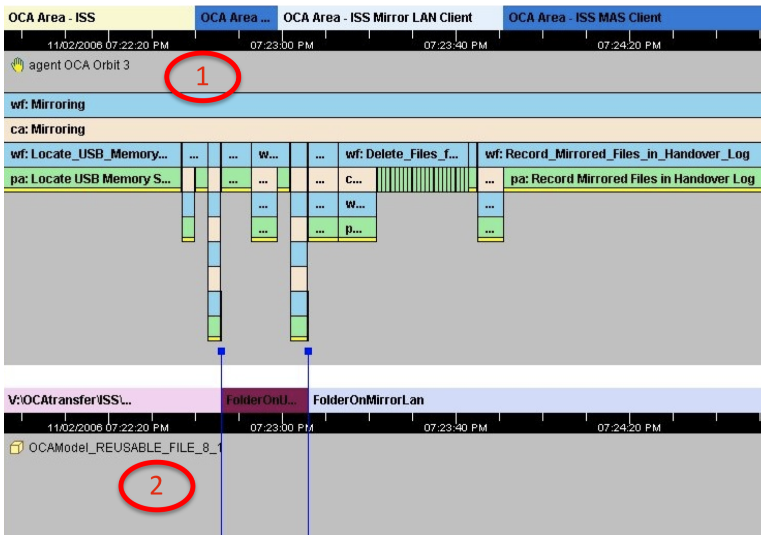


Fig. 13.13 Simulation timeline of OCA Officer's Mirroring Activity during the 3rd shift (Orbit 3)

Shift #	*Communicating*	*Configuring Resource*	*Deleting*	*Logging*	*Moving*	*Searching*	*Transferring Files*	*Verifying*	*Total Time*	*% of Month*
OCA Orbit 1	7	33	13	346	36	62	43	209	751	4.64%
OCA Orbit 2	10	45	19	412	51	91	56	272	958	6.49%
OCA Orbit 3	7	35	15	449	39	68	99	320	1035	7.02%

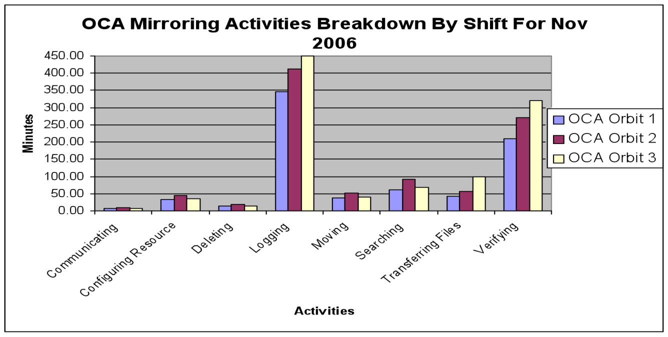


Fig. 13.14 Current operations sim statistics generated for each OCA Officer on every shift over a simulated month of work (time in minutes)

In the logging activity the OCA Officer writes the particulars of files uplinked, downlinked and mirrored in a Word document (not shown in Figure 13.13). This document provides the log of what happened during the shift, and is used for the next OCA shift to get a grasp of what happened in the previous shift. The verifying activity is where the OCA Officer verifies the processing of specific types of files (e.g. the ISS astronaut activity timeline updates files) by the Mirror LAN. As part of the copying processes, the Mirror LAN server executes and processes the "dropped" files. The OCA Officers have to verify that these batch processes execute correctly and the files are "absorbed" without errors. In case of errors, the OCA Officer needs to deliver the error files to the responsible flight controller in the MCC. Of course, all this needs to be logged in the handover log.

Figure 13.14 shows, as part of the output of the OCA current operations simulation, the amount of time the OCA Officer spends on mirroring files in a month. The table above the bar-graph shows the percent time each shift (orbit) spends on mirroring files of the total shift time. The actual numbers differ slightly each month, depending on how many files are being uplinked and downlinked. The statistics generated in Figure 13.14 are from actual uplink and downlink data from November 2006. Striking in these results is the fact that the highest time-cost for the OCA Officer are the (handover) logging and mirroring verification (verifying) subactivities.

13.7.4 Designing the future OCA work system

After the Current OCA simulation phase, a participatory design phase with a design team from the OCA Flight Officers group was started. The Future OCA Operations Work Process from Figure 13.14 was designed as a MAS that includes both the OCA

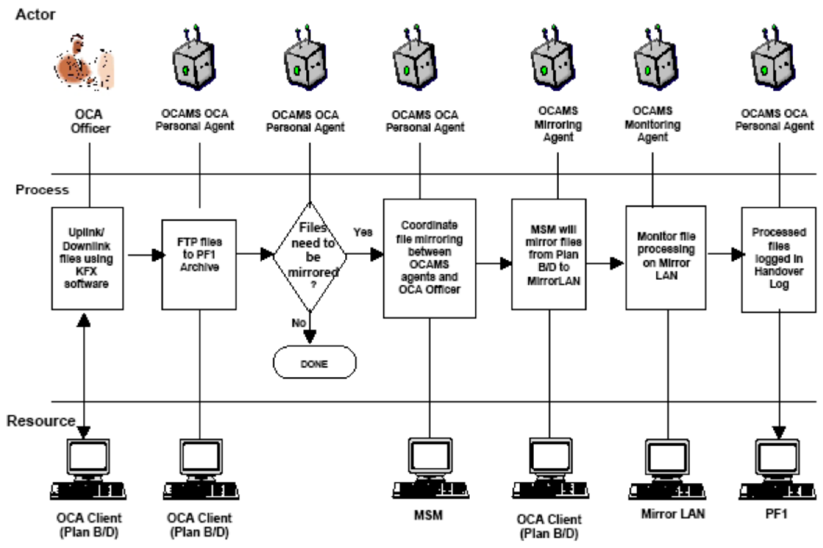


Fig. 13.15 Future OCA Mirroring Work Process

Officer and the OCAMS system. All of the Mirroring activity that in the current work process is done by the OCA Officer is now replaced by the OCA Personal and OCA Mirroring agents. Part of the design was the design of a graphical user interface (GUI) through which the OCA Officer interacts with the OCA Personal agent.

The OCAMS agents perform all the mirroring and logging activities. The OCA Officer agent only has to select uplinked and downlinked files in the GUI, and review the OCAMS activity when complete. The GUI agent displays the mirroring status back to the OCA Officer who verifies the mirroring by OCAMS. All OCAMS activity is also reported in the handover log, which the OCA Officer uses to verify mirroring was completed correctly.

13.7.5 Simulating the future OCA work system

The future operations simulation model was then implemented as a Brahms MAS simulation, including a simplified work practice model for the OCA Officer agent and the GUI agent (see Figure 13.16).

This *Future OCAMS MAS Simulation Model* was simulated with Brahms using the same November 2006 data. This allowed us to generate the same statistics as were generated in the Current OCA model. Comparing Figure 13.17 with Figure 13.14 there is more than ten-fold decrease in mirroring time for the OCA Officer. Especially, the logging and verifying activities have decreased significantly giving the OCA Officer time to work on other tasks. Based on these results the decision was made to develop and implement the OCAMS MAS.

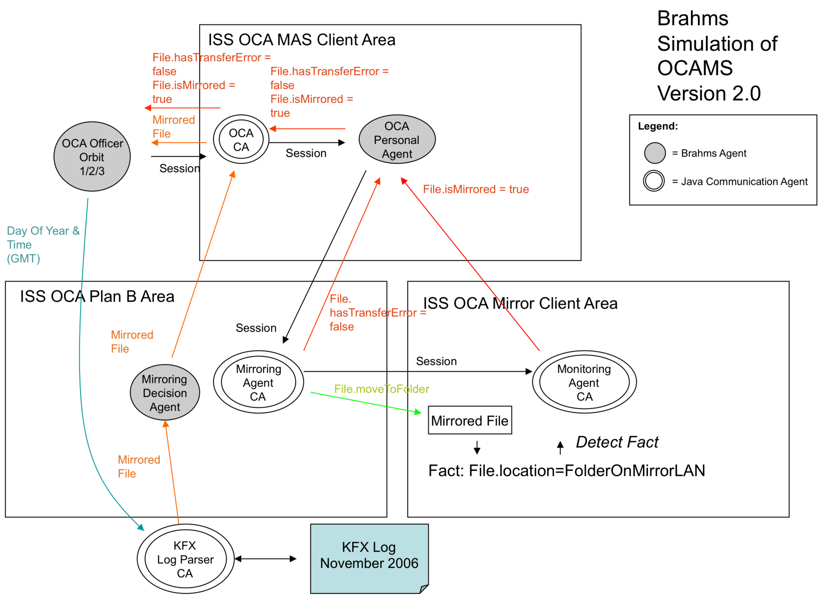


Fig. 13.16 Future OCAMS MAS Simulation Model

Shift #	*Checking*	*Communicating*	*Configuring Resource*	*Deleting*	*Logging*	*Moving*	*Searching*	*Transferring Files*	*Verifying*	*Total Time*	*% of Month*
OCA Orbit 1	17	7	2	2	0	24	0	0	16	52	0.32%
OCA Orbit 2	22	9	2	2	0	33	0	0	22	71	0.48%
OCA Orbit 3	40	7	2	2	0	27	0	0	40	79	0.54%

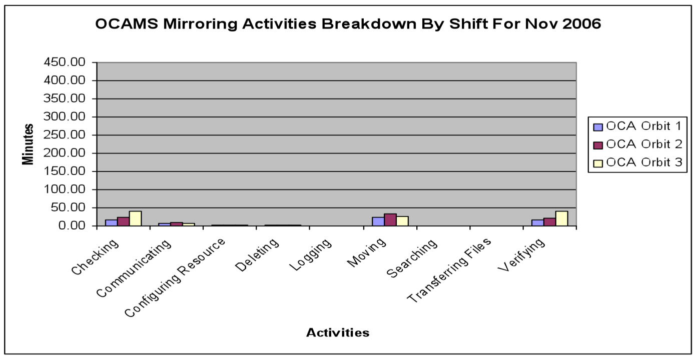


Fig. 13.17 Future operations with OCAMS: sim statistics generated for each OCA Officer on evert shift over a simulated month of work (time in minutes)

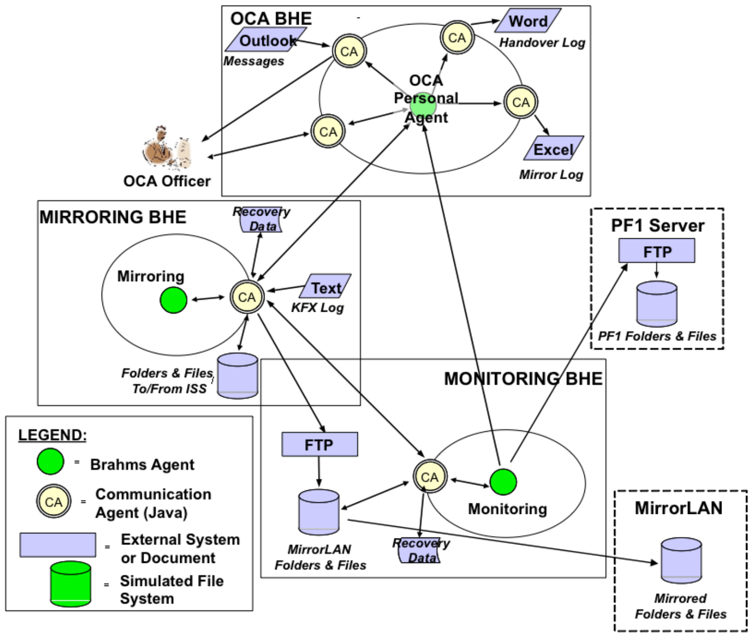


Fig. 13.18 OCAMS MAS Architecture

13.7.6 Implementing OCAMS

In the last phase we moved *from simulation to implementation* of the OCAMS MAS. This step was accomplished within one month with three developers. Architecturally, the OCAMS system is divided into three separate distributed agent systems, each of which are running in a separate Brahms Hosting Environment (BHE) (see Figure 13.18). These BHEs can run on any desired computer and network configuration, making the architecture easily adaptable to the computer architecture and network safety concerns of the ISS MCC.

The OCAMS Release 1 MAS consists of nine individual software agents (see Figure 13.18). Three agents are rule-based BDI agents written in the Brahms language, while six are Java-based agents written using the Brahms Java application interface. These Java agents are referred to as a “communication agent,” because their purpose is to communicate with external systems or files outside of the OCAMS system.

The agents are currently divided over three agent subsystems (BHEs), but could easily be loaded in a different number of BHEs. The OCA BHE consists of the central OCA Personal Agent that coordinates all work with the other agents in the system, and four human interface agents, one GUI agent and three Microsoft Office agents. The Mirroring BHE consists of the agents that interface with the KFX file uplink and downlink software currently used by the flight controllers and decides which file to mirror on the Mirror LAN. The Monitoring BHE is the workhorse of OCAMS. It has agents that monitor FTP and copy files to the Mirror LAN, and monitor for errors that might occur in the FTP and copying between file systems on the network, as well as problems with monitoring the specific file processing that needs to occur on the Mirror LAN depending on the mirrored file type. Brahms agents (the solid circles in Figure 13.18) are used there where belief- and rule-based decision-making is needed. From our experience in developing BDI-based MAS, we developed some basic rules-of-thumb for deciding whether an agent should be a BDI agent (i.e. written in the Brahms language), or an imperative agent (i.e. written in the Java language).

13.8 CONCLUSION

In this chapter we presented an agent-based engineering methodology for analyzing and designing work systems. The methodology is based on a theory for modeling and simulating work practice. This theory is founded in social science and informatics research in mainly Europe, as well as psychological theories of activities and situated action. The theory is operationalized in the Brahms multiagent environment. Brahms has been developed with the idea that an agent-based simulation based on BDI and Java agents can easily be transformed into a real-time MAS. This approach has led to the creation of our *human-centered Work Systems Design* methodology, in which we use work practice observations and participatory design to develop a simulation of the organization's work practice and model the designed software system as a MAS in Brahms, first as a simulation which later is transformed to a distributed MAS.

This methodology has been developed over almost ten years of research at NASA Ames Research Center. The approach has been successfully applied in NASA's Mission Control Center in Houston, Texas. The OCAMS system, described as a case study in this chapter, has been in operations for more than six months at the time of this writing. Currently, release two of OCAMS has been delivered to our customer. In addition to automating the mirroring activity, Release 2 automates the archiving activity as well. There are in total five releases planned for OCAMS in the next two years.

Not only is the OCAMS MAS the first MAS deployed in the ISS MCC, the system was delivered within budget, on time and without any serious problems the moment it was turned on. The system enjoys a 100% use by all OCA Officers. The success of OCAMS and the ease with which it was deployed within mission control is due to the *from simulation to implementation* methodology.

Acknowledgments

This work was supported by the National Aeronautics and Space Administration, in particular by the Mission Operations Directored (MOD) of NASA Johnson Space Center and the Intelligent Sciences Division at NASA Ames Research Center. We would like to thank the OCA team at NASA JSC for their support and acceptance of our approach and of the OCAMS system. Brahms has been funded by NASA, the former NYNEX corporation (now Verizon), and the former Institute of Research on Learning. Last, but not least, we thank all those who have been part of the Brahms development team over the last 16 years.

REFERENCES

1. H. Beyer and K. Holtzblatt. *Contextual Design: Defining Customer-Centered Systems*. Morgan Kaufmann Publishers, San Francisco, CA, 1998.
2. R. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2(1):14–23, 1986.
3. B. G. Buchanan and E. H. Shortliffe. *Rule-based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project*. Addison-Wesley, 1984.
4. G. Button and R. Harper. The relevance of 'work-practice' for design. *Computer Supported Cooperative Work*, 1996(4):263–280, 1996.
5. J. M. Carroll. *HCI Models, Theories, and Frameworks: Toward a Multidisciplinary Science*. Morgan Kaufmann, 2003.
6. P. Checkland and J. Scholes. *Soft Systems Methodology in Action*. John Wiley & Sons Ltd., Chicester, England., 1990.
7. W. Clancey, P. Sachs, M. Sierhuis, and R. v. Hoof. Brahms: Simulating practice for work systems design. *International Journal on Human-Computer Studies*, 49:831–865, 1998.
8. W. J. Clancey. Simulating activities: Relating motives, deliberation, and attentive coordination. *Cognitive Systems Research*, 3(3):471–499, 2002.
9. W. J. Clancey. Observation of work practices in natural settings. In K. A. Ericsson, N. Charness, P. J. Feltovich, and R. R. Hoffman, editors, *The Cambridge handbook of expertise and expert performance*, pages 127–146. Cambridge University Press, 2006.
10. T. H. Davenport. *Process Innovation: Reengineering Work Through Information Technology*. Harvard Business Press, 1993.
11. T. H. Davenport. The fad that forgot people. *Fast Company*, 1995(1), November 1995 1995.
12. P. Ehn. *Work-Oriented Design of Computer Artifacts (2nd Edition)*. Lawrence Erlbaum Associates, Hillsdale, NJ., 1989.
13. F. E. Emery and E. Trist. Socio-technical systems. In C. Churchman, editor, *Management Sciences, Models and Techniques*. Pergamon, London, 1960.

14. Y. Engeström. Expansive visibilization of work: An activity-theoretical perspective. *Computer Supported Cooperative Work*, 8:63–93, 1999.

15. Y. Engeström. Activity theory as a framework for analyzing and redesigning work. *Ergonomics*, 43(7):960–974, 2000.

16. FIPA. Fipa communicative act library specification, 2002.

17. J. Greenbaum and M. Kyng, editors. *Design at Work: Cooperative design of computer systems*. Lawrence Erlbaum, Hillsdale, NJ., 1991.

18. P. Ehn. *Work-Oriented Design of Computer Artifacts (2nd Edition)*. Lawrence Erlbaum Associates, Hillsdale, NJ., 1989.

19. R. Kling and S. L. Star. Human centered systems in the perspective of organizational and social informatics. *Computers and Society*, 28(1):22–29, 1998.

20. J. Lave. *Cognition in Practice*. Cambridge University Press, Cambridge, UK, 1988.

21. A. N. Leont'ev. *Activity, Consciousness and Personality*. Prentice-Hall, Englewood Cliffs, NJ, 1978.

22. M. Luck and L. Padgham, editors. *Agent-Oriented Software Engineering VIII: 8th International Workshop, AOSE 2007*. Springer, Honolulu, HI, 2008.

23. G. H. Mead. *Mind, Self, and Society; From the standpoint of a social behaviorist*. University of Chicago Press, Chicago, IL., 1934.

24. B. A. Nardi, editor. *Context and Consciousness: Activity Theory and Human-Computer Interaction*. The MIT Press, Cambridge, MA, 1996.

25. M. Polanyi. *The Tacit Dimension*. Peter Smith, Magnolia, MA, 1983.

26. J. Rasmussen, A. M. Pejtersen, and L. P. Goodstein. *Cognitive Systems Engineering*. 1994.

31. D. Schön. *The Reflective Practitioner: How Professionals Think in Action*. Basic Books, 1982.

28. R. Searle, John. *Speech Acts*. Cambridge University Press, Cambridge, UK, 1969.

29. R. Searle, John. A taxonomy of illocutionary acts. In K. Gunderson, editor, *Language, Mind, and Knowledge*, volume 1-29, pages 344–369. University of Minnesota, Minneapolis, 1975.

30. P. M. Senge. *The Fifth Discipline: The Art and Practice of the Learning Organization*. Random House, Inc., 2006.

31. D. Schön. *The Reflective Practitioner: How Professionals Think in Action*. Basic Books, 1982.

32. M. Sierhuis. “it’s not just goals all the way down” – “it’s activities all the way down”. In G. M. P. O’Hare, A. Ricci, M. J. O’Grady, and O. Dikenelli, editors, *Engineering Societies in the Agents World VII, 7th International, Workshop, ESAW 2006, Dublin, Ireland, September 6-8, 2006, Revised Selected and Invited Papers*, volume LNCS 4457/2007 of *Lecture Notes in Computer Science*, pages 1–24. Springer, Dublin, Ireland, 2007.

33. M. Sierhuis, W. J. Clancey, and R. J. v. Hoof. Brahms: An agent-oriented language for work practice simulation and multi-agent systems development. In Rafael H. Bordini, Mehdi Dastani, J. Dix, Amal El Fallah-Seghrouchni editors, *Multi-Agent Programming, 2nd Edition*. Springer, To Appear.

34. J. D. Sterman. *Business dynamics: systems thinking and modeling for a complex world*. Irwin/McGraw-Hill, 2000.

35. L. A. Suchman. *Plans and Situated Action: The Problem of Human Machine Communication*. Cambridge University Press, Cambridge, MA, 1987.

36. K. J. Vicente. *Cognitive Work Analysis: Toward Safe, Productive, and Healthy Computer-based Work*. Lawrence Erlbaum Associates, 1999.

37. L. S. Vygotsky. *Mind in Society: The Development of Higher Psychological Processes*. Harvard University Press, Cambridge, MA, 1978.

38. E. Wenger. *Communities of Practice; Learning, meaning, and identity*. Learning in doing: Social, cognitive and computational perspectives. Cambridge University Press, Cambridge, MA, 1999.