Paper submission to the special issue on “Simulating Organisational Processes”, Journal for Simulation Modelling Practice and Theory, Elsevier, The Netherlands

Brahms

A multiagent modeling environment for simulating work practice in organizations.

Maarten Sierhuis[1]
William J. Clancey[2]
Ron van Hoof[3]

Contact Info

Maarten Sierhuis
RIACS/NASA Ames Research Center
Mail Stop B269-1
Moffett Field, CA 94035-1000, USA
msierhuis@mail.arc.nasa.gov

Abstract

Modeling and simulating organizational processes is often done at such an abstract level that individual work practice—collaboration, communication, ‘off-task’ behaviors, multi-tasking, interrupted and resumed activities, informal interactions, use of tools and movements—is left out, making the description of how the work in an organization actually gets done impossible. This paper describes the Brahms modeling and simulation environment, developed at NASA Ames Research Center. The Brahms modeling language is geared towards modeling people’s activity behavior, making it an ideal environment for simulating organizational processes at a level that allows the analysis of the work practice and designing new work processes at the implementation level.

Brahms

A multiagent modeling environment for simulating work practice in organizations.

Maarten Sierhuis[1]
William J. Clancey[2]
Ron van Hoof[3]

1. Research Institute for Advanced Computer Science (RIACS)
2. Institute for Human-Machine Cognition (IHMC)
3. QSS Group, Inc.
NASA Ames Research Center, CA 94035-1000, USA

msierhuis@mail.arc.nasa.gov
William.J.Clancey@nasa.gov
rvanhoof@mail.arc.nasa.gov

keywords multi-agents, teamwork, work practice, human behavior modeling, simulation

Introduction

In this paper we position Brahms as a tool for simulating organizational processes. Brahms is a modeling and simulation environment for analyzing human work practice, and for using such models to develop intelligent software agents to support the work practice in organizations. Brahms is the result of more than ten years of research at the Institute for Research on Learning (IRL), NYNEX Science & Technology (the former R&D institute of the Baby Bell telephone company in New York, now Verizon), and for the last six years at NASA Ames Research Center, in the Work Systems Design and Evaluation group, part of the Computational Sciences Division (Code IC). Brahms has been used on more than ten modeling and simulation research projects, and recently has been used as a distributed multiagent development environment for developing work practice support tools for human in-situ science exploration on planetary surfaces, in particular a human mission to Mars [1] [2] [3] [4] [5].

Brahms was originally conceived of as a business process modeling and simulation tool that incorporates the *social systems of work*, by illuminating how formal process flow descriptions relate to people's actual located activities in the workplace. Our research started in the early nineties as a reaction to experiences with work process modeling and simulation [6]. Although an effective tool for convincing management of the potential cost-savings of the newly designed work processes, the modeling and simulation environment (Sparks™ from Coopers & Lybrand) was only able to describe work as a normative workflow. However, the social systems, uncovered in work practices studied by the design team played a significant role in how work actually got done—*actual lived work* [7]. Multi-tasking, informal assistance and circumstantial work interactions could not easily be represented in a tool with a strict workflow modeling paradigm. In response, we began to develop a tool that would have the benefits of work process modeling and simulation, but be distinctively able to represent the relations of people, locations, systems, artifacts, communication and information content [8]. Thus, Brahms models work processes at the work practice level.

Agent architectures often do not link to theories of human behavior, or empirical data on human

This research has been funded by the National Aeronautics and Space Administration (NASA) under the Human-Centered Computing Technology Area, part of the Intelligent Systems Project of the Computing, Information and Communications Technology Program (CICT)

behavior in comparable situations. The Brahms environment is based on a number of behavioral and cognitive theories, most important situated cognition [9], activity theory [10] [11], situated action [12] [13] and cognitive modeling [14] [15].

Brahms has been validated as a modeling and simulation tool for work practice design and analysis. Simulation models of the work practices of the Apollo astronauts on the surface of the Moon have been developed, simulated and validated with empirical Apollo mission data available from NASA, such as video analysis, voice transcripts and lunar surface procedures [16]. The Brahms modeling language is dedicated to modeling human behavior at the work practice level. This means that the Brahms language is particularly suited for modeling humans working together as individuals in organizations performing individual and teamwork activities. The Brahms language is unique in that it not only models both individual agent and group behavior, but also systems and artifact-behavior, human interaction, as well as the interaction with the environment and its influence on behavior. Most other multiagent languages leave out artifacts and the interaction with the environment, making it difficult to develop a holistic model of real-world situations, based on ethnographic observation and data collection (c.f. [17]). Brahms makes it easier to model empirical data gathered using ethnographical observations and data collection, because Brahms is geared towards modeling real-world activities.

We first explain what we mean by the term *work practice*, and describe our theory of modeling work practice based on existing theories of activity theory, situated action and distributed cognition. We then discuss the Brahms language in detail, specifying the different conceptual models that build up a Brahms model, providing model examples and code fragments to explain the representational capabilities and workings of Brahms. We end the paper with a discussion of the use of Brahms as a organizational process simulation tool.

Modeling Work Practice – a theoretical view

Work practice is embodied in the way people perform their daily work activities in organizations. Our notion of work practice modeling has been developed as a reaction on the traditional views of workflow modeling in organizations. The concept of work practice originates in the research disciplines of socio-technical systems, business anthropology, and management science, focusing on both the informal and formal features of work and applying ethnography and participant observation to the analysis and design of human-machine work systems [18] [19] [20] [21] [22] [6] [23] [24] [25].

We define work practice as the collective activities of a group of people who collaborate and communicate, while performing these activities synchronously or asynchronously. Very often, people view work merely as the process of transforming input to output, which is a Tayloristic view. Work practice includes how people behave in situations, at specific moments in the real world. To describe people's circumstantial behavior we need to include ecological (environmental) influences on individual activity (not only problem-solving behavior), such as collaboration, 'off-task' behaviors, multi-tasking, interrupted and resumed activities, informal interaction, use of tools, and movements [8] [16].

Our theory about modeling work practice is based on a number of elements borrowed from different existing approaches. It should be noted that Brahms models are models about real world phenomena, and thus the model is a description of the world as viewed by the modeler. Models of work practice are description of work practice and are not the same as real work practice. Winograd and Flores explain that just as we can ask how *interpretation* plays a role in understanding text, we can ask how it plays a role in understanding the world as a whole. They put forward four assumptions that, simply put, explain the way humans interpret the world [26]. We relate this, more narrowly to the way we can interpret how people work, and we postulate the following four worldviews:

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

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

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

This means that we cannot model a world by just modeling the individual interpretations of that world. We need to separate the different individual interpretations from the 'objective facts.' Here we get confronted with *solipsism*[1], i.e. the modeler of the 'objective facts' is also an individual in the world, and hence also interprets the facts of the world according to his or her subjectivity. However, it is important to make a distinction between modeling the interpretation of an individual in a world, and the interpretation of facts in the world. Both are subjective, but both are necessary if we want to take a holistic view of the way people work. However, we should never forget that this means that our model of work practice is *our* interpretation of reality.

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

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

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

This means that if we want to model work practice, we need to model physical motion of individuals. We can satisfy this assumption by simply modeling the causal relation between thoughts about action and physical motion, and we do not need to model how this happens in the human body (i.e. the neurophysiology)[2].

These four worldviews are our starting point for describing work practice as a knowledge-level concept. The *context* in which people perform real world activities is an important aspect that we describe next.

Understanding context

A broad range of work in psychology and anthropology has shown that to fully understand how people work we need to study context in order to understand the relation between individuals, artifacts and social groups. We describe three approaches in the study of context—activity theory, situated action models, and distributed cognition—that have been fundamental in the development of our theory for modeling work practice. All these approaches use the notion of *activity* as the central point in the way

---

[1] The theory or view that the self is the only reality (American Heritage Dictionary).

[2] We are currently working with Digitalspace, Inc. on a 3-D virtual reality integration with Adobe's Atmosphere. The BrahmsVE will allow representing agents with three-dimensional bodies in a virtual world, thus dealing with the world as a three-dimensional space.

they analyze the context in looking at human behavior.

Activity theory

Activity theory goes back to the 1920s and the developmental psychology work done in the former Soviet Union. The main developers of activity theory are Vygotsky and Leont'ev [11] [10]. In activity theory the unit of analysis is an activity. An *activity* is composed of a subject, the object, its actions and operations. A *subject* is the person or group of persons that is engaged in the activity, which makes the analysis of activities focus our attention on one or more people.

An *object* is the objective of the activity as it is held by the subject(s) and *motivates* them in the engagement. *Actions* are processes that must be undertaken to fulfill the object. Subjects are conscious about the actions to take to accomplish the object of an activity. The notion of an activity can span multiple actors being engaged together in coordinated actions. The actors engaged together might actually have different, even conflicting objects [27]. This is an important concept for understanding teamwork and what collaboration between individuals is about [28].

Operations are routinized and unconscious actions. For example, when learning to drive a car with a standard gear, the shifting of the gear is at first a conscious action with an explicit goal. Later on, when we are well versed in driving with a stick shift, shifting gears becomes operational and is not a specific goal-driven process anymore. The difference of actions and operations reminds us strongly of the difference between explicit and tacit knowledge [29]. The important take away point from this is that it seems that activities are decomposed into actions, when the activity is not yet ‘automatic,’ while an activity that is already operationalized is not decomposed into lower-level actions, but can be seen as a primitive action. Another key notion in activity theory is the notion of *mediation* by artifacts [27]. Artifacts include instruments, machines, et cetera, that mediate activity and are created or used by people to control their behavior.

Engaging in an activity creates a context through its enactment of actions and operations of those involved and the artifacts used to coordinate action. As such, we can see practice as the engagement in activities over a period of time.

Situated action models

Situated action models emphasize the emergence of activities within the situation. Situated action can be viewed as the refinement of activity theory as a tool in the analysis of every day activities, as opposed to Vygotsky’s original pedagogical tool for understanding children’s learning behavior and interaction between teacher and child. The focus is therefore on *situated action* or what we call *practice* as opposed to problem solving, which means that it is an inquiry into the everyday activity of people acting in a particular setting. The analysis of situated action is a moment-by-moment analysis of the interaction between people, and between people and the artifacts used in a particular situation. Lave identifies the basic unit of analysis for situated action as the *activity* of people as it relates to the setting in which this activity takes place and is constructed at the same time; “The setting is both generated out of [the] activity and at the same time generates the activity” [30]. A *setting* is the relation between acting people and the arena in which they act, almost like a theatrical play. The *arena* is the physical place, i.e. the geographical space, as well as the institution with its social, political and economical background, like the stage within the theatre [31].

An important aspect of the focus on people acting within an arena is that it forces the analyst to pay attention to the flux of the ongoing activity, the minute-by-minute understanding of a real activity in a real setting [32]. One of the interesting notions coming out of situated action studies, put forward by Suchman, is that plans are not the mechanism to action, but are resources for action; a ‘retrospective reconstruction’ of situated action [12]. In that sense, we postulate that goals are generated within the activity, as an individual's ongoing realization of the intention of the activity [33].

Distributed cognition

Distributed cognition is a branch of cognitive science that studies the representation of knowledge both inside the heads of people, as well as within the artifacts and systems they use [34]. A cognitive system can be seen as always being engaged in an activity, in the definition of activity theory and situated action. For example, Hutchins, in his study of the activity of ‘flying a plane,’ describes the cognitive system as the total setting of the cockpit [35]. He takes the *cockpit system* as the unit of analysis and observes the many representations that are inside the cockpit system, yet outside the head of the pilots. By taking this social-technical systems approach he can describe the ‘cognitive’ properties of the system, meaning giving an account of the system's behavioral properties in terms of its internal representations, without saying anything about the processes that operated inside the heads of the individuals within the system. Thus, distributed cognition moves the unit of analysis to the system as a whole, and analyzes the functioning of the system as a ‘functional unit,’ instead of as a cognitive system. In doing this, the emphasis is on understanding the coordination among the individuals and the artifacts in the system. This understanding is created by focusing on the available information in the system, as represented in the artifacts and the heads of the individual. There is less of a focus on the activity and situated-actions by the people in the system as a whole, but more on how the lack of information creates a breakdown in the execution of plans and tasks by the individuals in the system.

The interesting part of distributed cognitive analysis for getting an understanding of the work practice of pilots is the focus on the ‘memory’ of the system as driving the activities of the pilots. This emphasizes the importance of a total systems view as part of the context in the understanding of practical knowledge.

Our notion of activities

The key construct in the Brahms language is an *activity* and is easily confused with the traditional notion of a *task*—a representational construct that describes human behavior in terms of problem-solving with goals and operators (cf. [36]). Some argue that there is no distinction between tasks and activities and thus all human behavior can be simulated using a problem-solving framework (e.g. Soar, [37] and ACT-R, [15]). Here we merely touch upon the difference, and briefly discuss that representing human activity at the work practice level is a different paradigm for representing human behavior then the paradigm of representing human behavior as problem-solving behavior or as functional task behavior. For a more detailed discussion on the nature of activities, see [33].

The theory of humans as an information processing systems (IPS, [37]) defines problem solving in terms of pursuing pre-specified *goals* in order to accomplish pieces of work that need to be done (i.e. *tasks*). The specification of a goal is a way to make a stated problem actionable, that is, solvable by means of well-defined decisions. Problem solving is the systematic search over the problem space describing how one can attain a goal. Such an approach is in contrast to a theory for describing *how people actually work* within the constraints of their environment, and how the environment determines their actions and the interactions with other people and artifacts in that environment. Describing the behavior in terms of what actually happens in the world does not lead to a description of the individual’s problem-solving behavior. Rather, it leads to a description of the emergent *total system behavior* in terms of the individual interactions, responses to the other elements in the system (people and artifacts), as well as the emergent sequence of individual *activity* (i.e., the state of being active), something Newell calls 'microepics.'

In using Brahms as a modeling and simulation environment for an organization's work processes, we emphasize the difference between describing people's work behavior as the performance of subsumed situated activities and describing it as the sequential performance of tasks in a business process, with the flow of information between organizations being the focus. Unlike an activity-level description, describing individual behavior at a functional process-level focuses on the decision-making behavior only as it relates to the idealized tasks in the business process. The actual behavior, including off-task behaviors and interruptions, of individuals cannot be included, because only those tasks that are part of the workflow can be represented. People are represented as functional resources—similar to any information process—from which communication can only flow according to well-defined organizational channels. Individual work practice is idealized or left out.

In contrast, in Brahms a business process is described as the individual activities of people in the organization. We are forced to describe each person's behavior as situated in the organization's context; their physical location, tools used, personal knowledge gathered over time, organizational culture adopted. In short behavior is described in terms of people's work-practice. The question not only becomes what activities each individual performs, but more specifically, how they get to perform these activities. As we describe work practice in terms of activities engaged in, we get to ask when and how people's interactions are constraint by activities. For example, although we are still parents when at work, we do not engage in parenting activities while at work, until our child calls us at work to ask a question. In that sense, we are constantly managing our situated activities in context. To view a work process as the interaction of activities of people over time makes us view workflow as the practice of people, instead of a flow of information through a well-defined hierarchical organization.

The next section describes the Brahms language and explains what is meant with concepts multiagent, rule-based and activity.

The Brahms Language

We will explain the modeling concepts of the language. For a more detailed description of the language see Sierhuis [38] and van Hoof & Sierhuis [39]. The Brahms language is necessarily an agent and object-based language in which people are represented as intentional agents and artifacts in the world as objects (with or without behavior), positioned in a model of places. Agents in Brahms are belief-desire-intention-like (BDI) agents, with beliefs representing not only the desires of the agents, but also the agent's understanding of the world, and workframes and thoughtframes representing their intentions to perform activities (see Figure 1) [40] [41]. The work practice of a person is represented as a combination of activities that can be performed by an agent representing the person, situated-action rules (called *workframes*) constraining when the agent can perform the activities, *beliefs* the agent acquires by means of reasoning (e.g. problem-solving), perception of *facts* in the world depending on *location* and current activity, and *communication* with other agents and objects.

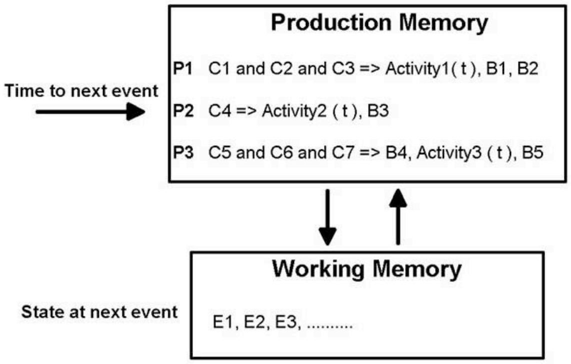

Production Memory
P1 C1 and C2 and C3 => Activity1(t), B1, B2
P2 C4 => Activity2 (t), B3
P3 C5 and C6 and C7 => B4, Activity3 (t), B5
Time to next event
Working Memory
State at next event
E1, E2, E3, ...

Figure 1. Situated-Action Rules in a Production System

The Brahms language was first of all designed to help researchers develop simulations of human and machine behavior. However, the current version of Brahms also supports the development of multiagent software systems that are based on models of human activity behavior. To enable the modeling of human activity behavior, the Brahms language embodies assumptions about how to describe social situations, workplaces and work practice. Thus, Brahms is an agent language that operationalizes a theory for modeling work practice, allowing the researcher to develop models of human activity behavior that corresponds with how people actual behave in the real world. This is in contrast to an agent-language such as Swarm [42], an often-used language for modeling and simulating social and economic behavior of large agent societies [43]. Swarm is *not* based on any particular theory of human behavior and is not an agent modeling language in the strict sense of the word, but rather an addition in the form of object libraries for Objective-C (an object-oriented programming language) [44]. Swarm agents are not intentional, but are simple objects with behavior represented as inherited imperative methods that can be triggered by a higher-level schedule object in the model. In contrast, Brahms is a pure agent language where agent actions are not scheduled by an overall scheduler, but by each agent having its own inference engine scheduling and executing the agent's activities based on its current belief-set. Figure 2 shows how multiple Brahms agents interact with each other and the world environment. Each agent has is its own inference engine, allowing independent behavior using situated-action rules (Figure 1) (see section The Activity Model). Agents can communicate their beliefs via a simulation scheduler (see section The Communication Model for a description). Agents and activities can also be written in Java.

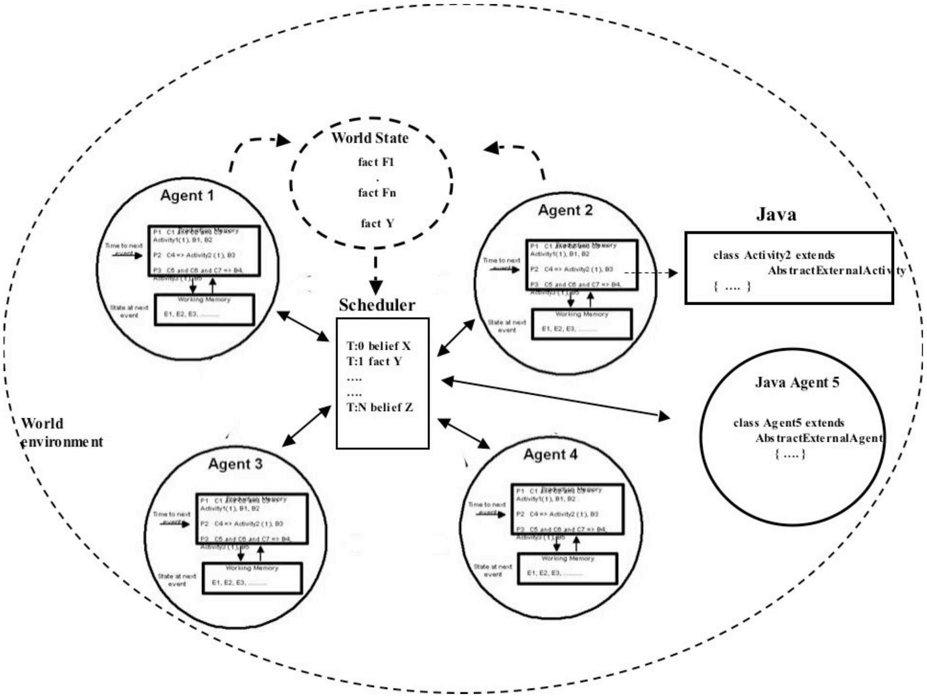


Figure 2. A Multi-Agent BDI-language

An Example Model: Simulating a Robotic Mission to the Moon

Sending robots to the Moon or to Mars is a difficult task, one that mission designers at the Jet Propulsion Laboratory (JPL) in Pasadena, CA spent years to design and implement. Understanding how scientists and engineers will work together during a mission, and also how people will communicate with the robot on the planetary surface is a research topic in order to design better mission operations for future missions [45] [28]. We use our work in modeling mission operations for robotic missions to the Moon and Mars as our example to explain the Brahms language and its representation capabilities. The Victoria model is a Brahms model of the mission operations for the Victoria mission proposal. The Victoria mission was a proposed robotic mission to the Moon. The model simulates the mission operations concept for the mission in such detail that the model allows for the understanding of the relationship between mission and science support on the ground and the efficiency of the robot in exploring the lunar surface for water ice [16] [45].

Model Skeleton

A Brahms model consists of several model files. Brahms model files are ascii-files ending in a .b extension and consisting of legal Brahms syntax. Good modeling practice is to create a separate source file for each Brahms model element, such as groups and agents, classes and objects, although one can write a Brahms model completely in one source code file if one so pleases. A common approach is to create one main model file that imports all other model files. Since Brahms does not have an initialization function, such as the 'main' function in the C program language [46], the main model file simply contains import statements for the agents and objects in the model. Excerpt 1 shows the main model file for the Victoria mission operations model.

Excerpt 1. Model File

```
package Victoria;

import MyBase.*;
import *;
```

The package statement defines that the model exists as a package called Victoria. A package is a directory file structure allowing the modeler to compartmentalize the model into appropriate sub-directories. The import statement loads in the needed model files. Excerpt 1 shows that the model is loading all .b files in the MyBase sub-directory of the Victoria package, as well as all the .b files in the Victoria package's root directory. To start a simulation a compiled version of this file is the model file that is first loaded into the Brahms virtual machine (BVM). The Brahms compiler compiles each .b file separately into a Brahms 'byte-code' file. The 'byte-code' language for Brahms is a XML data definition language, making each compiled Brahms model file a Brahms XML file that can be loaded in and executed by the BVM.

The Agent Model

When developing a Brahms model we first design an Agent Model. The Agent Model is a conceptual modeling construct and is not a language construct within the Brahms language. The Brahms modeling approach is based on a model-based method that divides any system to be modeled into a number of more or less interdependent sub-models, the Agent, Object, Geography, Knowledge, Activity and Communication model. The Brahms model development environment—the Composer—supports this model-based approach, and allows the modeler to create groups and agents using a graphical user interface.

Group hierarchy

The agent model consists of a group hierarchy representing the social, organizational or functional groups of which agents are members. In the mission operations domain we can represent the mission operation workers according to their functional roles, such as the science team. Members of the science team are responsible for the science deliverables of the mission. They are often world-class scientists in specific domains, such as specialized science instruments that are carried onboard the robot. The science team members are divided into science theme groups that represent the functional roles during the mission, such as the 'instrument synergy team,' the 'science operations team' and the 'data analysis and interpretation team'. Excerpt 2 shows the definition of some of the groups in Brahms source code (the excerpt shows partial source code; '…' means that source code is left out):

Excerpt 2. Partial Agent Model

```
group MyBasegroup memberof BaseGroup {
   attributes:
          public symbol groupMembership;
}

group VictoriaTeam memberof BaseGroup {…}

group ScienceTeam memberof VictoriaTeam, MyBaseGroup {
   location: Building244;

   attributes:
          …
   initial_beliefs:
          (VictoriaRover.location = ShadowEdgeInCraterSN1);

   activities:
          …
   workframes:
          …
   thoughtframes:
}

group ScienceOperationsTeam memberof ScienceTeam {…}

agent Agent1 memberof ScienceOperationsTeam {
   initial_beliefs:
          (current.groupMembership = ScienceOperationsTeam);

   intial_facts:
          (current.groupMembership = ScienceOperationsTeam);
}
```

We will go step-by-step through the source code of Excerpt 2 explaining how groups and agents are defined. Note that this excerpt describes the definition of four groups and one agent. The bold characters show Brahms language keywords. Every Brahms language element definition is actually placed in a separate source file, but is here shown as if it is part of one source code file.

The first two groups are *MyBaseGroup* and *VictoriaTeam*. *MyBaseGroup* is a group defined by the modeler and is used to define common features for all groups. It is a non-domain specific ‘root’ of the group hierarchy, used by the modeler to define common group properties. *MyBaseGroup* and *VictoriaTeam* are both members of the group *BaseGroup*, which is the root of all groups and is part of a base library that comes with the Brahms language, with certain predefined standard attributes. Here the *MyBaseGroup* group defines a common attribute for all groups, i.e. the *groupMembership attribute*. The *groupMembership* attribute is used in the model to allow agents to know to what group they belong. The third group that is defined is *ScienceTeam*. The group *ScienceTeam* is a member of two parent groups, *VictoriaTeam* and *MyBase*. This example shows that Brahms supports multiple inheritance for groups and agents. Group inheritance means that the subgroups and/or agents inherit all the elements defined in the parent group. The Brahms compiler will recognize naming conflicts in multiple inheritance and will report these at compile time. At this moment Brahms does not support ‘late-binding’ and thus there are no possible inheritance conflicts at run-time. Next, the group *ScienceOperationsTeam* is defined as a member of the *ScienceTeam* group. Last, but not least, is the definition of an actual agent. The keyword *agent* declares agents, and in this example *Agent1* is an agent that is a member of the *ScienceOperationsTeam* group. Thus, the definition of groups and agents in Excerpt 2 explicitly defines the group hierarchy in Figure 3

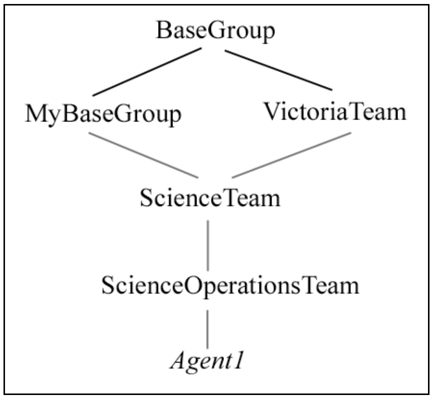


Figure 3. Group Hierarchy from Excerpt 2

Agent beliefs

Intentional agents are entities whose behavior can be predicted by the method of attributing belief, desire and acumen [47]. This philosophical stance has resulted in representing intentionality as a logical framework in which agents have beliefs and there is a deduction model for beliefs [48]. Brahms agents are intentional and represent this intentionality as the set of beliefs at time t and the set of rules (workframes and thoughtframe) that can be used to act in the world and deduce new beliefs. Beliefs are represented as first-order logic propositions. An agent's belief-set changes over time based on actions in the world, communication with other agents, world fact detection and reasoning. As the belief-set of an agent changes, the behavior of the agent can change. In other words, there is a logical relationship between an agent's intention and its action in the world.

An agent's beliefs are object- or agent-attribute-value triplets (OAV). The modeler can specify initial beliefs for an agent. Initial beliefs are beliefs that the agent receives at initialization. Initial beliefs specify the initial belief-set of an agent in the model and are a way to define initial scenarios for a simulation run. Excerpt 2 shows that *Agent1* will have two beliefs in its initial belief-set. The first is an initial-belief that is declared at the agent-level (i.e. in *Agent1*). The standard form of beliefs is *(AgentOrObject.attributename = value)*. The initial belief in *Agent1* states that the agent belongs to the group *ScienceOperationsTeam* (the keyword *current* represents the agent itself, and is bound at run-time for each agent). The second belief of *Agent1* is inherited from the *ScienceTeam* group. Excerpt 2 shows an initial-belief declared in the *ScienceTeam* group. This belief states that the *VictoriaRover* agent is located at the *shadow-edge of crater SN1*.

Beliefs are represented as values for attributes of agents or objects. Brahms is a strongly-typed language, which means that every attribute value or parameter is type-checked during compile- and runtime. In order for an agent to get a specific belief, the attribute and its type needs to have been defined. In Excerpt 2 the declaration of the *groupMemberShip* attribute is shown in the *MyBaseGroup* group as an attribute of type *symbol*. *Agent1* inherits this attribute and thus any agent can have a belief about the group membership of *Agent1* (not only those that inherit this attribute). The initial-belief in *Agent1* declares this belief for *Agent1*, but other agents can have this belief as well (this is not shown in Excerpt 2). Since beliefs are OAVs, another agent can have a different belief about *Agent1*'s group membership, e.g. agent *Agent2* can belief that *Agent1* is a member of the *InstrumentEnergyTeam*. Thus, it is possible that different agents have either similar or different beliefs about aspects of the world, allowing similar type agents to have a different belief-set and thus behave differently (see section The Activity Model).

World facts

If agents can have different beliefs about attributes of agents or objects, how can we represent the actual state of the environment in which agents are located? Brahms operationalizes the second world-view from Winograd and Flores by representing 'objective facts' about the world as *facts* in the simulated environment, similarly as beliefs. In some sense we can see the environment as an implicit object (the World object) with a fact-set. Agents and objects can create facts in the world either by acting in the world or as initial-facts, similar as initial-beliefs (see section World facts). Excerpt 2 shows that at initialization *Agent1* creates a fact about its group membership. The meaning of the declaration of the same initial-belief and initial-fact is that not only does *Agent1* believes it is a member of the *ScienceTeam* group, it is also a *true* fact in the simulated world. Where beliefs are local to an agent, facts are not, and thus we could have also represented that the fact is that the agent is a member of the science team, but the agent is simply not aware of that fact (i.e. it does not have the belief). Thus, facts in the model represent the objective truth about the state of the simulated world.

The Object Model

Similar to the Agent Model, the Object Model defines the objects in the world. There are two types of objects, physical artifacts—plainly called *objects*—with or without behavior, and concepts represented as *conceptual objects* with attributes. Using these two distinct object types we can both represent the behavior of physical objects in the world (e.g. computer, spacecraft, science instruments, etc) or a concept—a non-physical entity—of which an agent can have beliefs. *Objects* can have beliefs and create facts, similarly as agents. However, *conceptual objects* cannot. Objects and conceptual objects can be part of a class inheritance hierarchy, similar to other object-oriented programming languages (see Figure 4), a screenshot of the MyBaseClass from the Brahms development environment)

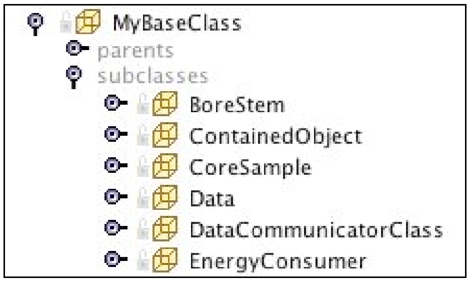


Figure 4. Class Hierarchy

The Knowledge Model

The knowledge model consists of production rules for agents and objects. Production rules in Brahms are forward chaining inference rules associated with groups and agents, acting on the beliefs of an agent. These rules are called *thoughtfames*, because using these rules an agent can 'think' and deduce new beliefs. Thoughtframes can be declared in groups, agents, classes and objects. Each agent and object has a set of thoughtframes, a combination of thoughtframes locally declared and inherited. The *ScienceTeam* group from Excerpt 2 shows the position of a thoughtframe section underneath which all the group's thoughtframes need to be declared. Here, Excerpt 3 shows the declaration of the *CalculateEnergyLevel* thoughtframe in the *Rover* group.

Excerpt 3. Partial Knowledge Model for Rover group

```
group Rover memberof MyBaseGroup {
  …
```

```
      thoughtframe CalculateEnergyLevel {
          repeat: true;
          variables:
                 forone(double) energyused;

          when (knownval(current.energyUsedInActivity = energyused) and
                knownval(current.consumedEnergy = true))
          do {
                 conclude((current.energyUsed = current.energyUsed + energyused));
                 conclude((current.energyLevel = current.energyLevel – energyused);
                 conclude((current.energyLeftToUse = current.energyLeftToUse -
                                                    energyused));
                 conclude((VictoriaRover.consumedEnergy = false));
          } //end do
      } //end thoughtframe
    …
  } //end group
```

A thoughtframe (TFR) consists of a number of elements which we will describe using Excerpt 3. First of all, a TFR is used to infer new beliefs based on current beliefs in the belief-set of the agent. New beliefs are created when a thoughtframe executes. The *conclude* statement in the do-part or body of the TFR creates a new belief for the agent. Excerpt 3 shows four such *conclude* statements, each of the form *(O.A = ν)*, where *O = 'current'*, *A = [an attribute of the Rover group]* and *ν* is the outcome of a numerical expression that is evaluated before the belief is created.

The numerical expression ν is evaluated as follows; ν ::= Operand-1 Operator Operand-2. In all four conclude statements in Excerpt 3, Operand-1 is of the form O.A, where O = 'current' and A = [an attribute of the Rover group]. Operand-2 is not of the same form as Operand-1. In this case, Operand-2 is the name of a variable of type double declared in the TFR (i.e. energyused). Because of the forward chaining inference mechanism, the value of this variable had to be bound in the precondition before the TFR can 'fire'. The precondition is the when-part of the TFR in Excerpt 3. To explain how the variable gets its value, we need to explain how the precondition of a TFR is matched.

Let us investigate the matching of the first precondition from Excerpt 3, *knownval(current.energyUsedInActivity = energyused)*. The *knownval* keyword means that the agent's inference engine needs to find a belief in the belief-set of the agent of the form given in between the round brackets. The inference engine will pattern-match on the left-hand side (lhs) of the belief-pattern. First, the value of the variable *current* is bound to the current agent for which the TFR is being executed (e.g. *Rover-1*). The pattern-matching algorithm finds *all* beliefs that match the lhs (e.g. *Rover-1.energyUsedInActivity*), potentially returning a list of beliefs matching this pattern. Next, the right-hand side (rhs) of the precondition is evaluated and the result matched against the list of beliefs returned by this initial pattern matching. In this case the evaluation is simple, because the rhs consists solely of a *forone* variable declared in the TFR. A *forone* variable means that it can have one and only one value (there are also *foreach* and *collectall* type variables, which are not explained further). The result of this is that the second step in the pattern-matching process returns the rhs-value of the first belief in the previously matched set of beliefs. If this previously matched set is empty the *knownval* function returns *false*, and the precondition fails and the TFR is thus not 'fired'. However, in case there is a matching belief *true* is returned and as a side effect of the pattern matching the variable *energyused* is now bound to the rhs-value of the matched belief. The variable stays bound to this value for the duration of the TFR execution, and can thus be used in subsequent TFR statements, such as in the *conclude* statements.

Every precondition in the when-part of the TFR is evaluated, as long as the previous precondition returns true. If one of the preconditions evaluates to false the TFR is abandoned and the do-part is not executed. Thus, in conclusion, when the agent has one or more beliefs that are matching all the preconditions, the TFR is immediately executed. Using this approach we can represent the forward-reasoning behavior of an agent; the *conclude* statement in one TFR can trigger the execution of a subsequent TFR, thus creating a 'forward chaining' of belief-set changes simulating the reasoning behavior of a person. Every time the agent gets a new belief, only those TFRs are evaluated that have a precondition that is a potential match on the newly created belief. This makes the reasoning behavior efficient, because at every belief change event in an agent only a small number of preconditions have to be evaluated [49].

The Activity Model

The activity model consists of the possible activity behavior for an agent. This is the heart of a Brahms model, because modeling work practice is about the representation of people's activity behavior over time, and performing these activities based on their beliefs. The activity model consists of two elements, *activities* and *workframes*. We explain these two important Brahms concepts using the source code in Excerpt 4 as the example. The activitiy and workframe from Excerpt 4 is from the *ScienceOperationsTeam* group. The source code specifies a group member's behavior during the rover activity of 'finding water-ice in a specific crater on the Moon.' As mentioned before, there are two parts to the encoding of such behavior. First, we need to encode what a science operations team member does (i.e. what activities he or she is engaged in) while the rover is in the activity of finding water-ice in a crater. Secondly, we need to specify when this is done. In the Brahms language the first part is encoded in a *composite-activity*, while the second part is encoded in a *workframe*; a similar production rule-like construct as a *thoughtframe*.

Workframes

In Excerpt 4, there are two workframes shown: a 'high-level' workframe called *wf_SearchForWaterIce* (at the end of the excerpt), and a workframe part of the *FindingWaterIce* activity called *wf_WaitingForData*. Workframes work similar as thoughtframes, but the important difference is that workframes allow for the execution of activities. While thoughtframes represent an agent's reasoning, a workframe represents an agent's activity execution *constraint*. Since activities take time, a major difference between a workframe and a thoughtframe is that a thoughtframe does not take any simulation time, while a workframe has a duration based on the time that the activity takes (see explanation of activities bellow). Workframes 'fire' according to the same pattern-matching process explained for thoughtframes (see section The Knowledge Model). Thus, workframe preconditions are tested in the same way as thoughtframe preconditions and workframe variables are bound in the same way. The body of a workframe (i.e. the do-part) can have *conclude* statements, similar to thoughtframes, however the body of workframes can also contain *activity* calls. *Conclude* statements in workframes are meant to represent the belief-state of the agent in relation to the activity that is going to be executed (i.e. before the activity call) or has finished executing (i.e. after the activity call), and are not meant to represent reasoning of the agent (for this we use thoughtframes).

Excerpt 4. Partial Activity Model for the ScienceOperationTeam group

```
composite_activity FindingWaterIce (Crater crater, int pri) {
   priority: pri;

   activities:
         primitive_activity WaitingForData( ) {
                 max_duration: 3600;
          } //end activity
          …
   workframes:
          workframe wf_WaitingForData {
                 repeat:true;
                 priority: 0;
                 detectables:
                        detectable ReceiveHydrogenData {
                         detect((VictoriaRover.nextSubActivity = DoDrilling))
                         then abort;
                        } //end detectable
                        …
                 when (knownval(current.nextSubActivity = WaitForData))
                 do {
                        WaitingForData( );
                 } //end do
          } //end workframe
          …
   thoughtframes:
          …
} //end composite_activity

workframe wf_SearchForWaterIce {
   repeat: false;
   variables:
          foreach(Crater) rover-loc

   when (knownval(VictoriaRover.currentActivity = SearchForWaterIce) and
         knownval(VictoriaRover.location = rover-loc))
   do {
          conclude((current.currentActivity = SearchForWaterIce));
          FindingWaterIce(rover-loc, 0);
   } //end do
} //end workframe
```

One way of thinking about the role of workframes is to view them as constraints on when an agent can perform an activity. Workframe (WFR) *wf_SearchForWaterIce* constrains when the agent can perform the *FindingWaterIce* activity. The constraints are represented as the preconditions of the workframe. The preconditions encode what beliefs the agent needs to have in its belief-set to enable it to perform the activity or activities (there can be more than one activity call in the workframe body). In plain English *wf_SearchForWaterIce* says: ‘When I believe that the *VictoriaRover* is currently in the activity *SearchForWaterIce* and I believe that the *VictoriaRover* is currently located in a crater, first bind the name of the crater to the variable *rover-loc*, then execute the workframe body with priority zero’ (Brahms allows for parallel execution of worframes, but uses a ‘time-sharing’ approach using priorities, see Activities section for explanation). Note also that *wf_SearchForWaterIce* has the *repeat:false* statement at the top. This means that this workframe will only fire once for a particular set of beliefs that match all its preconditions (a WFI-context). The result is that the agent will only execute *wf_SearchForWaterIce* once for any crater the *VictoriaRover* visits.

When the agent’s inference engine has determined that the preconditions of *wf_SearchForWaterIce* are satisfied (due to finding matching beliefs in the agent’s belief-set) and it is the WFR with the highest priority, the agent will start executing the first statement in the body of the WFR, which in Excerpt 4 is the *conclude* statement that creates the belief for the agent that says that its current activity is *SearchForWaterIce*. This represents that the agent knows that it is currently in the activity of searching for water ice. Next, the engine calls the activity *FindingWaterIce*. We next explain how this works. It should first be said that what has been explained, i.e. from matching of beliefs to preconditions, to binding variables and firing the workframe, executing the *conclude* statement and calling the activity *SearchForWaterIce*, is all done in the same simulation time-event. Thus, although these processes take actual CPU time, they don't take any simulation time for the agent.

Activities

Activities are the most important construct in the Brahms language. All agent behavior has to be modeled as an activity. There are three different types: primitive, composite and Java activities. All activities have a user-defined name representing a behavior defined by the modeler. According to our theory of activities [33], the name of an activity should be the name of an observed behavior of a person in the real-world that the agent represents, but there is no rule in Brahms that states that the agent has to represent a person and that this has to be a person in the real world. It is the responsibility of the modeler to decide the relevance of the model to the system behavior that is being modeled. This allows the use of the Brahms language in any domain and for any purpose, including, but not restricted to, modeling social phenomena, human behavior, and software agent behavior. In Excerpt 4, the name of the activity that is called in WFR *wf_SearchForWaterIce* is *SearchForWaterIce*. It represents what the agent—a member of the science team—is doing while the rover is searching for water ice in a crater on the Moon.

Activities can have parameters that are passed as bounded variables into the activity when it is called in a worframe. In WFR *wf_SearchForWaterIce* the parameter-values that are passed are the value of the *rover-loc* variable, bound in the precondition as the *crater* in which the rover is searching for water ice, and an activity *priority* value of zero.

The activity *SearchForWaterIce* called in the WFR *wf_SearchForWaterIce* is declared at the top of Excerpt 4. This activity is of type *composite_activity*. Composite activities are activities that are *decomposed* into lower-level subactivities, workframes and thoughtframes. Excerpt 4 shows only a partial implementation of the *SearchForWaterIce* activity. It shows the declaration of one subactivity called *WaitingForData*, and one workframe called *wf_WaitingForData*. Activity *WaitingForData* is a *primitive_activity* type. A primitive activity is an activity that is not further decomposed. It can be used to represent an *operation* (as in activity theory) or an *action* in the world that is not further decomposed. Primitive activities have a specified maximum or a random duration. This is different from a *composite_activity* in that it has a pre-specified duration. In contrast, the duration of a composite activity depends on the duration of the subactivities executed within it (note that thoughtframes have no duration).

Primitive activity duration is determined at the start of its execution—either randomly chosen between a given min-max duration interval, or given as a max duration—but is not necessarily the actual duration of the activity. The actual duration of an activity depends on the state of the workframe instance[3] (WFI) in which the activity is being called. Each WFI is in one of the states shown in Figure 5. The state of an agent's activity behavior is defined by the combined sets of available, working, interrupted, and interrupted-with-impasse WFIs at any moment in time.

---

[3] When a workframe (or thoughtframe) is fired (i.e. the preconditions are matched against beliefs in the agent's belief-set) a workframe instance is created for every workframe variable context that matches all preconditions. Each workframe instance is now an independent version of the workframe and will be executed independently from each other, with different variable bindings (determined by the WFI-context).

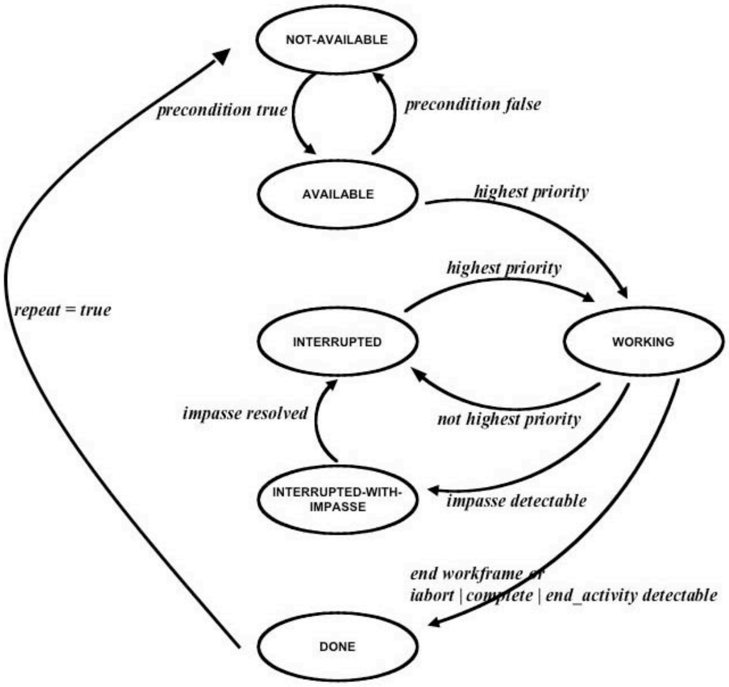


Figure 5. State transition diagram for workframe instances

There can only be one *current* activity for an agent. The time an activity has been active can only change when the activity is the current activity. Therefore, when an activity is in a *non-active* state its active time is *not* increasing, although simulation time is always increasing. Which activity is the current activity depends on which WFI is in the *working* state and the execution of the WFI-body.

There are different ways a WFI can change state. One way is through the use of priorities. Every time a workframe fires the created WFIs receive a priority, based on the prioriy of the workframe, if given, or the highest priority of the activities called within the workframe body. The default priority is always zero. The agent's inference engine determines which of the *available, working* and *interrupted* WFIs have the highest priority. This one is moved to the *working* state. Every time a new WFI becomes available, there exist the potential that the *working* WFI is interrupted by a higher-priority WFI. In that case the current working instance is moved to the *interrupted* state, and the new instance with the highest priorities is moved to the *working* state, and thus becomes the current WFI the agent is executing.

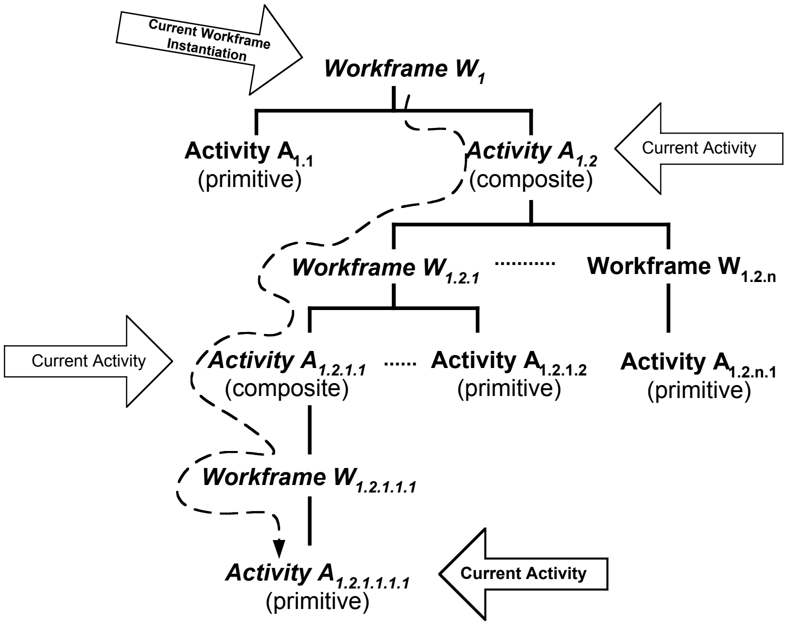
There are other ways for an activity to change from a *working state*. The state change described above is based on other 'independent' workframes firing. However, a WFI can change its own state. The default way for a WFI to change its *working* state is when the body is finished executing. At that moment the WFI automatically moves from the *working* state to the *done* state and there it gets deleted, or moved to the *not-available* state if the repeat-clause is set to *true*. However, there are other state-changing events that can be represented inside a workframe. This is done using a *detectable*. Excerpt 4 shows the declaration of the *ReceiveHydrogenData* detectable. A detectable defines that if the agent detects a *fact* in the world this fact becomes a belief of the agent. The belief is then matched to the *detect* condition in the detectable. If the agent has a belief that matches the condition the body of the detectable is executed. The body of a detectable can contain one specific action: *abort*, *complete, impasse* or *continue*. The *ReceiveHydrogenData* detectable specifies an *abort* action. The detectable says that if the agent gets a belief (either through the detection of a fact in the world, or through other means) that the

VictoriaRover's next subactivity is to drill in the lunar surface, it will abort the *working* workframe, which means it will end the activity *WaitingForData*.

The actual behavior of the agent is thus dependent on which of its workframes fire, and when. Firing of workframes depends on the beliefs of the agent at every moment in time. The beliefs in the belief-set of the agent depend on the initial-beliefs, conclude statements in thoughtframes and workframes that fire, communication with other agents (see section The Communication Model), and detection of facts in the world. The behavior of the agents is therefore situation specific and it is not only dependent on its internal reasoning (using thoughtframes), but also determined by the interaction of the agent with other agents and with the modeled environment. We refer to the Brahms modeling paradigm as a *situated activity paradigm*.

Activity Subsumption Architecture

An important aspect of the Brahms *activity paradigm* is that activities are *not* the same as functions and procedures in imperative languages [50]. Imperative languages use a computer memory-based program stack to keep track of function calls. When a function is executed, the function's context is 'popped' onto the program stack. When in a subfunction the program is not also still in the context of the 'parent' function. Thus the program cannot move execution back and forth between a function and its subfunctions that are called. Function execution is sequential and cannot be interrupted. Not so with activities. In contrast, Brahms activities stay active while they are being executed.



Current Workframe Instantiation
Workframe W_1
Activity $A_{1.1}$ (primitive)
Activity $A_{1.2}$ (composite)
Current Activity
Workframe $W_{1.2.1}$
Workframe $W_{1.2.n}$
Current Activity
Activity $A_{1.2.1.1}$ (composite)
Activity $A_{1.2.1.2}$ (primitive)
Activity $A_{1.2.n.1}$ (primitive)
Workframe $W_{1.2.1.1.1}$
Activity $A_{1.2.1.1.1.1}$ (primitive)
Current Activity

Figure 6. Workframe-Activity hierarchy

Thus, if a subactivity in a workframe of a composite activity gets executed, the 'parent' composite activity is still active (see Figure 6). All workframes, thoughtframes and detectables in the 'parent' activity are still being evaluated while the agent is executing the subactivity. This is part of the Brahms *subsumption* architecture [51], and is based on the principle that humans are always multi-tasking by being in a hierarchy of activities at the same time. For example, the science team member from Excerpt 4 is also still in the activity of finding water ice when it is in the activity of waiting for data to be returned by the rover. Thus, every workframe, thoughtframe or detectable in the current *activity* *hierarchy* is part of the agent's context, and can be fired at any moment, changing the belief and behavioral state of the agent.

In a Brahms simulation, an agent may engage in multiple activities at any given time, but only one activity in one workframe is active at any one time. At each event the simulation engine determines which workframe should be selected as the *current working*, based on the priorities of available, current and interrupted work (see Figure 5). The state of an interrupted or impassed workframe is saved, so that the agent will continue an interrupted workframe with the activity that it was performing at the moment it was interrupted.

An important consequence and benefit of this subsumption architecture is that all of the workframes of a model are simultaneously competing and active, and the selection of a workframe to execute is made *without* reference to a program stack of workframe execution history.

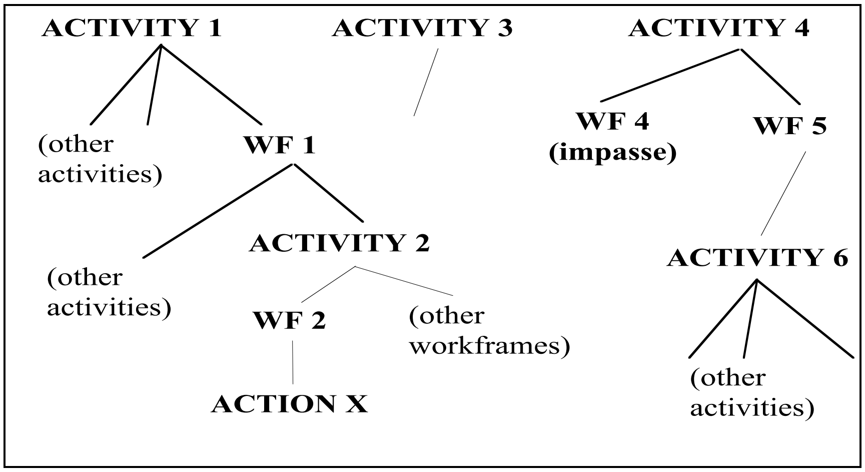


Figure 7. Multi-tasking in Brahms

An illustration of this is given in Figure 7. An agent (not shown) in a running model may have multiple competing general activities in process: Activities 1, 3, and 4. Activity1 has led the agent (through workframe WF1) to begin a subactivity, Activity 2, which has led (through workframe WF2) to a primitive activity Action X. When Activity 2 is complete, WF1 will lead the agent to do other activities. Meanwhile, other workframes are competing for attention in Activity 1. Activity 2 similarly has competing workframes. Priority rankings led this agent to follow the path to Action X, but interruptions or reevaluations may occur at any time. Activity 3 has a workframe that is potentially active, but the agent is not doing anything with respect to this activity at this time. The agent is doing Activity 4, but reached an impasse in workframe WF4 and has begun an alternative approach (or step) in workframe WF5. This produced a subactivity, Activity 6, which has several potentially active workframes, all having less priority at this time than WF2.

The Brahms subsumption architecture allows two forms of *multi-tasking*. The first form is inherent in the activity-base paradigm; Brahms can simulate the reactive situated behavior of humans. An agent's context forces it to be *active* in only one low-level activity. However, at any moment an agent can change focus and start working on another competing activity, while queuing others. Having the simulation engine switch between current and interrupted work for each agent, simulates this type of multi-tasking behavior as represented in Figure 7. The second form is subtler. People can be working concurrently on many high- and medium-level activities in a workframe-activity hierarchy. While an agent can only execute one primitive activity in the hierarchy at a time (e.g. ACTION X in Figure 7), the agent is concurrently within all the higher–level activities in the workframe-activitiy hierarchy. For example, the agent in Figure 7 is concurrently within Activity 1, Activity 2, and primitive activity Action X. It should be noted that while a workframe, and its associated activities are interrupted or impassed, the agent is still considered to be in the activity. The agent is conceptually within all current, interrupted and impassed activities.

Java Activities

A special type of activity is the *Java activity.* A Java activity is a primitive activity that is declared similar as other primitive activities, but is implemented in the Java programming language. Java activities are helpful if the agent or object needs to perform complicated calculations that can easier be done in the Java language, or if the agent needs to interact with systems outside of the Brahms language (As shown in Figure 2, Brahms also allows an agent to be completely written in Java, allowing external programs to be “wrapped” as Brahms agents). The java activity specifies the fully qualified name of the Java class that either implements the *IExternalActivity* interface or extends the *AbstractExternalActivity* class. The interface and class are specified in the Brahms Java Application Interface (JAPI). When the java activity is executed an instance of the class is created and the Java code executed. If the class extends the *AbstractExternalActivity* class, the java code has access—using the JAPI methods available—to the parameters passed into the activity, the belief-set of the agent or object, as well as the fact-set of the world. The java activity is also able to conclude new beliefs and facts, create new agents and objects, as well as communicate with other agents and objects in the Brahms model. In other words, for any built-in activity allowed in the body of a workframe there exists a JAPI method equivalent.

Excerpt 5 gives an example of a Java activity. The *getCurrentTime* Java activity is part of the *CalendarUtil* group and class in the Brahms base library. The calendar utility implements a calendar object that allows agents to deal with the Gregorian calendar and concepts such as ‘yesterday’, ‘tomorrow’, ‘last week’, ‘last month’, et cetera. The implementation of this Java activity is located in the *brahms.base.util.GetCurrentTimeActivity* java class in the Brahms *common.jar* file, which is loaded at the start of the BVM.

Excerpt 5. Java Activity Example

```
/**
 * Creates a new Calendar object and initializes its beliefs with the
 * current date and time based on the currently set or default
 * time zone. In simulated mode the time will be initialized to
 * the simulated date/time unless the timeType is set to REAL_TIME in
 * which case the system time is returned. In (distributed) real-time
 * mode always the actual system time is used regardless of whether the
 * timeType is set to SIM_TIME (a warning will be generated in the vm.log
 * file if SIM_TIME is set when this activity is called when the virtual
 * machine is in (distributed) real-time mode.
 *
 * @param timeType one of SIM_TIME or REAL_TIME to indicate from what clock
 *  the time should be returned, the internal simulation clock or the
 *  system clock.
 * @return out an unbound variable of type Calendar to which to assign
 *  the newly created Calendar object
 */

java getCurrentTime(symbol timeType, Calendar out) {
   class: ‘brahms.base.util.GetCurrentTimeActivity’;
} //end java
```

The Communication Model

One of the most important aspects of modeling human behavior is the interaction with other people and systems. Brahms supports representing human-human communication, as well as human-machine communication using the concept of communication as an activity. The communication model consists of a definition of communication activities between agents and objects. In Brahms, communication is defined as the *transfer of beliefs* between agents and/or objects. Just as in human communication, communicating takes time and is situated in an activity. In order to model human communication we thus have to represent the time it takes to communicate, either between people, systems, or between people and systems. To do this there is a special type of *primitive* activity called a *communication activity*. An agent or object can perform a communication activity like any other primitive activity. However, a communication activity has a 'side effect', namely that when the agent (or object) performs the activity it can *send* (i.e. tell) beliefs to agents (or objects) it is communicating with, or it can *receive* (i.e. ask) beliefs from an agent (or object). There is an obvious catch; an agent (or object) can only communicate beliefs that it has in its belief-set. In other words, agents and objects cannot communicate that what it has no beliefs about.

Modeling work practice of people means that we are interested in modeling how communication actually happens in the real world. Therefore, in Brahms we are able to model communication not as a simple sending or receiving of beliefs between agents or objects (although this is possible), but we represent the complete communication path of the communication. If we model an organization of communicating people we want to represent the communication tools that are used, e.g. the use of e-mail, telephones or faxes. We have gone as far as modeling the operation of the telephone system with voice mail capability. This way we are able to model the fact that communication of information in practice often takes more time then a (abstracted) flow of information between sender and receiver suggests. If a person calls another person who is not available at that moment, the caller might (or might not) leave a voice mail. It will depend on the receiver's activity of listening to his or her voice mail for the content of the message to actually be transferred. To model this, we model the telephones (as objects) and their voice mail capability with activities, the location of telephones (see section The Geography Model), as well as the agents' activities of calling someone via the telephone, the telephone object transferring the communicated beliefs to the receiver's voice mail (in case the receiver is not answering the phone), and the receiver's activity of listening to its voice mail and responding back to the caller if necessary. Obviously, it is not necessary to model communication paths in that much detail if this is not the objective of the simulation effort. We only explain this so that the reader is aware of the possible detail of modeling communication between agents.

Excerpt 6 shows a communication example between members of the *VictoriaTeam* group. In this example the communication model is abstracted to a simple transfer of beliefs, without the complicated model of a communication tool used.

Excerpt 6. Communication Activity Example

```
communicate ComAct_NextRoverActivity(VictoriaTeam rcvr, int pri, int md) {
   priority: pri;
   max_duration: md;
   with: rcvr;
   about:
          send(VictoriaRover.nextActivity = anyvalue),
          send(VictoriaRover.subActivity = anyactivity),
          send(SATM.lengthIntoSurface = anyvalue),
          send(SATM.sampleVolume = anyvalue),
          send(VictoriaRover.gotoLocation = anylocation),
          send(rcvr.transmitCommand = true);
    when: end;
} //end activity

workframe wf_CommunicateDoDrillActivity {
    repeat: true;
    when (knownval(current.nextSubActivity = CommunicateDoDrillActivity))
    do {
          conclude((current.subActivity = CommunicateDoDrillActivity));
          conclude((VictoriaRover.nextActivity =
                   SearchForWaterIceInPermanentDarkAreaActivity));
          conclude((VictoriaRover.subActivity = DrillingActivity));
          //conclude((VictoriaRover.gotoLocation = ShadowAreaInCraterSN1));
          conclude((SATM.lengthIntoSurface = 10.0)); //drill 10cm deep
          conclude((SATM.sampleVolume = 1.0)); //take a 1.0 cc sample
          conclude((VehicleAndSpacecraftOpsTeam.transmitCommand = true));
          ComAct_NextRoverActivity(VehicleAndSpacecraftOpsTeam, 100, 10);
          conclude((current.nextSubActivity = WaitForDataActivity));
   } //end do
} //end workframe
```

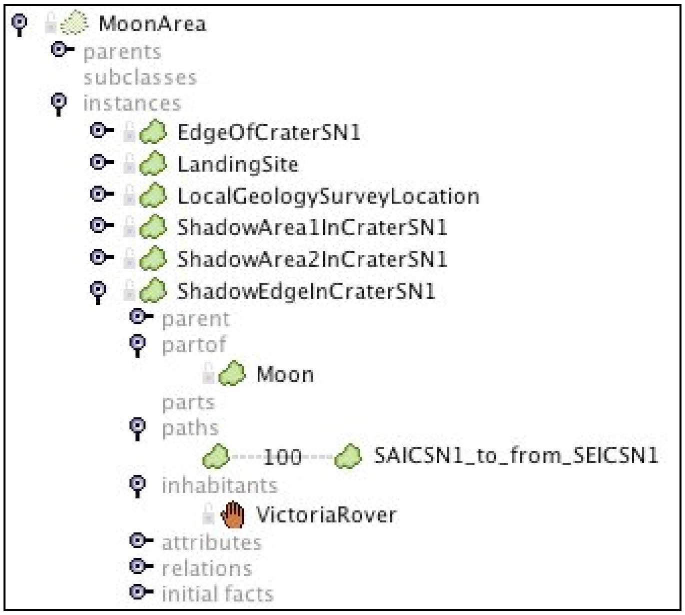
By now the reader should be familiar enough with the Brahms language concepts of *workframe*, *preconditions, conclude* commands and *activity calls* that we will not explain this again. However, we explain the new properties of the communication activity *ComAct_NextRoverActivity* in Excerpt 6. Every communication activity has a *with* property. This property declares with which agents or objects the communication is held. In the example, the value of the *with* property is the *rcvr* parameter. This parameter is of type *VictoriaTeam*, which means it can thus have one or more agents that are a member of *VictoriaTeam* as its bounded value. If we look in WFR *wf_CommunicateDoDrillActivity* we see that the *rcvr* parameter is bound to the agent *VehicleAndSpacecraftOpsTeam* (see the *ComAct_NextRoverActivity* activity call in the body of the workframe). In this example, the communication receiver is one agent that represents a whole team of people, which means the model is not concerned with the detail of individual agents and their use of communication tools.

The next important communication activity property is the *about* property. This property specifies the possible content of the communication. We specifically say 'possible content', because as mentioned an agent can only communicate those beliefs it actually has. Therefore, if during the execution of the *ComAct_NextRoverActivity* the performing agent does not have any of the beliefs that match the send transfer definition in the about property, the belief–transfer cannot take place. An example of this is actually shown in the body of the WFR in Excerpt 6. Note that one of the conclude statements before the *ComAct_NextRoverActivity* call has been commented out (using the single line comment symbol '//'). This means that the agent will not get the belief about the *gotoLocation* attribute of the *VictoriaRover*. Thus, when in the communication activity the transfer definition for sending this belief to the receiver is executed, no belief-match is found and thus this belief-transfer will not take place.

The last important property for communication activities is the when property. This property can have two values, *start* or *end*. This property specifies when during the activity the beliefs are actually transferred. Imagine we want to model that the communication of a message takes some time, and we don't want the receiver to act on this communication until the end of the communication activity. In that case we would model this using a *when:end* value. On the other hand, if we want to model that the receiver acts on the message during the communication activity we would model this with a *when:start* value. At this moment it is not possible to specify other values for the when property, we can therefore not model the actual transfer of beliefs using some kind of timed distribution of actual transfer of beliefs during the activity. To model this we would have to represent it as a number of sequential communication activities, separating out the ‘conversation’.

The Geography Model

The last conceptual model that is part of a complete Brahms model is the Geography Model. The Geography Model consists of the definition of places (a.k.a. locations) where agents and objects can be placed. The first worldview in modeling work practice states that our actions take place in the world, so-called located or situated action. We thus include a model of the world in order to represent the fact that all our activities are located. A Brahms model that represents located behaviors must at least include those locations where modeled activity takes place. It should be noted that there is no rule that says that a Brahms model has to be about located behaviors, and it is thus perfectly fine to use Brahms to model more abstract behaviors and leave out the geography model. In Brahms we model places conceptually. This means that we do not model geography as a three-dimensional virtual reality model. Instead, we model the geography in terms of special type objects, called *areas,* representing a hierarchical organization of locations corresponding to locations in the modeled world. Figure 8 presents a partial geography of the Victoria model.



```
MoonArea
  parents
  subclasses
  instances
    EdgeOfCraterSN1
    LandingSite
    LocalGeologySurveyLocation
    ShadowArea1InCraterSN1
    ShadowArea2InCraterSN1
    ShadowEdgeInCraterSN1
      parent
      partof
        Moon
      parts
      paths
        ----100---- SAICSN1_to_from_SEICSN1
      inhabitants
        VictoriaRover
      attributes
      relations
      initial facts
```

Figure 8. Partial Geography Model of the Moon

Areas are instances of classes called *AreaDefinitions* (areadefs). Figure 8 shows the *MoonArea* areadef. As with any other type of object or class, areas and areadefs can have attributes and be hierarchically organized. Using the area attributes, agents can have beliefs about areas (e.g. the temperature in a Building area). The *MoonArea* areadef has six instances (i.e. area objects).

Location facts and beliefs

Each area can have a number of relationships associated with it: *parent, partof*, *parts*, *paths* and *inhabitants*. As shown in Figure 8, the *ShadowEdgeInCraterSN1* has the following relationships; First off, this area is *part of* the *Moon* area. This means that the crater area is located on the Moon. The *partof* relation is important for the localization of agents and objects. That is, when an agent or object is located in an area (i.e. is an inhabitant), it is automatically also located in the area of which this area is part. For example, Figure 8 shows that the *VictoriaRover* agent is an *inhabitant* of area *ShadowEdgeInCraterSN1*. *ShadowEdgeInCraterSN1* is *part of* the *Moon* area, and thus the *VictoriaRover* is both located in the *ShadowEdgeInCraterSN1* area and in the *Moon* area.

Agent and object location has a special semantics in Brahms. When an agent or object is located in an area (i.e. is an inhabitant), a number of facts and beliefs are automatically kept track of by the simulation engine. First, localization is a fact in the world and the engine automatically generates a so-called *location* fact for each inhabitant. Thus, for the model in Figure 8 the engine generates the following fact: *(VictoriaRover.location = ShadowEdgeInCraterSN1)*. Second, every agent that is an inhabitant of an area receives the location facts of itself and all co-inhabitants of that area as beliefs. This means that the default is that every agent will know which other agents and objects are also located in its area. This engine behavior is dynamic, which means that if an agent moves from one area to another area the engine first retracts the current location fact, and asserts the new location fact. Next, the agents still located in the from-area have the location belief of the moved agent negated, so that they realize the agent has moved. The moved agent gets its new location as a new belief, thus overriding its old location belief. Next, all agents that are already inhabitants of the new location will receive the location of the newly arrived agent as beliefs. In short, agents and objects can only be in one location at a time, location of agents and objects are facts in the world, agents always know where they are and also always know which other objects and agents are in the same location.

Movement

As mentioned above, agents and objects can move between areas. There are two important notions about representing movement that need to be kept in mind, moving with a specific duration and moving along a defined path. Moving is an activity that takes time. There is a special activity type called a *move* activity. Excerpt 7 shows an example of a Rover agent's ability to execute the *TraverseToLocation* activity in a workframe. Moving is constraint, similar to any other activity, by the beliefs of the agent matching the precondition of a workframe calling a move activity.

Excerpt 7. Rover moving activity example

```
move TraverseToLocation(int pri, int md, MoonArea loc) {
   priority: pri;
   max_duration: md;
   location: loc;
} //end move

workframe wf_TraverseToLocationInShadowArea {
   repeat:true;
   variables:
         forone(MoonArea) loc;
         forone(int) drivingtime;
   when (knownval(current.gotoLocation = loc) and
        not(current.location = loc) and
        knownval(current.nextActivity = MoveToLocationActivity) and
        knownval(current.drivingTime = drivingtime) and
   do {
         conclude((RoverBattery.initialize = true));
         conclude((current.currentActivity = MoveToLocationActivity));
         TraverseToLocation(100, drivingtime, loc);
         CommunicateToEarthTeam(100);
   } //end do
} //end workframe
```

Excerpt 7 shows an example of movement with a specific duration. When the preconditions of the WFR *wf_TraverseToLocationInShadowArea* match the beliefs of the agent *VictoriaRover*, the rover calls the move activity *TraverseToLocation* with as parameters the duration of the move activity (i.e. the value of the variable *drivingtime*) and the location to which the rover needs to move (i.e. the value of the variable *loc*). In this example the rover moves from its current location to the *gotoLocation* in the given time *drivingtime* (as long as the rover is not already in the *gotoLocation*). This workframe represents the rover's generic capability to execute a command driving to a location with a certain speed (speed is modeled as an implicit calculation based on time and distance).

Another way of modeling movement duration of an agent or object is by pre-specifying the paths that can be taken from one location to another. *Paths* are objects that specify two end locations (i.e. *area* objects) and duration. Figure 8 shows that the geography model specifies that there exists a path *SAICSN1_to_from_SEICSN1* between the areas *SunlitAreaInCraterSN1* and *ShadowEdgeInCraterSN1*, and that the distance (specified in time) is a 100 simulation clock-ticks. With a clock grain-size of 1 second and a rover speed of 1 m/sec, the length of the path is 100 meters. Excerpt 8 shows the declaration of this path in source code.

Excerpt 8. Path declaration from Figure 8

```
path SAICSN1_to_from_SEICSN1 {
    area1: SunlitAreaInCraterSN1;
    area2: ShadowEdgeInCraterSN1;
    distance: 100; //100m, when rover speed in shadowed zones is 1 m/sec
}
```

The use of *paths* in move activities goes as follows: when an agent or object performs a move activity without a duration and there is a *path* defined from the current location to the 'move to' location, the duration of the move is taken from the duration of the path. In case of multiple possible paths the engine calculates the shortest route and uses that as the duration of the move activity.

Discussion

After the detailed description of the Brahms language, its human behavior modeling capabilities and the workings of the simulation engine, we now turn to the use of Brahms as a modeling and simulation tool for the organizational process simulation community.

The Brahms tool was originally developed to model work processes at the work practice level to include the 'social systems of work' in a simulation of work process in human organizations. Our research over the past decade has shown that, discounting the difficulties of modeling human behavior with all the representational limitations, Brahms allows the detailed modeling of work practice at a level of detail that enables a) researchers to get insight into the way people actually work in an organization and b) system developers to use these models to develop computer software that, at a minimum, has a better representation of the user and its environment. In this paper we are trying to persuade modeling and simulation researchers that the Brahms language is suitable for studying kinds of social and work practice phenomena of interest to the organizational process simulation community. Our experience and results in modeling work practice makes us believe that larger social phenomena can also be modeled. The Brahms modeling language has great advantages for the researcher, because compared to existing tools, such as Swarm, the language allows for a more 'natural' representation of human behavior at the level of activities, reasoning, communication, interaction with objects and movement in the world (a level we might call the meso-level of human systems [52]). We believe that when the objective is to model human behavior at an individual agent level in order to analyze or predict organizational behavior at the macro system level, the best way to do this is to use a representational language that is geared towards representing human interaction that is based on theories of activity behavior of people.

We would argue that imperative programming languages that are suited to model macro-level system behavior using an agent design paradigm, are not flexible enough to claim a correspondence with actual human behavior, whereas cognitive architectures that are suited to model single agent cognitive behavior, based on a theory about how the brain actually stores and processes information, are too detailed to conveniently model human behavior at the level of agent interaction with other agents and the world (e.g. the simulation clock grain-size in both Soar and ACT-R is in the 100msec range). Brahms on the other hand, is a language that falls in between these two types of modeling languages. Brahms is a language that allows for an easy representation of agent behavior at the micro-level (i.e. reasoning behavior, without the brain correspondence claim) and meso-level (human interaction with each other and the world), allowing the researcher to show the effects of these behaviors at the macro-level (i.e. the organizational process or system level).

The Brahms environment is completely developed in the Java language and, with its language compiler, Brahms virtual machine, development environment (the Composer), and its simulation display environment (the AgentViewer), is freely available from the internet for research purposes (URL: http://www.agentisolutions.com).

References

1. Clancey, W.J., et al. *The Mobile Agents Integrated Field Test: Mars Dessert Research Station 2003*. in *FLAIRS 2004*. 2004. Miami Beach, Florida.
2. Clancey, W.J., et al. *Advantages of Brahms for Specifying and Implementing a Multiagent Human-Robotic Exploration System*. in *The 16th International FLAIRS Conference 2003*. 2003. St. Augustine, Fl.
3. Sierhuis, M., A. Acquisti, and W.J. Clancey. *Multiagent Plan Execution and Work Practice: Modeling plans and practices onboard the ISS*. in *3rd International NASA Workshop on Planning and Scheduling for Space*. 2002. Houston, TX.
4. Acquisti, A., et al. *Agent Based Modeling of Collaboration and Work Practices Onboard the International Space Station*. in *11th Computer-Generated Forces and Behavior Representation Conference*. 2002. Orlando, Fl.
5. Sierhuis, M. *Modeling and Simulation of Work Practices on the Moon*. in *Computational Analysis of Social and Organizational Systems 2000*. 2000. Carnegie Mellon University, Pittsburgh, PA: http://www.ices.cmu.edu/casos/papers_content.html.
6. Sachs, P., *Transforming Work: Collaboration, Learning, and Design.* Communications of the ACM, 1995. **38**(9): p. pp. 36-44.
7. Button, G. and R. Harper, *The Relevance of 'Work-Practice' for Design.* Computer Supported Cooperative Work, 1996. **1996**(4): p. 263-280.
8. Clancey, W.J., et al., *Brahms: Simulating practice for work systems design.* International Journal on Human-Computer Studies, 1998. **49**: p. 831-865.
9. Clancey, W., J., *Situated Cognition: On Human Knowledge and Computer Representations*. 1997: Cambridge University Press.
10. Leont'ev, A.N., *Activity, Consciousness and Personality.* 1978, Englewood Cliffs, NJ: Prentice-Hall.
11. Vygotsky, L.S., *Mind in Society: The Development of Higher Psychological Processes*, ed. M.

Cole, et al. 1978, Cambridge, MA: Harvard University Press.

12. Suchman, L.A., *Plans and Situated Action: The Problem of Human Machine Communication*. 1987, Cambridge, MA: Cambridge University Press.
13. Lave, J., *Cognition in Practice*. 1988, Cambridge, UK: Cambridge University Press.
14. Laird, J.E., A. Newell, and P.S. Rosenbloom, *Soar: An architecture for general intelligence.* Artificial Intelligence, 1987. **33**: p. 1-64.
15. Anderson, J.R. and C. Lebiere, *The atomic components of thought*. 1998, Mahwah, NJ.: Lawrence Erlbaum Associates.
16. Sierhuis, M., *Modeling and Simulating Work Practice; Brahms: A multiagent modeling and simulation language for work system analysis and design*. Ph.D. Thesis. Social Science Informatics (SWI). 2001, Amsterdam, The Netherlands: University of Amsterdam, SIKS Dissertation Series No. 2001-10. 350.
17. Wooldridge, M. and N.R. Jennings, *Intelligent Agents: Theory and Practice.* Knowledge Engineering Review, 1995. **10**(2): p. 115-152.
18. Emery, F.E. and E.L. Trist, *Socio-Technical Systems*, in *Management Sciences, Models and Techniques*, C.W. Churchman, Editor. 1960, Pergamon: London.
19. Pava, C.H.P., *Managing New Office Technology: An Organizational Strategy.* 1983, New York: The Free Press.
20. Weisbord, M.R., *Productive Workplaces: Organizing and Managing for Dignity, Meaning, and Community.* 1987, San Francisco, CA: Jossey-Bass Inc., Publishers.
21. Ehn, P., *Work-Oriented Design of Computer Artifacts*. 1988, Stockholm, Sweden: Arbetslivcentrum.
22. Greenbaum, J. and M. Kyng, eds. *Design at Work: Cooperative design of computer systems*. 1991, Lawrence Erlbaum: Hillsdale, NJ.
23. Clancey, W.J. *Human Exploration Ethnography of the Haughton-Mars Project 1998-99*. in *Mars Society Annual meeting*. 1999. Boulder, CO.
24. Clancey, W.J., *Field Science Ethnography: Methods for Systematic Observation on an Expedition.* Field Methods, 2001. **13**(3): p. p. 223-243.
25. Sierhuis, M. and W.J. Clancey, *Modeling and Simulating Work Practice: A human-centered method for work systems design.* IEEE Intelligent Systems, 2002. **Volume 17(5)**(Special Issue on Human-Centered Computing).
26. Winograd, T. and F. Flores, *Understanding Computers and Cognition*. 1986, Menlo Park, CA: Addison-Wesley Publsihing Corporation.
27. Kuutti, K., *Activity theory as a potential framework for human-computer interaction research*, in *Context and Consciousness: Activity Theory and Human-Computer Interaction*, B.A. Nardi, Editor. 1996, The MIT Press: Cambridge, MA. p. 17-44.
28. Sierhuis, M., et al. *Human-Agent Teamwork and Adjustable Autonomy in Practice*. in *The 7th International Symposium on Artificial Intelligence, Robotics and Automation in Space (i-SAIRAS)*. 2003. Nara, Japan.
29. Polanyi, M., *The Tacit Dimension*. 1983, Magnolia, MA: Peter Smith.
30. Lave, J., M. Murtaugh, and O.d.l. Roche, *The Dialectic of Arithmetic in Grocery Shopping*, in *Everyday Cognition: Its Development in Social Context*, B. Rogoff and J. Lave, Editors. 1984, Harvard University Press: Cambridge, MA.
31. Lave, J. and E. Wenger, *Situated Learning - Legitimate peripheral participation*. 1991: University Press.
32. Nardi, B.A., *Context and Consciousness: Activity Theory and Human-Computer Interaction*.

1996, Cambridge, MA: The MIT Press.

33. Clancey, W.J., *Simulating Activities: Relating Motives, Deliberation, and Attentive Coordination.* Cognitive Systems Research, 2002. **3**(3): p. 471-499.
34. Hutchins, E., *Cognition in the Wild*. 1995, Cambridge, MA: MIT Press.
35. Hutchins, E., *How a Cockpit Remembers Its Speeds.* Cognitive Science, 1995. **19**: p. 265-288.
36. Clancey, W.J., *Model construction operators.* Artificial Intelligence, 1992. **53**(1): p. 1-124.
37. Newell, A., *Unified theories of cognition*. 1990, Cambridge, MA: Harvard University Press.
38. Sierhuis, M., *Modeling and Simulating Work Practice; Brahms: A multiagent modeling and simulation language for work system analysis and design*, in *Social Science Informatics (SWI)*. 2001, University of Amsterdam, SIKS Dissertation Series No. 2001-10: Amsterdam, The Netherlands. p. 350.
39. van Hoof, R. and M. Sierhuis, *Brahms Language Reference*. 2000, http://www.agentisolutions.com/documentation/language/ls_title.htm.
40. Cohen, P.R. and H.J. Levesque, *Intention is choice with commitment.* Artificial Intelligence, 1990. **42**: p. 213-261.
41. Rao, A.S. and M.P. Georgeff. *Modeling rational agents within a BDI-architecture*. in *Proceedings of Knowledge Representation and Reasoning (KR&R-91)*. 1991. San Mateo, CA: Morgan Kaufmann Publishers.
42. Minar, M., R. Burkhart, and C. Langton, *Swarm Development Group*. 1996.
43. Luna, F. and A. Perrone, eds. *Agent-Based Methods in Economics and Finance: Simulations in Swarm*. Advances in Computational Economics, ed. A.N. Hans Ammam. 2002, Kluwer Academic Publishers: Norwell, MA.
44. Terna, P., *simulation Tools for Social Scientists: Building Agent Based Models with SWARM.* Journal of Artificial Societies and Social Simulation, 1998. **1**(2): p. available at http://www.soc.surrey.ac.uk/JASS/1/2/4.html.
45. Sierhuis, M., et al., *Modeling and Simulation for Mission Operations Work System Design.* Journal of Management Information Systems, 2003. **Vol. 19**(No. 4): p. 85-129.
46. Kernighan, B.W. and D.M. Ritchie, *The C Programming Language*. 2nd ed. 1988, Engelwood Cliffs, NJ: Prentice Hall.
47. Dennett, D.C., *The Intentional Stance*. 1987, Cambridge, MA: Bradford Book, MIT Press.
48. Konolige, K., *A Deduction Model of Belief.* 1986, San Mateo, CA: Morgan Kaufmann.
49. Forgy, C.L., *Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem.* Artificial Intelligence, 1982(19): p. 17-37.
50. Pratt, T.N. and M.Z. Zelkowitz, *Programming Languages: Design and Implementation*. 3rd. ed. 1996, Englewood Cliffs, NJ.: Prentice Hall.
51. Brooks, R., A., *Intelligence without representation.* Artificial Intelligence, 1991. **47**: p. 139-159.
52. Carley, K.M. and M.J. Prietula, *ACTS Theory: Extending the Model of Bounded Rationality*, in *Computational Organization Theory*, K.M. Carley and M.J. Prietula, Editors. 1994, Lawrence Erlbaum Associates: Hillsdale, NJ.