Brahms: a multi-agent modelling environment for simulating work processes and practices

Maarten Sierhuis*

Research Institute for Advanced Computer Science,
NASA Ames Research Center,
Moffett Field, CA 94035-1000, USA
E-mail: Maarten.Sierhuis-1@nasa.gov
*Corresponding author

William J. Clancey

Florida Institute for Human and Machine Cognition,
NASA Ames Research Center,
Moffett Field, CA 94035-1000, USA
E-mail: William.J.Clancey@nasa.gov

Ron J.J. van Hoof

QSS Group, Inc., NASA Ames Research Center,
Moffett Field, CA 94035-1000, USA
E-mail: rvanhoof@mail.arc.nasa.gov

Abstract: Modelling and simulating work processes is often done at such an abstract level that individual work practice – collaboration, communication, 'off-task' behaviours, 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 organisation actually gets done impossible. This paper describes the Brahms modelling and simulation environment, developed at NASA Ames Research Center. The Brahms modelling language is geared towards modelling people's activity behaviour, making it an ideal environment for simulating organisational processes at a level that allows the analysis of the work practice and designing new work processes at the implementation level.

Keywords: multi-agent; language; simulation; work practice; business process modelling.

Reference to this paper should be made as follows: Sierhuis, M., Clancey, W.J. and van Hoof, R.J.J. (2007) 'Brahms: a multi-agent modelling environment for simulating work processes and practices', *Int. J. Simulation and Process Modelling*, Vol. 3, No. 3, pp.134–152.

Biographical notes: Maarten Sierhuis received his PhD in Social Science Informatics at the University of Amsterdam, The Netherlands and a BSc in Informatics from the The Hague Polytechnic, The Netherlands. He is currently Senior Research Scientist at RIACS/NASA Ames Research Center, Moffett Field, CA. His current research interests include multi-agent modelling and simulation and human-agent interaction.

William J. Clancey is Chief Scientist, Human-Centred Computing in the Intelligent Systems Division at NASA Ames Research Center. He received his PhD in Computer Science from Stanford University, CA and a BA in Mathematical Sciences from Rice University, Houston, TX. His current research focuses on integrating people, systems, and simulations.

Ron J.J. van Hoof is a Software Engineer for QSS Group, Inc. consulting for NASA Ames Research Center. He received his Masters Degree in Knowledge Engineering from the University of Middlesex, London, UK, and a BSc in Computer Science from the Hogeschool West-Brabant, The Netherlands. His current responsibilities include the maintenance and development of the Brahms language, virtual machines and an agent-based exploration support system.

1 Introduction

Since the mid-1990s, the start of Business Process Reengineering (BPR), modelling of business processes has become more widespread. It has been argued that some of the early failures in BPR were due to an inability to evaluate and predict the impact of design changes in a work process (Hlupic and Vreede, 2005). Simulation of business processes offers a great potential to overcome the inability to predict the effects of BPR on an organisation.

However, predictions with the use of simulation can be only as good as the underlying model of the business process. The saying 'garbage in, garbage out' holds especially true for the use of simulation in predicting the impact of changes in a business process. In other words, if the underlying model of the process is not at the level of how people actually do the work, the prediction fails to show how a change to the process will actually change people's work in the future. Therefore, it is important to model a business process at such a level that there is a direct correspondence with how this process is implemented.

Brahms is a modelling and simulation environment for analysing human work practice and for developing intelligent software agents to support work practice in organisations. Brahms is the result of more than 50 person years of research since 1992.[1] Brahms has been used on more than ten modelling and simulation research projects, including simulated human-robotic science exploration on planetary surfaces (Clancey et al., 2003, 2004, 2005a, Clancey, 2004). Brahms can be run in different simulation and runtime modes on distributed platforms, enabling flexible integration of people, hardware-software systems, and other simulations. Other publications about Brahms have described the theoretical foundations of work practice modelling (Clancey et al., 1998; Sierhuis, 2001; Sierhuis and Clancey, 2002) and particular applications (Acquisti et al., 2002; Seah et al., 2005; Sierhuis, 2000; Sierhuis et al., 2003a). Here we present a more technical overview of the modelling language and simulation engine.

Brahms was originally conceived as a business process modelling 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. Brahms models are at the implementation level of a work process and are especially suitable for modelling and simulating a designed change in a business process. Using Brahms we can predict the impact of a process design change on its implementation in an organisation. Predicting the impact is done by first modelling and simulating the current implementation of the work process, and subsequently simulating a future implementation model that changes the current implementation based on the new design. Our research started in the early nineties as a reaction to experiences with work process modelling and simulation (Sachs, 1995). Although an effective tool for convincing management of the potential cost-savings of the newly designed work processes, the modelling and simulation environment (Sparks™ from Coopers & Lybrand) was only able to describe work as an abstract, 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* (Button and Harper, 1996; Clancey, 2006). Multi-tasking, informal assistance and circumstantial work interactions could not easily be represented within a strict workflow modelling framework. In response, we began to develop a tool that would have the benefits of work process modelling and simulation, but be distinctively able to represent the relations of people, locations, systems, artifacts, communication and information content (Clancey et al., 1998). Thus, Brahms models work processes at the work practice level.

Agent architectures often do not link to theories of human behaviour, or empirical data on human behaviour in comparable situations. The Brahms environment is based on a number of behavioural and cognitive theories, most importantly situated cognition (Sachs, 1995; Clancey, 1997), activity theory (Leont'ev, 1978; Vygotsky, 1978), situated action (Suchman, 1987; Lave, 1988) and cognitive modelling (Laird et al., 1987; Anderson and Lebiere, 1998).

Brahms has been validated as a modelling and simulation tool for work practice design and analysis by applying and analysing Brahms in three different uses of modelling and simulation of work practice

- analysis of an existing work practice
- prediction of people's practice behaviour in an existing work system
- designing a new, not yet existing, work system.

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 (Sierhuis, 2001), as well as ethnographic data and video from Mars analog field missions (Clancey et al., 2005b).

To simulate human behaviour at the work practice level, one must model how people work together as individuals in organisations, performing both individual and teamwork activities. The Brahms language is unique in that it not only models both individual agent and group behaviour, but also systems and artifact behaviour, interpersonal interaction, as well as interaction of people, systems and objects with the environment. Most other multi-agent languages leave out artifacts and the interaction with the environment, making it difficult to develop a holistic model of real-world situations (c.f. Wooldridge and Jennings, 1995). By incorporating an ontology of objects, agents, groups, geography, etc. and means to model interactions, Brahms makes it possible to model empirical data gathered using ethnographic observations; this facilitates involving the workers being modelled in the simulation and work design process.

In this paper, we first review the meaning of work practice and our theory of modelling 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 an organisational process simulation tool.

2 Modelling work practice: a theoretical view

Work practice is embodied in the way people perform their daily work activities in organisations. Our notion of work practice modelling has been developed as a reaction to historically conventional views of workflow modelling in organisations. 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 (Emery and Trist, 1960; Pava, 1983; Weisbord, 1987; Ehn, 1988; Greenbaum and Kyng, 1991; Sachs, 1995; Clancey, 1999, 2001; Sierhuis and Clancey, 2002).

Our definition of work practice is narrower than Hofstede's dimensions of national culture (power distance, individualism, masculinity, uncertainty avoidance, long-term vs. short-tem orientation) and organisational culture or practices (process-oriented vs. results-oriented, job-oriented vs. employee-oriented, professional vs. parochial, open systems vs. closed systems, tightly vs. loosely controlled, and pragmatic vs. normative). We only intend to model the practice of people in an organisation in terms of their activities, which narrows our scope[2] (c.f. with Hofstede and Hofstede, 2005).

We define work practice as the collective social activities of a group of people who collaborate, cooperate, coordinate, and communicate, while performing their activities synchronously or asynchronously. Very often, people view work only as the process of transforming input to output, which is a functional, Tayloristic view. A functional or task-oriented perspective is often useful for work design, but such abstracted and idealised process models do not capture how work actually gets done 'in practice'.

Work practice is how people behave in everyday, located, circumstantial interactions in the real world. That is, a practice model describes behaviours (as activities); a task model describes functional relations of processes (Clancey, 2002). Put another way, a practice model emphasises interactions with the environment such as communication and movement; a task model emphasises mental operations (inference). A practice model indicates how information flows (e.g., by mobile phone or e-mail) and how that choice is made circumstantially; a task model indicates what information flows (and usually only what tools are supposed to be used). A task model indicates what methods are applied to transform work objects; a work practice model emphasises who selected those methods and how that person became involved in the work process.

To model people's behaviour we need to include ecological (environmental) influences on individual activity, especially layout of facilities, tools, and perhaps body posture. For example, it may be relevant to simulate whether two people can hear each other speak. The circumstantial details of work practice may include: physically joint actions (e.g., carrying something together), 'off-task' behaviours (e.g., joking), multi-tasking, interrupted and resumed activities (e.g., answering the phone while eating), informal or improvised interactions (e.g., unscheduled planning conversations), work-arounds (Clancey et al., 1998; Sierhuis, 2001).

A useful heuristic in simulating work practice is to model communications and how they occur. Brahms has been designed to facilitate modelling and visualising communications over time between people and systems. We define communication as:

> "The activity (speech act) of directional transferring of information (in the form of beliefs), held by one individual called the sender, to one or more individuals called the receiver(s), using a specific communication tool (face-to-face, telephone, e-mail, fax, document, etc). After the transfer activity is complete, and successful, the receiver(s) will hold the same information (belief) as the sender of the information, and can now react to it." (Sierhuis, 2001)

Our theory about modelling work practice is based on a number of elements borrowed from different existing approaches. Brahms models are models about real world phenomena, and the model is a description of the world as viewed by the modeller. Models of work practice are descriptions of work practice and as abstractions do not replicate all or even most of the aspects of human knowledge, experience, and behaviour. For example, situated cognition suggests that learning is always occurring and behaviour is always adapted, but Brahms agents behave rotely without any learning. Perception is modelled, but not facial expressions, gaze, or emotion.

Winograd and Flores (1986) 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. The context in which people perform real world activities is an important aspect. 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. Three approaches in the study of context – activity theory (Vygotsky, 1978; Leont'ev, 1978), situated action models (Clancey, 2002; Lave et al., 1984; Lave and Wenger, 1991; Nardi, 1996; Suchman, 1987), and distributed cognition (Hutchins, 1995a, 1995b) – have been fundamental in the development of our theory for modelling work practice. All these approaches use the notion of activity as the central concept for analysing the context of human behaviour.

2.1 Work activities

The key construct in the Brahms language is an activity, which as mentioned above, is distinguished from the traditional notion of a task – a representational construct that describes human behaviour in terms of problem-solving with goals and functional operators (e.g., Newell, 1990; Anderson and Lebiere, 1998) (cf. Clancey, 1992). While tasks are goal-driven functional behaviour representations of planned action, an activity is a construct that can also represent unplanned and not goal-driven behaviour; especially those behaviours that are based on culturally learned daily practice.

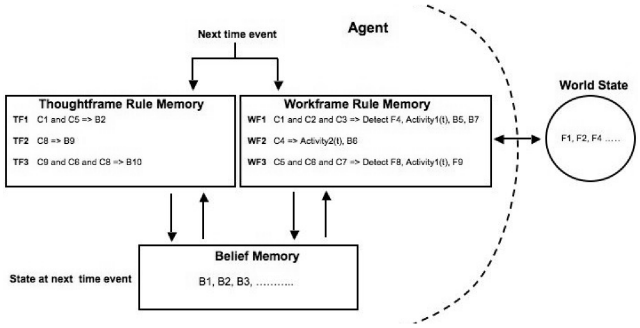
In Brahms a business process is described as the individual activities of people in the organisation. We describe each person's behaviour as situated in the organisation's physical and conceptual context, including location, tools, personal knowledge, interpersonal relationships, and aspects of organisational cultural behaviour. The question not only becomes what activities each individual performs, but more specifically, how they come to perform these activities at a particular time and place with other people. As we describe work practice in terms of activities engaged in, we also inquire how activities subsume and thus constrain each other. 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, and how that call is handled privately will blend the commitments and norms of both the parenting and business activities. In that sense, we are constantly managing our activities in context, which is to say that our behaviour is situated. Viewing a work process as the interaction of activities of people over time leads us to conceive workflow as the interactive, circumstantially adapted practice of people, instead of just a flow of information by a well-defined procedure through a hierarchical organisation.

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

3 The Brahms language

In this section we explain the modelling concepts of the language. For a more detailed description of the language see Sierhuis (2001) and van Hoof and Sierhuis (2000). Agents in Brahms are Belief-Desire-Intention (BDI) agents, with beliefs representing the agent's understanding of the world and rules representing their intentions to reason and perform activities (Bratman et al., 1988; Cohen and Levesque, 1990a; Rao and Georgeff, 1991). Figure 1 is a conceptual diagram of a Brahms agent showing Situated-Action (SA) Rules in a production system, used to represent people's situated activities in the world.

Figure 1 Brahms situated-action rule system



SA Rules are a combination of a situated action and a cognitive framework. P1–P3 are SA Rules. C1–C7 are preconditions on the rules. They are matched against elements in working memory (E1, etc.). When elements match to all the conditions in the rules, the right hand side of the rule is executed. In contrast with a conventional production rule, the action of an SA rule has a duration and may specify another activity, which is itself modelled as SA Rules. Thus, an SA rule system adds timing to the execution of production rules. In Figure 1, B1–B5 are new beliefs that are created in working memory when SA rules execute.

A Brahms agent has a Thoughtframe Rule Memory (TRM), a Workframe Rule Memory (WRM), and a Belief Memory (BM). The BM contains a set of all beliefs of the agent. Beliefs are first-order logic propositions (see section for a description of beliefs).

The WRM contains a set of rules called workframes. A workframe is a special type of production rule, namely one that can create not only new beliefs, but also execute activities and create facts in the World State. Each workframe consists of a left-hand side and a Right-Hand Side (RHS). The left-hand side is a set of preconditions that are matched against the beliefs in the BM. RHS contains a list of statements, including detectables (see 'Detect F4' in Figure 1), activities (see 'Activity 1(t)' in Figure 1) and belief or fact conclude statements (see 'B5' and 'F9' in Figure 1). A detectable is a declarative statement that during the execution of the workframe detects a fact from the world state that matches its detect condition. For example, in Figure 1 'Detect F4' represents that the agent is able to detect a fact of the form F4. When an agent detects fact F4 in rule P1, it creates a belief from the fact (B4) in the agent's belief memory. This belief is now used in the next event cycle to match against the preconditions of all the rules.

The TRM contains a set of rules called thoughtframes. A thoughtframe is a 'pure' production rule (also 'inference rule'), in that it can only create new beliefs. Unlike a workframe, a thoughtframe's application does take time; the change in beliefs occurs immediately. The agent's rule engine (called the inference engine) is event-based. Events are scheduled by the Scheduler, which in turn is based on a

simulation clock (neither the Scheduler, nor the simulation clock are shown in Figure 1).

The work practice of a person is represented as a combination of beliefs, thoughtframes, workframes, and activities that can be performed by an agent representing the person. The workframes constrain when the agent can perform the activities. Beliefs the agent acquires by executing workframes and thoughtframes represent the person's interpretation of the world. Detection of facts represents perception in the world.

The Brahms language was primarily designed to develop simulations of human and machine behaviour.[3] To enable the modelling of human activity behaviour, the Brahms language embodies assumptions about how to describe social action,[4] workplaces and work practice. Brahms is an agent language that operationalises a theory for modelling work practice, allowing a researcher to develop models of human activity behaviour that corresponds with how people actually behave in the real world.

3.1 Declarative vs. imperative

In this section we discuss Brahms as a declarative language. For completeness, we start with a definition of imperative (a.k.a procedural) vs. declarative programming:

> "In computer science, imperative programming, as opposed to declarative programming, is a programming paradigm that describes computation in terms of a program state and statements that change the program state. […]. Logical programming languages, like Prolog, are often thought of as defining 'what' is to be computed, rather than 'how' the computation is to take place, as an imperative programming language does."[5]

Brahms is a declarative language. Brahms agents act based on beliefs matching to preconditions on SA rules. Workframes (SA Rules) are declarative statements about when new beliefs can be inferred and when activities can be performed. These are declarative statements, because workframes do not say when and how an agent's behaviours (reasoning and situated-actions) will be executed. The order of execution is dependent on the implementation of the inference engine. In contrast, in an imperative language, such as C, C++ or Java, the behaviour of a piece of code (method, object, function, etc.) is not declarative, but specified in terms of when and how the code will be executed. Any code-fragment (statement) specifies how it is to be executed, i.e. what memory to use, the order of execution, and the next statement to be executed.

A workframe has variables, which look like imperative constructs, however a Brahms variable is not an assignment of data to a memory location. Instead, it represents a binding (instantiation) of a concept (i.e., a value, either an object or an agent), based on the matching of a condition to the agent's belief memory. Secondly, the set of statements in the body of a workframe is declarative in the sense that it only states what the ordering relation is between two or more activities. It does not state how these activities will be executed, because each activity is decomposed into another set of declarative workframes. A workframe can be interrupted, impassed, stopped, etc, based on beliefs that the agent can acquire through communication, detections, and other workframes and thoughtframes.

3.2 Comparison with multi-agent programming languages

In this section we compare Brahms briefly with other multi-agent programming languages and with agent-based cognitive architectures. For an introduction to multi-agent systems we refer the reader to Wooldridge (2002).

As mention above, Brahms is a type BDI system conceived of and developed in 1992, before many of today's multi-agent languages. Unlike today's BDI agent languages, Brahms is inspired most strongly by Brooks' (1986, 1999) biologically inspired reactive robot architectures and Cohen et al.'s (1989) simulation of located team communications. For a more complete description of different multi-agent languages see Bordini et al. (2005).

We categorise multi-agent languages as follows (Table 1): Java-based Agent Languages (BDI-based and Imperative) and BDI Languages (Goal-based and Subsumption-based), and Agent Simulation Languages (BDI-based and Imperative).

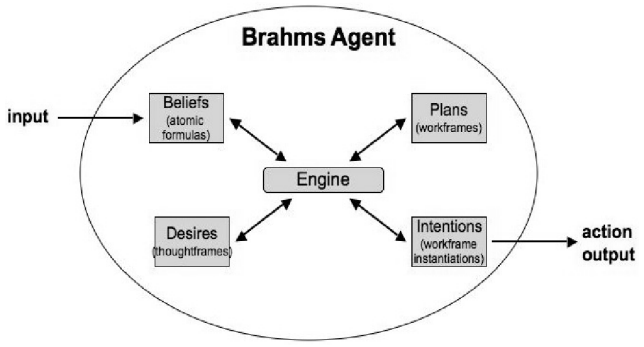
Table 1 Agent oriented languages comparison

	Java-based agent languages		*BDI languages*		*Agent simulation languages*	
	BDI-based	*Imperative*	*Goal-based*	*Subsumption-based*	*BDI-based*	*Imperative*
Brahms				X	X	
Jason			X			
Agents speak			X			
Jade		X				
Jack	X					
Jadex	X					
Swarm						X
Repast						X

Brahms uniquely combines a subsumption-based architecture (Brooks, 1986) with a BDI-based agent-oriented language (see Section 4.5.3). In contrast, Swarm (Minar et al., 1996), an often-used language for modelling and simulating social and economic behaviour of large agent societies (Luna and Perrone, 2002), is not based on any particular human behaviour theory and is not an agent-oriented language in the strict sense. Rather, Swarm extends an imperative language in the form of object libraries for Objective-C (an object-oriented programming language) (Terna, 1998). Swarm agents are not BDI agents, but objects with inherited imperative methods that are called by a higher-level schedule object in the model. In contrast, in Brahms agent actions are not scheduled by an overall scheduler, but by each agent's individual inference engine

that schedules and executes the agent's activities based on the agent's plans, beliefs, desires and intentions (see Figure 2).

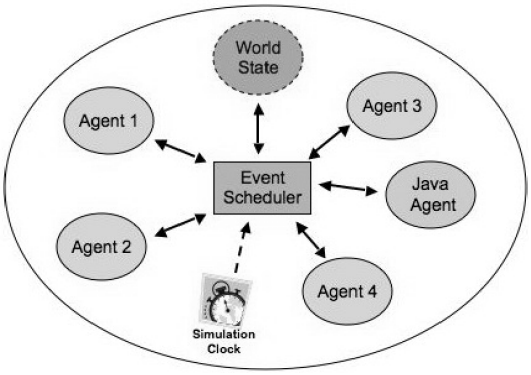
Figure 2 Brahms BDI agent



Source: Adapted from Figure 4.5 in Woodridge (2002)

However, Brahms is a multi-agent language allowing agents to communicate beliefs to one another. Agents can also change and detect facts of the world state. Both the communication of beliefs and creation and detection of facts are scheduled for all agents based on the simulation clock (see Figure 3).

Figure 3 Brahms multi-agent virtual machine



Recent research in the agent community recognises the environment as a first-order abstraction in Multi-Agent Systems (MAS). Researchers in this newly created subfield of MAS (see E4MAS workshops)[6] recognise that several aspects of MAS are part of the environment and should be included as a separate first-order entity (Omicini et al., 2004). In modelling work practice we have recognised the importance of modelling the environment from the onset. In Brahms the environment is modelled with specialised object types (see sections The Object Model and The Geography Model).

4 An example model: simulating a robotic mission to the Moon

Sending robots to the Moon (or Mars) is a difficult task, one that mission designers at the Jet Propulsion Laboratory (JPL) in Pasadena, CA devoted years to design and implement. Understanding how scientists and engineers work together during a mission, and also how people communicate with robots on a planetary surface is a research topic for designing better mission operations (Sierhuis et al., 2003a, 2003b). We use our work in modelling mission operations for planetary robotic missions as an example to explain the Brahms language and its representation capabilities. In particular, the Victoria model is a Brahms model of the mission operations for 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 (Sierhuis, 2001; Sierhuis et al., 2003a).

4.1 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 modelling 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. Our convention is to create one main model file that imports all other model files. Since Brahms does not have an initialisation function, such as the 'main' function in the C program language (Kernighan and Ritchie, 1988), 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 declares that the model exists as a package called Victoria. A package is a directory file structure allowing the modeller to compartmentalise 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 model file 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 an XML data definition language, making each compiled Brahms model file a Brahms XML file that can be loaded and executed by the BVM.

In developing a Brahms model, we conceptually divide the system to be modelled 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 modeller to create language constructs within these sub-models using a graphical user interface.

4.2 Agent model

When developing a Brahms model we first design an Agent Model. The Agent Model systematically relates groups, agent beliefs, and facts about agents.

4.2.1 Group hierarchy

The agent model consists of a group hierarchy representing the social, organisational 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. The science team members are all top scientists in their field, and specialise in different scientific disciplines. For example, some science team members specialise in the 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 omitted):

Excerpt 2 Partial agent model

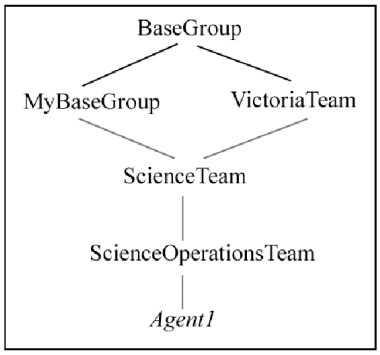
```
group MyBasegroup memberof BaseGroup {
   attributes:
      public symbol groupMembership;
}

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 code file, but is shown here as if it were part of one file.

The first two groups are *MyBaseGroup* and *VictoriaTeam*. *MyBaseGroup* is a group defined by the modeller. It is a non-domain specific 'root' of the group hierarchy, used by the modeller 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 recognise naming conflicts in multiple inheritance and will report these at compile time. 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 4.

Figure 4 Group hierarchy from Excerpt 2



4.2.2 Agent beliefs

Intentional agents are entities whose behaviour can be predicted by the method of attributing belief, desire and acumen (Dennett, 1987; Wooldridge, 2002). This philosophical stance has resulted in representing intentionality as a logical framework in which agents have

beliefs and a deduction model indicating how beliefs change (Bratman, 1999; Cohen and Levesque, 1990b; Konolige, 1986). 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 behaviour 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 modeller can specify initial beliefs for an agent. Initial beliefs are beliefs that the agent receives at initialisation. 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 4.5).

4.2.3 World facts

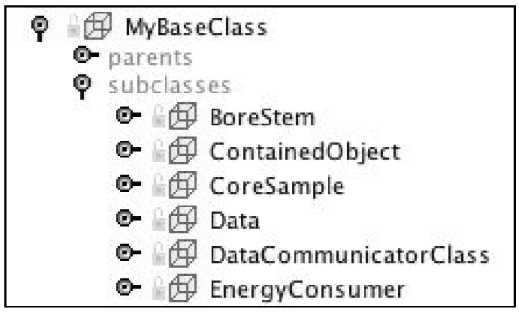
If agents can have different beliefs about attributes of agents or objects, how can we represent the actual state of the environment in which agents are located? Brahms operationalises the second world-view from Winograd and Flores (1986) by representing 'objective facts' about the world as facts in 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 4.2.3). Excerpt 2 shows that at initialisation *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. Whereas 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 (from the modeller's perspective) about the state of the simulated world.

4.3 Object model

Similar to the agent model, the object model defines the objects in the world. There are two types of objects, data and physical artifacts with or without behaviour – plainly called objects – and concepts represented as conceptual objects with attributes. Using these two distinct object types we can both represent the behaviour 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 can not. The notion that objects can have beliefs might, at first thought, seem problematic. However, beliefs are nothing more than first-order propositions. In objects such propositions represent information held by the object; that is, like the beliefs of agents they are representations of the state of the world.

Objects and conceptual objects can be part of a class inheritance hierarchy, similar to other object-oriented programming languages (see Figure 5), a screenshot of the *MyBaseClass* from the Brahms development environment.

Figure 5 Class hierarchy



4.4 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

thoughtframes. Each agent and object can have a set of thoughtframes locally declared and/or inherited from a group or class. For example, 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 {
  …
 thoughtframes:
  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. 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 ($O.A = v$), where O = '*current*', A = [*an attribute of the Rover group*] and v is the outcome of a numerical expression that is evaluated before the belief is created.

Expression v is evaluated as follows; v ::= *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 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.[7] 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 behaviour 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 behaviour 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 behaviour efficient, because at every belief change event in an agent only a small number of preconditions have to be evaluated (Forgy, 1982).

4.5 Activity model

The activity model consists of the possible activity behaviour for an agent. This is the heart of a Brahms model, because modelling work practice is about the representation of people's activity behaviour 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 activity and workframe from Excerpt 4 is from the *Science OperationsTeam* group. The source code specifies a group member's behaviour 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 behaviour. 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.

Excerpt 4 Partial activity model for the ScienceOperationTeam group

```
composite activity FindingWaterIce(Crater
                              crater, int pri)
{
  priority: pri;
  activities:
    primitive activity WaitingForData( ) {
      priority: 0;
      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
```

4.5.1 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. A difference between a workframe and a thoughtframe is that a thoughtframe does not have duration, while a workframe has duration based on the duration of the activities within it (see explanation of activities below. Workframes 'fire' according to the same pattern-matching process explained for thoughtframes (see section 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 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). Recall that the reasoning of the agent is represented by thoughtframes.

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 workframes, 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. 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 emphasised that the process presented so far – that is, matching of beliefs to preconditions, 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 do not take any simulation time for the agent.

4.5.2 Activities

Activities are the most important construct in the Brahms language. All agent behaviour has to be modelled as an activity. There are three different types: primitive,

composite and Java activities. All activities have a user-defined name representing a behaviour defined by the modeller. According to our theory of activities (Clancey, 2002), the name of an activity should be the name of an observed behaviour 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 modeller to decide the relevance of the model to the system behaviour that is being modelled. This allows the use of the Brahms language in any domain and for any purpose, including, but not restricted to, modelling social phenomena, human behaviour, and software agent behaviour.

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.

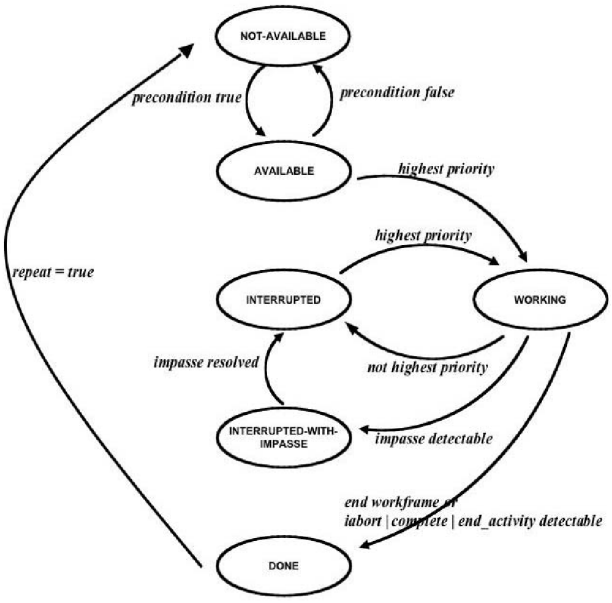
Activities can have parameters that are passed as bounded variables into the activity when it is called in a workframe. In WFR *wf_SearchForWaterIce* the parameter-values that are passed are the value of the *rover-loc* variable, bound in the precondition as the *crater* in which the rover is searching 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*. In general, thoughtframes in a composite activity can be used to model reasoning within the context of a specific activity. 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 again 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[8] (WFI) in which the activity is being called. Each WFI is in one of the states shown in Figure 6. The state of an agent's activity behaviour is defined by the combined sets of available, working, interrupted, and interrupted-with-impasse WFIs at any moment in time.

Figure 6 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 *Receive HydrogenData* 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 behaviour 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 Communication Model), and detection of facts in the world. The behaviour 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 modelled environment. We refer to the Brahms modelling paradigm as a *situated activity paradigm*.

4.5.3 Activity subsumption architecture

A good definition of a subsumption architecture is given in Travers (1996), also see Brooks (1986):

> "A subsumption program consists of a number of modules connected in a network, usually arranged in a layered fashion, with each layer capable of controlling action independently. Higher-level layers are capable of preempting lower-level ones using a scheme based on fixed priority gates that make up the network. Each module is an autonomous augmented finite-state machine, which can communicate with the outside world through sensors, to other modules through the network, and to itself through a small set of registers. Modules typically implement fairly simple behavioural control rules, sometimes augmented with state. Goals are implicit rather than explicit, and conflict between modules is handled by hardwired priorities in the connections between the modules and the network."

Each Brahms agent's engine independently operates according to a subsumption algorithm. An agent's activities are like 'modules connected in a network'. Activities are decomposed in lower-level activities. The higher-level activities preempt the lower-level ones using a dynamic priority-based scheme that selects the one activity being executed at any one time. Each activity can be seen as an independent finite-state machine with workframes and thoughtframes making up the network. Goals are implicit and the conflicts between activities are handled by dynamic priorities that are passed as parameters. Activities can change state (i.e., beliefs) of an agent, creating simple behavioural situated-action rules.

An important aspect of the Brahms activity paradigm is that activities are *not* the same as subroutines and co-routines in imperative languages (Pratt and Zelkowitz, 1996). Imperative languages use a computer memory-based program stack to keep track of subroutine calls. When a subroutine is executed, its context is 'popped' onto the program stack. When in a subroutine the program is not also still in the context of the parent routine. The program can not move execution back and forth between a subroutine and its subsequent subroutines. Subroutine execution is sequential and can not be interrupted until it exits. Some high-level imperative languages have solved the problem of subroutine exit and reentrance with co-routines. Co-routines generalise subroutines to allow multiple entry points, enabling suspending and resuming of execution at certain locations. However, co-routines are still imperative constructs with a pre-defined deterministic order of execution. Although co-routines can be interrupted and resumed, this can only happen at pre-defined points, so-called yield statements in the co-routine. The parent or any other routine can not control co-routine interruption. It is the yield statement within the co-routine itself that relinquishes program control to its calling routine.

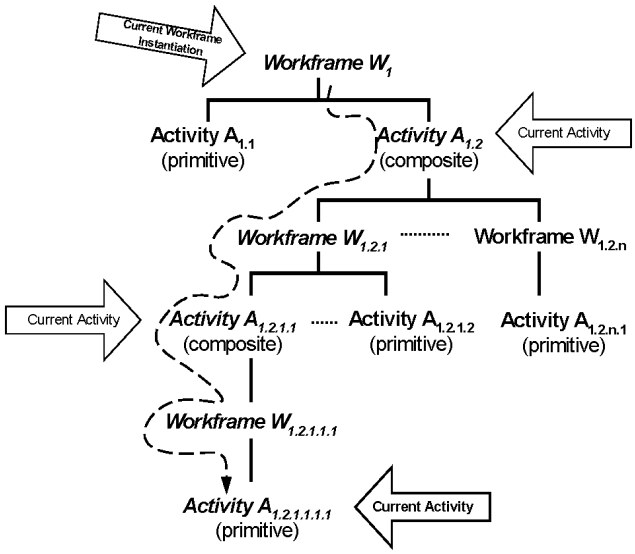
In contrast, in Brahms all workframes can interrupt the current workframe and activity at any point during execution. There is no yield statement necessary inside the current workframe. It only depends on the whether the preconditions are satisfied and the priority of the interrupting workframe is higher that the current one, if a current activity is to be interrupted by another activity. In short, the interruption is done from outside the current workframe and activity, whereas in a co-routine it is done from within the co-routine. Importantly, when a co-routine is interrupted its context is not active and no statement inside it can be evaluated. Interruption of a co-routine thus means it is made inactive. In contrast, Brahms impassed workframes and activities are still executing detectables, while they are impassed.

If an activity in a workframe of a composite activity is executed, the context of the parent composite activity is also still active (see Figure 7). All workframes, thoughtframes and detectables in the parent activity are still being evaluated while the agent is executing the subactivity. This is the essence of the Brahms *subsumption* architecture (Brooks, 1991), and is based on the principle that humans are always multi-tasking by being in multiple subsumed activity hierarchies 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 behavioural state of the agent.

In Brahms, an agent may engage in multiple activities at any given time, but only one activity in one workframe is *working* 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, interrupted and impassed work (see Figure 6). 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. There is no need for a yield statement relinquishing control to the interrupting activity. Unlike co-routines, it is not the current activity deciding to interrupt its execution, instead it is the agent's reasoning engine determining which active workframe and activity has the highest priority. It is the highest priority activity that interrupts the current activity.

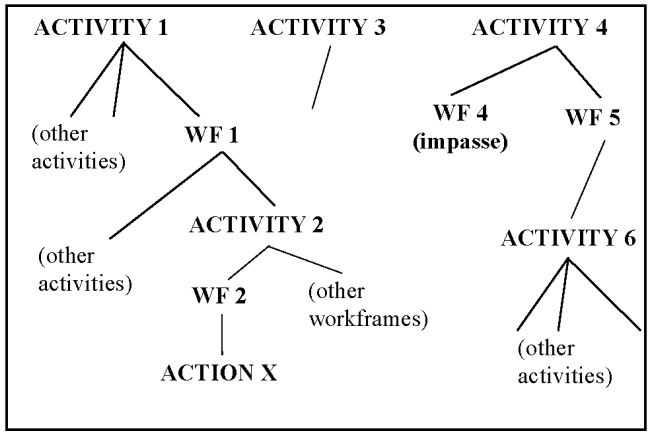
Figure 7 Workframe-activity hierarchy



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* a stack for workframe and activity execution. A stack would only allow those workframes at the level of the current active workframe on the stack to be considered for execution. Instead all workframes at all levels are always activated.

An illustration of multi-tasking is given in Figure 8. An agent (not shown) in a running model may have multiple competing general activities in process: Activities 1, 3, and 4. Activity 1 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.

Figure 8 Multi-tasking in Brahms



The Brahms subsumption architecture allows two forms of *multi-tasking*. The first form is inherent in the activity paradigm; Brahms can simulate reactive situated behaviour 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 behaviour as represented in Figure 8. The second form is subtler. People can be working concurrently on many high- and medium-level activities in a workframe-activity hierarchy. Although an agent can only execute one primitive activity in the hierarchy at a time (e.g., ACTION X in Figure 8), the agent is concurrently within all the higher-level activities in the workframe-activity hierarchy. For example, the agent in Figure 8 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 executing all current, interrupted and impassed activities.

4.5.4 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 3, 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', etc. 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

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

4.6 Communication model

One of the most important aspects of modelling human behaviour 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 send beliefs in its belief-set, however an agent/object can receive missing beliefs by asking.

Modelling work practice of people means that we are interested in modelling how communication actually happens in the real world. Therefore, in Brahms we usually represent the media and path of the communication. For example, when we model an organisation of communicating people we represent the communication tools that are used (e.g., e-mail, telephones or faxes). We have even modelled the operation of the telephone system with voice mail capability. Accordingly, we can represent how communication occurs and how long it requires. For example, this includes the practice by which a phone number is known, such as looking it up in an computerised address book. 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 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. This level of detail is not required to model communication, but it may be important to explain how work gets done (or is delayed).

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(VictoriaRover.drivingTime =
         anyvalue),
    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=
SearchForWaterIceInPermanentDarkAreaActivi
ty));
    conclude((VictoriaRover.subActivity =
              DrillingActivity));
    conclude((SATM.lengthIntoSurface =
              10.0)); //drill 10cm deep
    conclude((SATM.sampleVolume = 1.0));
   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 constructs of *workframe*, *preconditions*, *conclude* commands and *activity calls*. Here 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 *VehicleAndSpacecraft OpsTeam* (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.

Another communication activity property is *about*. This property specifies the possible content of the communication. 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 can not take place.

The last property for communication activities is *when*. 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 do not 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. Note that to model the actual transfer of beliefs using some kind of timed distribution of actual transfer of beliefs during the activity, we would represent a number of sequential communication activities, such as parts of a conversation.

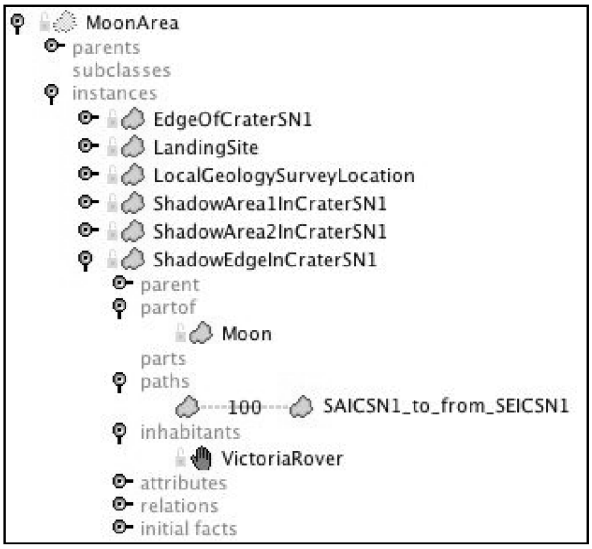
4.7 Geography model

The Geography Model consists of declaration of areas (a.k.a. locations) where agents and objects can be placed. Areas can represent a conceptual hierarchical organisation of locations. Figure 9 presents the partial geography model of the Moon.

Areas are instances of classes called *AreaDefinitions* (areadefs). Figure 9 shows the *MoonArea* areadef. As with any other type of object or class, areas and areadefs can have attributes and be hierarchically organised. 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).

Related, in an associated project, called BrahmsVE, we map the Brahms geography model onto a three-dimensional virtual reality model, which can be used to model and visualise the physics of the world, including line of sight and obstacles during movements (Clancey et al., 2005b).

Figure 9 Partial geography model of the moon



4.7.1 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 9, the *ShadowEdgeInCraterSN1* has the following relationships; First off, this area is part of the *Moon* area (i.e., the crater area is located on the Moon). The *partof* relation is important for the localisation 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 9 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.

Location of agents and objects 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 updated by the simulation engine. First, localisation is a fact in the world and the engine automatically generates a *location* fact for each inhabitant. Thus, for the model in Figure 9 the engine generates the following fact: (*VictoriaRover.location = ShadowEdge InCraterSN1*). Second, every agent that is an inhabitant of an area gets a belief about its location, as well as that of all co-inhabitants in that area. Thus an agent always knows the location of all other agents and objects in its location – i.e., agents have perfect perception. It is possible for an agent to simulate imperfect perception by changing its gotten location beliefs in three ways:

- a location conclude statement in a workframe
- a location conclude statement in a thoughtframe
- through a communication about the location with another agent/object (send or receive).

The agent's inference engine dynamically updates the agent's location-fact and -belief when the agent moves from one area to another. The agents still located in the old area have the location-belief of the moved agent retracted so that they know the agent has moved. Next, all agents that are inhabitants of the newly move-to location receive a location-belief for the arrived agent. In short, agents always know where they are and also always know which other objects and agents are in their location.

4.7.2 Movement

As mentioned above, agents and objects can move between areas; this occurs in a *move* activity, which like all activities takes time. 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. Excerpt 7 shows an example of a Rover agent's ability to execute the *TraverseToLocation* activity in a workframe. Moving is constrained, 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 two 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 modelled as an implicit calculation based on time and distance).

Another way of modelling 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 9 shows that the geography model specifies that there exists a path *SAICSN1_to_from_SEICSN1* between the areas *SunlitAreaInCraterSN1* and *ShadowEdge InCraterSN1*, 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 9

```
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 occurs 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 determined from the duration of the path. In case of multiple possible paths the engine calculates the shortest route[9] and uses that as the duration of the move activity.

5 Discussion

After the detailed description of the Brahms language, its human behaviour modelling capabilities and the workings of the simulation engine, we now turn to the use of Brahms as a modelling and simulation tool for the organisational 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 organisations. Our research over the past decade has shown that, discounting the difficulties of modelling human behaviour with all the representational limitations, Brahms allows the detailed modelling of work practice at a level of detail that enables

- researchers to get insight into the way people actually work in an organisation
- 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 have argued that the Brahms language is suitable for studying kinds of social and work practice phenomena of interest to the organisational process simulation community. Our experience and results in modelling work practice suggests that larger social phenomena can also be modelled. The Brahms modelling language has great advantages for the researcher, because compared to other tools, such as Swarm, the language allows for a more 'natural' representation of human behaviour 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 (Carley and Prietula, 1994)). Our experience indicates that when the objective is to analyse or predict organisational behaviour at the macro system level, an activity-based simulation of individual behaviour captures a level of detail about information flow and tool use that is useful for explaining the quality and timing of work processes.

Activity-scanning is a known simulation method for modelling work processes. Activity-scanning models represent how activities are performed, based on the resources needed and the conditions under which they are performed (Ioannou and Martinez, 1999). Although at first glance activity-scanning looks similar to our definition of activities, an activity-scanning model only includes tasks and queues of resources as input and output to activities. The model is a functional activity-resource network and operates similarly to a Petri-Net model (Aalst, 2003) in that resources flow through the model as constraints on when activities can be executed. People are equal to other resources and seen as resources consumed or created by an activity. This model is

- not an agent-based model
- does not deal with agent beliefs (BDI)
- does not include agent communication and interaction
- does not allow agent reasoning
- also does not have any model of the environmental/geography that influences when and how activities are executed.

We argued that imperative programming languages suited for modelling macro-level system behaviour using an agent design paradigm are not flexible enough to claim a correspondence with actual human behaviour. Cognitive architectures that are suited to model single agent cognitive behaviour, based on a theory about how the brain actually stores and processes information, are too detailed to conveniently model human behaviour 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 100 msec range, which makes modelling activities that span days, hours, or even minutes very cumbersome). Brahms lies between these types of modelling languages. Brahms is a language that allows for an easy representation of agent behaviour at the micro-level (i.e., reasoning behaviour, 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 behaviours at the macro-level (i.e., the organisational process or system level).

Brahms is a declarative language like all BDI languages. Brahms differs from other BDI languages in several ways:

- Brahms is activity-based, while most other BDI languages are task-based
- Brahms uses a subsumption architecture, while most other BDI languages use a goal-based architecture
- Brahms allows modelling of the environment (geography), movement of agents in the environment, etc.
- Brahms is also an object language allowing the representation of artefacts and data objects
- Brahms represents a separate fact-state for modelling the world state outside of the agent's belief-set, whereas traditionally BDI-languages only model agents with an independent belief-state.

The Brahms environment is completely developed in the Java language and, with its language compiler, BVM, development environment (the Composer), and its simulation display environment (the AgentViewer). Brahms is freely available from the internet for research purposes.[10]

Acknowledgement

We are grateful for the funding provided over the years by NYNEX Science & Technology and NASA.

References

Aalst, W.v.d. (2003) 'Putting Petri Nets to work in the workflow arena', in van der Aalst, J.M.C.W., Kordon, F., Kotsis, G. and Moldt, D. (Eds.): *Petri Net Approaches for Modeling and Validation*, Lincom Europa, Munich, pp.125–143.

Acquisti, A., Sierhuis, M., Clancey, W.J. and Bradshaw, J.M. (2002) 'Agent based modeling of collaboration and work practices onboard the international space station', Paper presented at the *11th Computer-Generated Forces and Behavior Representation Conference*, Orlando, FL.

Anderson, J.R. and Lebiere, C. (1998) *The Atomic Components of Thought*, Lawrence Erlbaum Associates, Mahwah, NJ.

Bordini, R.H., Dastani, M., Dix, J. and Seghrouchni, A.E.F. (Eds.) (2005) *Multi-Agent Programming: Languages, Platforms and Applications*, Springer Science + Business Media, Inc., New York, NY.

Bratman, M.E. (1999) *Faces of Intention: Selected Essays on Intention and Agency*, Cambridge University Press, Cambridge, UK.

Bratman, M.E., Israel, D.J. and Pollack, M.E. (1988) 'Plans and resource-bounded practical reasoning', *Computational Intelligence*, Vol. 4, pp.349–355.

Brooks, R.A. (1986) 'A robust layered control system for a mobile robot', *IEEE Journal of Robotics and Automation*, Vol. 2, No. 1, pp.14–23.

Brooks, R.A. (1991) 'Intelligence without representation', *Artificial Intelligence*, Vol. 47, pp.139–159.

Button, G. and Harper, R. (1996) 'The relevance of 'work-practice' for design', *Computer Supported Cooperative Work*, No. 4, pp.263–280.

Carley, K.M. and Prietula, M.J. (1994) 'ACTS theory: extending the model of bounded rationality', in Carley, K.M. and Prietula, M.J. (Eds.): *Computational Organization Theory*, Lawrence Erlbaum Associates, Hillsdale, NJ.

Clancey, W.J. (1992) 'Model construction operators', *Artificial Intelligence*, Vol. 53, No. 1, pp.1–124.

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

Clancey, W.J. (1999) 'Human exploration ethnography of the Haughton-Mars project 1998–99', Paper presented at the *Mars Society Annual Meeting*, Boulder, CO.

Clancey, W.J. (2001) 'Field science ethnography: methods for systematic observation on an expedition', *Field Methods*, Vol. 13, No. 3, pp.223–243.

Clancey, W.J. (2002) 'Simulating activities: relating motives, deliberation, and attentive coordination', *Cognitive Systems Research*, Vol. 3, No. 3, pp.471–499.

Clancey, W.J. (2004) 'Roles for agent assistants in field science: personal projects and collaboration', *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, Vol. 34, No. 2, May, pp.125–137.

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

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

Clancey, W.J., Sierhuis, M., Kaskiris, C. and Hoof, R.v. (2003) 'Advantages of Brahms for specifying and implementing a multiagent human-robotic exploration system', Paper presented at *The 16th International FLAIRS Conference 2003*, St. Augustine, FL, pp.7–11.

Clancey, W.J., Sierhuis, M., Alena, R., Crowford, S., Dowding, J., Graham, J., Kaskiris, C., Tyree, K.S. and Hoof, R.v. (2004) 'The mobile agents integrated field test: Mars Dessert Research Station 2003', Paper presented at the *FLAIRS 2004*, Miami Beach, Florida, pp.732–737.

Clancey, W.J., Sierhuis, M., Alena, R., Berrios, D., Dowding, J., Graham, J.S., Tyree, K.S., Hirsh, R.L., Garry, W.B., Semple, A., Buckingham Shum, S.J., Shadbolt, N. and Rupert, S. (2005a) 'Automating CapCom using mobile agents and robotic assistants', *American Institute of Aeronautics and Astronautics 1st Space Exploration Conference*, 31 January – 1 February, Orlando, FL, Available as AIAA Meeting Papers on Disc [CD-ROM]: Reston, VA, and as an Advanced Knowledge Technologies Project e-Print [http://eprints.aktors.org/375].

Clancey, W.J., Sierhuis, M., Damer, B. and Brodsky, B. (2005b) 'The cognitive modeling of 'day in the life' social behaviors using Brahms', in Sun, R. (Ed.): *Cognitive Modeling and Multi-agent Interaction*, Cambridge University Press, New York, pp.151–184.

Cohen, P.R. and Levesque, H.J. (1990a) 'Intention is choice with commitment', *Artificial Intelligence*, Vol. 42, pp.213–261.

Cohen, P.R. and Levesque, H.J. (1990b) 'Rational interaction as the basis for communication', in Cohen, P.R., Morgan, J. and Pollack, M.E. (Eds.): *Intentions in Communication*, The MIT Press, Cambridge, MA, pp.221–256.

Cohen, P.R., Greenberg, M.L., Hart, D.M. and Howe, A.E. (1989) 'Trial by fire: understanding the design requirements for agents in complex environments', *AI Magazine*, Vol. 10, No. 3, pp.34–48.

Dennett, D.C. (1987) *The Intentional Stance*, Bradford Book, MIT Press, Cambridge, MA.

Dijkstra, E.W. (1959) 'A note on two problems in connection with graphs', *Numerische Math*, Vol. 1, pp.269–271.

Ehn, P. (1988) *Work-Oriented Design of Computer Artifacts*, Arbetslivcentrum, Stockholm, Sweden.

Emery, F.E. and Trist, E.L. (1960) 'Socio-technical systems', in Churchman, C.W. (Ed.): *Management Sciences, Models and Techniques*, Pergamon, London.

Forgy, C.L. (1982) 'Rete: a fast algorithm for the many pattern/many object pattern match problem', *Artificial Intelligence*, Vol. 19, pp.17–37.

Greenbaum, J. and Kyng, M. (Eds.) (1991) *Design at Work: Cooperative Design of Computer Systems*, Lawrence Erlbaum, Hillsdale, NJ.

Hlupic, V. and Vreede, G.J.d. (2005) 'Business process modeling using discrete-event simulation: current opportunities and future challenges', *International Journal of Simulation and Process Modeling*, Vol. 1, Nos. 1–2, pp.72–81.

Hofstede, G. and Hofstede, G-J. (2005) *Cultures and Organizations: Software of the Mind*, 2nd ed., McGraw-Hill, New York.

Hutchins, E. (1995a) *Cognition in the Wild*, MIT Press, Cambridge, MA.

Hutchins, E. (1995b) 'How a Cockpit remembers its speeds', *Cognitive Science*, Vol. 19, pp.265–288.

Ioannou, P.G. and Martinez, J.C. (1999) 'Who serves whom? Dynamic resource matching in an activity-scanning simulation system', Paper presented at *The 1999 Winter Simulation Conference*, Squaw Peak, Phoenix, AZ.

Kernighan, B.W. and Ritchie, D.M. (1988) *The C Programming Language*, 2nd ed., Prentice-Hall, Engelwood Cliffs, NJ.

Konolige, K. (1986) *A Deduction Model of Belief*, Morgan Kaufmann, San Mateo, CA.

Laird, J.E., Newell, A. and Rosenbloom, P.S. (1987) 'Soar: an architecture for general intelligence', *Artificial Intelligence*, Vol. 33, pp.1–64.

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

Lave, J. and Wenger, E. (1991) *Situated Learning – Legitimate Peripheral Participation*, University Press, Cambridge University Press, Cambridge, UK.

Lave, J., Murtaugh, M. and Roche, O.d.l. (1984) 'The dialectic of arithmetic in grocery shopping', in Rogoff, B. and Lave, J. (Eds.): *Everyday Cognition: Its Development in Social Context*, Harvard University Press, Cambridge, MA.

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

Luna, F. and Perrone, A. (Eds.) (2002) *Agent-Based Methods in Economics and Finance: Simulations in Swarm*, Kluwer Academic Publishers, Norwell, MA.

Minar, M., Burkhart, R. and Langton, C. (1996) *Swarm Development Group*, [Website], Available: http://www.swarm.org [2003, 08/05/2003].

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

Newell, A. (1990) *Unified Theories of Cognition*, Harvard University Press, Cambridge, MA.

Omicini, A., Ricci, A., Viroli, M., Castelfranchi, C. and Tummolini, L. (2004) 'Coordination artifacts: environment-based coordination for intelligent agents, Paper presented at the *Third International Joint Conference on Autonomous Agents and Multiagent Systems*, New York, NY.

Pava, C.H.P. (1983) *Managing New Office Technology: an Organizational Strategy*, The Free Press, New York.

Pratt, T.N. and Zelkowitz, M.Z. (1996) *Programming Languages: Design and Implementation*, 3rd ed., Prentice-Hall, Englewood Cliffs, NJ.

Rao, A.S. and Georgeff, M.P. (1991) 'Modeling rational agents within a BDI-architecture', Paper presented at the *Proceedings of Knowledge Representation and Reasoning (KR&R-91)*, San Mateo, CA.

Sachs, P. (1995) 'Transforming work: collaboration, learning and design', *Communications of the ACM*, Vol. 38, No. 9, pp.36–44.

Seah, C., Sierhuis, M. and Clancey, W.J. (2005) 'Multi-agent modeling and simulation approach for design and analysis of MER mission operations', *SIMCHI: Human-Computer Interface Advances For Modeling And Simulation*, January, pp.73–78.

Sierhuis, M. (2000) 'Modeling and simulation of work practices on the moon', Paper presented at the *Computational Analysis of Social and Organizational Systems 2000*, Carnegie Mellon University, Pittsburgh, PA.

Sierhuis, M. (2001) *Modeling and Simulating Work Practice; Brahms: A Multiagent Modeling and Simulation Language for Work System Analysis and Design*, Unpublished PhD Thesis, University of Amsterdam, SIKS Dissertation Series No. 2001-10, Amsterdam, The Netherlands.

Sierhuis, M. and Clancey, W.J. (2002) 'Modeling and simulating work practice: a human-centered method for work systems design', *IEEE Intelligent Systems*, Vol. 17, No. 5 (Special Issue on Human-Centered Computing), pp.32–41.

Sierhuis, M., Bradshaw, J.M., Acquisti, A., Hoof, R.v., Jeffers, R. and Uszok, A. (2003a) 'Human-agent teamwork and adjustable autonomy in practice', Paper presented at *The 7th International Symposium on Artificial Intelligence, Robotics and Automation in Space (i-SAIRAS)*, Nara, Japan.

Sierhuis, M., Clancey, W.J., Seah, C., Trimble, J.P. and Sims, M.H. (2003b) 'Modeling and simulation for mission operations work system design', *Journal of Management Information Systems*, Vol. 19, No. 4, pp.85–129.

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

Terna, P. (1998) 'Simulation tools for social scientists: building agent based models with SWARM', *Journal of Artificial Societies and Social Simulation*, Vol. 1, No. 2, available at http://www.soc.surrey.ac.uk/JASS/1/2/4.html.

Travers, M.D. (1996) *Programming with Agents: New Metaphors for Thinking about Computation*, Unpublished PhD Thesis, MIT, Cambridge, MA.

van Hoof, R. and Sierhuis, M. (2000) *Brahms Language Reference*, http://www.agentisolutions.com/documentation/language/ls_title.htm, Available: http://www.agentisolutions.com/documentation/language/ls_title.htm.

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

Weisbord, M.R. (1987) *Productive Workplaces: Organizing and Managing for Dignity, Meaning, and Community*, Jossey-Bass Inc., Publishers, San Francisco, CA.

Winograd, T. and Flores, F. (1986) *Understanding Computers and Cognition*, Addison-Wesley Publishing Corporation, Menlo Park, CA.

Wooldridge, M. (2002) *An Introduction to MultiAgent Systems*, John Wiley & Sons Ltd., Chichester, UK.

Wooldridge, M. and Jennings, N.R. (1995) 'Intelligent agents: theory and practice', *Knowledge Engineering Review*, Vol. 10, No. 2, pp.115–152.

Website

Brahms website at http://www.agentisolutions.com.

Notes

[1]Brahms was invented at the Institute for Research on Learning (IRL) and NYNEX Science & Technology (the former R&D institute of the Baby Bell telephone company in New York, now Verizon); since 1998 development has occurred at NASA Ames Research Center, in the Work Systems Design and Evaluation group of the Intelligent Systems Division.

[2]Based on personal e-mail correspondence with Gert Jan Hofstede.

[3]The newest version of Brahms also supports the development of multi-agent software systems.

[4]Social actions are actions in which the actor takes the reaction of other actors into account. The term was introduced by the sociologist Max Weber.

[5]From http://www.answers.com/topic/imperative-programming and http://en.wikipedia.org/wiki/Imperative_programming, accessed June 14, 2006.

[6]http://www.cs.kuleuven.ac.be/~distrinet/events/e4mas/, accessed June 14, 2006.

[7]There is no logical OR-operator in preconditions.

[8]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 (the WFI-context).

[9]According to Dijkstra's shortest path algorithm (Dijkstra, 1959).

[10]URL: http://www.agentisolutions.com.