

Chapter 8

Human Activity Behavior and Gesture Generation in Virtual Worlds for Long-Duration Space Missions

Maarten Sierhuis, William J. Clancey,² Bruce Damer,³
Boris Brodsky, and Ron van Hoof⁵

¹ RIACS/NASA Ames Research Center

² NASA Ames Research Center/IHMC

⁴ SAIC/NASA Ames Research Center

⁵ QSS/NASA Ames Research Center
Moffett Field, CA 94035

³ DigitalSpace Corporation.
Santa Cruz, CA

Abstract

A virtual worlds presentation technique with embodied, intelligent agents is being developed as an instructional medium suitable to present *in situ* training on long term space flight. The system combines a behavioral element based on finite state automata, a behavior based reactive architecture also described as subsumption architecture, and a belief-desire-intention agent structure. These three features are being integrated to describe a Brahms virtual environment model of extravehicular crew activity which could become a basis for procedure training during extended space flight.

1. Introduction

Future space flight will increasingly be longer and more complex. When space missions become longer—years as opposed to months—and include humans, the training of human astronauts will be increasingly more difficult. This is not in the least because training has to be done during the mission. Although we will always train our astronauts on Earth in the basics of human space flight, mission specific training and even basic astronaut capabilities on an extended duration mission, such as donning a space suit for an extra-vehicular activity (EVA) on Mars, need also be done during the mission. Even if trained on Earth, after months of space travel previously trained tasks will be forgotten or at minimum need to be reviewed. Mission critical training will have to be done as just-in-time training during the mission. Training is one of the critical elements in safety for human space flight. Today NASA trains its astronauts for years in close to real-life training simulations of Space Shuttle and International Space Station missions. Full-fledged vehicle and mission control simulations are done for months, if not years before every mission. While training is an important aspect of mission preparedness, today's ISS astronauts have very little training in some of the most basic procedures, such as medical emergency and maintenance procedures. When new science experiments are delivered to the ISS, astronauts often have not been trained to perform these experiments. Therefore, on the job just-in-time training is already a fact of astronaut life today. As NASA missions will continue to be longer, this issue is more likely to become more difficult. We are convinced that if the training issue for

long-duration space flight is not solved, humans will always be limited in how long they can be in space.

One way to address the mission-training problem is to use immersive training facilities onboard the spacecraft and provide the astronaut with just-in-time training during the mission. We believe that virtual inhabited 3D-spaces provide a potential training solution for long-duration space flight. Virtual inhabited 3D-spaces or virtual worlds provide a potential solution to the issues of just-in-time training during long space missions, because:

- a) Virtual world systems can be relatively small², which means that they can be provided in the small spacecrafts where space will always be a constraint.
- b) Virtual world are collaborative VR environments where people can enter and interact with the virtual world and each other.
- c) Virtual world systems can provide situation-specific training solutions, because real-life teamwork situations can be simulated inside the virtual world system.
- d) Virtual world systems consist largely of software, which means that the same system can be a platform for any training domain that is needed for the mission. This allows the mission developers to develop training modules for almost any mission task.

Virtual worlds (VW) are three-dimensional virtual spaces inside a computer connected to the internet that present as “stages” with objects and characters (avatars and autonomous agents or softbots). The user enters the virtual world via the Internet as an avatar³, a representation of the human user through which the user experiences and interacts with the virtual world, its bots and other avatars. A distinct difference between virtual worlds and immersive virtual reality (VR) systems is the way the user enters the virtual space. Most VR systems are standalone environments not shared via the Internet. In immersive VR systems the user enters as him or herself. There is no representation of the user inside the system. It is as if the user is inside the system, but is not a participant in the system. Computer games use an intermediate form of user interaction. In computer games the user enters the system as him or herself, but have a limited set of interactions they can choose from; Shooting at the enemy, driving a car, flying an airplane, et cetera. Some games, most notably sports games, allow the user to interact with the game world by controlling one or more team player bots in the game. In a VW the user enters as his or her avatar, i.e. a virtual representation of him or herself. It is as if the user could be a player on the team in the game. The VW is a shared virtual environment on the Internet. The user becomes a participant in the virtual world in that he or she cannot only interact with the other characters in the VW, but the other characters (e.g. other users entering the world via the Internet) can “see” and interact with the user by interacting with their avatar. Another difference between immersive virtual reality and virtual worlds is that the objective of a VW is not to make the user forget about virtuality and make them believe that they are in the real world. Instead, the objective is to create parallel worlds for the user. One of the reasons virtual worlds are popular is that they exist on the internet and allow ordinary people to come together in a virtual meeting place with a basic computer setup (an Internet browser or a VW client program) and an Internet connection. VW

² They run on laptops in internet browsers, and do not need large VR environments.

³ The original meaning of the word avatar stems from the Sanskrit word *avataraa* which means “descent.” The word comes from a Hindu myth about the incarnation of the deity Vishnu. A more contemporary Western meaning of the word is “a temporary manifestation or aspect of a continuing entity.”

systems are not the same as other virtual reality systems in that people are entering virtual realities that do not exist outside the computer to meet and interact with other people (Damer, 1997). One interesting consequence and opportunity of this difference is that it becomes possible to think of the virtual world as an extension of the physical world of the user. Eventually this could allow a seamless integration of the real and the virtual world for a realistic team-training scenario.

1.1. Objectives

The research described in this paper has as its ultimate objective to create a development environment for creating uploadable just-in-time VR training applications for crews on their way to or on the moon or Mars. With the term uploadable we mean training applications that can be created on Earth and uploaded via space communication networks to the crew for just-in-time delivery. This would allow training modules to be developed during the mission and delivered in time for training of the crew. This is an important feature, because it gives mission developers the flexibility to continue development of procedures and mission tasks while the mission is in progress. It is advisable that for long-duration missions not every aspect of the mission has to be completely finished before the mission. For example, while the human crew is on its way to Mars specific surface tasks and experiments can still be in development, or being tested on Earth. Also, during a long-term mission situations will develop that need changes in systems and procedures. New training material will then need to be developed and uploaded to the crew to handle these situations.

We are researching the development of a general VW training development environment for the creation of mission critical crew training applications that can be delivered to the crew just in time for the needed training. Our idea is that with this environment it will be possible to develop a wide-range of virtual reality training applications for the crew on long-duration missions. To give an idea of different possible training domains for crews, think about training the crew on putting on (donning) space suits where two or more astronauts are needed to accomplish the task, training the crew on maintaining the green house on Mars or maintain the habitat life support systems. Other examples are the training of team tasks where humans and autonomous robots have to coordinate the work.

1.2. Technical Challenge

Today's advances in virtual worlds allow the developer of the virtual world to create semi-autonomous agents (bots) that have some predefined behavior. The behaviors are mostly created in a scripting language. These bot-scripting languages are low-level languages that do not easily allow for the development of sophisticated bots that have a flexible set of behavioral capabilities enabling them to react to previously undetermined situations. Today, most bots are simple finite state automata (FSA) that have a small set of well-defined behaviors, based on simple perceptual inputs and state transitions (Madsen, Pirjanian, & Granum, 1999).

Agent behavior in a Virtual World

The challenges that we are addressing deal with the aspects of avatars and softbots—or simply bots—as more or less autonomous agents, their perception and behavior in and of the world that they “live”. Madsen and Granum (Madsen & Granum, 2001) describe three different ways of developing the behavioral component of autonomous avatars and bots within a VW:

1. A behavioral component implemented by an event-based finite state automaton (FSA). This approach is what is used in most current VW environments (e.g. Active Worlds (ActiveWorlds, 2004), Adobe Atmosphere™ (Atmosphere, 2004), Ogre (Ogre, 2004)), and is supported with a “built-in” scripting language (e.g. C libraries or Java-script). The down side of the FSA approach is the difficulty in developing flexible artificial intelligence (AI) like behavior. Scripting languages, although powerful, are not suited to represent complex behavior, for the same reason that imperative programming languages are not suited for implementing AI systems, and declarative programming languages were developed for this purpose.
2. A more flexible approach is to use a behavior-based reactive architecture, more commonly referred to as a subsumption architecture. Madsen and Granum describe the Blumberg Agent Architecture for the development of their Bouncy virtual world agent (Blumberg, 1996). This architecture is based on Rodney Brooks’ subsumption-based robots (Brooks, 1991) (Brooks, 1997).
3. A third approach is using a belief-desire-intention (BDI) agent architecture, such as JAM (Huber, 1999) and dMARS (d’Inverno, Luck, Georgeff, Kinny, & Woodbridge, 2004). BDI-architectures have their roots in distributed artificial intelligence, in which agents operate with explicit plans and goals that are activated based on pre- and post-conditions in production rules matching on the beliefs of an agent. The beliefs of an agent are symbolic representations of the agent’s correspondence to information it has about the world.

Madsen and Granum describe these three approaches as possible alternatives for describing avatar behavior. From our experience in modeling human behavior, we have independently come to the conclusion that there is a need for combining all three approaches in one unified avatar and bot behavior model. Figure 4 shows how the behavior of an autonomous avatar or bot can be divided into three behavioral components.

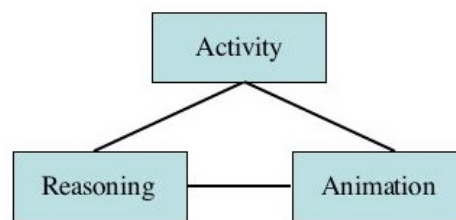


Figure 4. Model of autonomous avatar or bot behavior

The animation component describes the behavior of the agent (an autonomous software agent) within the graphical world. In this component the externally visible behavior of the agent is generated. This component is what distinguishes a bot from a software agent. Simple actions are visualized with scripted animations of a virtual character. These animation scripts are created using a FSA approach, usually using the VW environment’s scripting language. Complex behaviors are visualized by combining basic animation scripts into a more complex behavior. For example, to visualize an agent sitting down and drinking a cup of coffee we sequentially combine an animation of a bot sitting down and reaching for an object, with an animation of the bot grabbing the coffee cup object and then bringing the coffee cup to its mouth, and lastly an animation of showing the bot drinking from the cup. The resulting animation is a visualization of the agent’s drinking from the cup activity.



Figure 5. Astronaut crew agent enacting a make and drink coffee animation sequence inside the virtual FMARS habitat (ARC)

The activity component describes how particular behaviors are exhibited. This component describes what behavior in the animation component should be executed, for how long and in what context. The best way for describing the agent's behavior in this component is with a behavior-based subsumption approach. Unlike in a plan and goal-based approach used in most BDI-architectures, a bot's behavior is not always goal-driven. Modeling people as bots within a virtual world requires us to model real-world behavior based on interactions with other bots and avatars. There arise many situations in which people's behavior is not goal-oriented or intention driven. Clancey describes how human behavior is often related to motives and cultural norms (William. J. Clancey, 2002). Imagine standing waiting with a group of people to go outside. While you're waiting you take the opportunity and take a picture of the group. Although you could ultimately describe the waiting activity with a plan and goal approach (as you can describe it with an FSA approach), it seems strange and unnatural to describe the taking of the picture while you're waiting as a sub-goal or plan of the "waiting plan." A more flexible and natural approach to describing the "waiting activity" is a hierarchical subsumption-based organization of competing activity behaviors where, for example, the competition between the "standing still and do nothing" and the "take a picture while I am waiting" activities is decided based on a combination of reactive and motive-driven reasoning paradigms.

The reasoning component describes when particular behaviors are exhibited. To continue with the "taking a picture while you are waiting" example, this component determines how the competition between the "do nothing" and "take a picture" activity is resolved in a particular situation. Behavior is situated and it depends on the situation in which the waiting occurs how one would behave. The reasoning behavior of the agent described in this component correlates the agent's individual perception of the situational context with the possible activity behavior, such as where the waiting occurs, what role the agent believes it plays in the group, the purpose of the group going outside, the agent's motive, the group norms and culture, et cetera. Representing this type of reasoning behavior is best done using an agent-based BDI-approach in which an agent has its own belief-set. An agent can reason over its belief-set and the result of this reasoning behavior could be a decision for what activities in the activity component to execute. The reasoning component should be reactive as well as deliberative. A belief-based approach allows both, since the way an agent can obtain new beliefs in its belief-set can include not only forward or backward reasoning, but also the assertion of new beliefs via communication with other agents and perception of real-world events, enabling reactive behavior.

The Brahms Virtual Environment (BrahmsVE) uses this three-component model for representing autonomous avatar and bot behavior. We describe the implementation of this behavior model in more detail in subsequent sections in this chapter.

2. The EVA Prep scenario

This section describes a scenario of a BrahmsVE model, called FMARS, developed in 2002. The FMARS model consisted of two independent scenarios from Crew 1 of the Flashline Mars Arctic Research Station (FMARS) in which a crew of six people lived and worked for a week, conceptually as well as physically simulating a Mars surface mission (W.J. Clancey, 2002) (Clancey, Sierhuis, Damer, & Brodsky, in press). The scenario is about the activities and coordination of three crewmembers during the preparation activity for an Extra-Vehicular Activity (EVA) outside the FMARS habitat. We refer to this as the EVA Prep scenario. A Brahms model was developed for this scenario, showing what the crew did to get ready for an EVA and donning their space suits. The scenario was developed based on video clips and photographs—taken by Clancey, who was one of the FMARS crewmembers—of an actual EVA preparation activity during the Crew 1 mission. Clancey’s research as an ethnographer during this crew rotation was to study the daily life of the FMARS crew (Clancey, 2001).

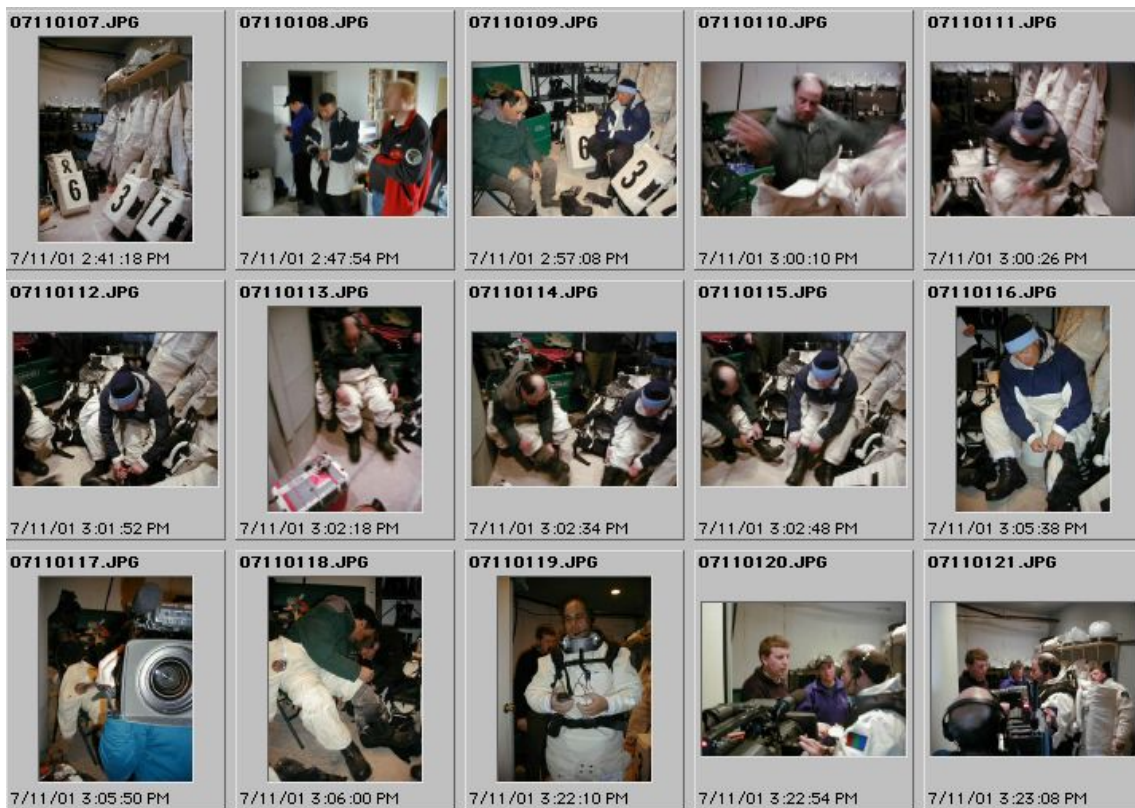


Figure 6. Clancey’s chronological EVA Prep photograph record

The resulting EVA Prep model was the culmination of Clancey’s ethnographical analysis of the EVA preparation activity, Brodsky’s representation of this analysis in a Brahms model and Damer’s virtual world design team’s creation of the Adobe Atmosphere FMARS habitat Virtual World and the OWorld bot animation models.

One of the first activities in the design of the EVA Prep scenario is the description of the scenario in plain English (the reader should refer back to Figure 6 while reading this description in Table 1):

Table 1. Scenario description in plain English

Notation:	
Object	– Italicized
<u>Areas</u>	– Underlined
Activities	– Bolded
Agents	– Italicized and bolded
[...]	
The Crew then proceeds with suit donning .	
The Crew walks to the <u>suit closet area</u> (not really a closet, just hanging there, see pictures), and finds her suit.	
The Crew then walks to the <u>shelf area</u> , selects boots (not person-specific, selected by size, assume any size is always available).	
The Crew then puts on suit bottom (stepping into suit and pulling it up to pants level. Probably to snapshots would suffice for now).	
The Crew then calls for an available Helper for assistance (in putting on boots and gaiters).	
The Helper responds (assume he is present in the EVA prep room), and checks if an empty chair is available. If not, he brings a chair (from the wardroom table. Again, don't have to show the lab area and the process of picking. Lets just have the Helper leave the room, and then come back holding the chair). He then puts the chair down. (Where exactly to put down the chair is something TBD. Need to settle on a “tiling” solution that wouldn't overburden the simulation with a generic array of areas, but rather just define a few that would be used throughout the simulation, perhaps even reusing the same area for multiple purposes.)	
When both the Helper and the chair are in place, the Crew walks to the chair (located at a pre-specified area, see above), sits down on it, and puts on boots. (Includes both pulling on and fastening them. For this and subsequent EVA prep fragments, we will need to browse through pictures to find how the Helper actually assists, in this case in putting on but. That may require some unique gestures. At a minimum, the Crew should be once seated feet on the ground, and another time with one foot in the air, with and without the boot. The Helper would be standing nearby holding the boot, then kneeling holding the boot, then kneeling not holding the boot... Just a speculation).	
When the Crew has her boots on, the Helper puts on gaiters on her.	
The Crew then gets up from the chair, and puts on jacket to wear under the suit. (No help necessary).	
The Crew then puts on suit top (pulling the suit up and donning the upper part over torso and arms. (Looks like two pictures: The suit pulled up, and the suit over torso and arms [...])).	
The Crew then calls for an available Helper for assistance (in zipping up the suit back).	
The (available) Helper walks over to the Crew chair, and zips up the suit back. (Showing the helper standing behind the crew, holding arms up towards the crew's back).	

The photographs and the scenario description are important starting data for the development of the Brahms model, as well as the FMARS Virtual World model and the animation models for the bots in the virtual world.

Next, we will describe the three behavioral components of the BrahmsVE architecture, using this scenario as our example.

3. Brahms Virtual Environment

The Brahms Virtual Environment (BrahmsVE) is a combination of different technologies developed at NASA Ames, DigitalSpace Corporation, and Adobe. The environment, shown in

Figure 7, incorporates the Brahms multi-agent language and virtual machine (developed by NASA Ames) for the Atmosphere™ interactive 3D multimedia creation and delivery platform for the Web (developed by Adobe), using the OWorld engine (developed by DigitalSpace).

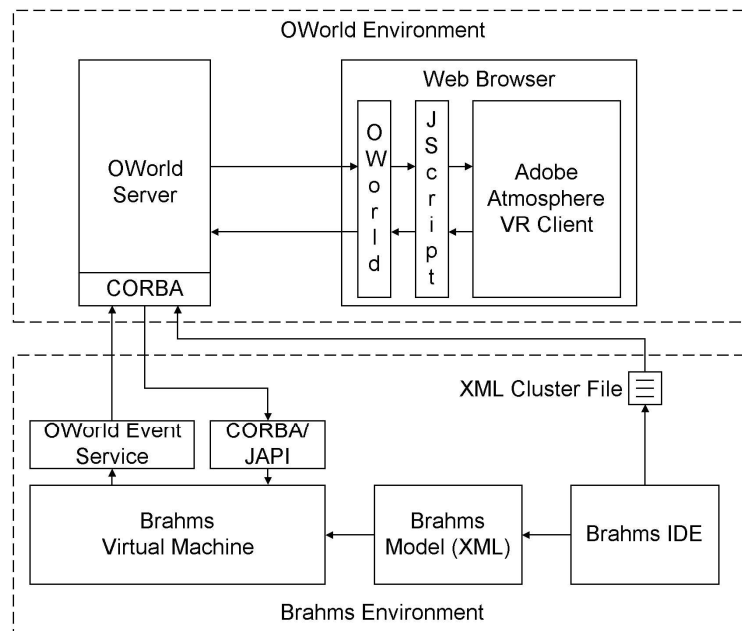


Figure 7. BrahmsVE Architecture

3.1. The Agent Behavior Model—Brahms

Brahms is a multiagent modeling and simulation language, earlier described in (Clancey, Sachs, Sierhuis, & van Hoof, 1998)(Sierhuis, 2001) (William. J. Clancey, 2002) (Sierhuis, Clancey, & Hoof, Submitted). The Brahms environment is in particular developed to model behavioral patterns of people engaged in purposeful activities—what people do—something which has been referred to in the past as people’s practice (Schön, 1982)(Lave, 1988)(Suchman, 1987). Without describing the Brahms language in detail—for that we refer you to (Sierhuis, 2001) (Sierhuis et al., Submitted)—we give a description of the important aspects of the Brahms language that allows us to model people’s behavior as purposeful and interleaved daily activities. The Brahms language implements the two components of the behavioral model shown in Figure 4—the activity and reasoning components—and is what makes Brahms a good autonomous avatar and bot-language for virtual 3D environment.

Brahms is a multiagent BDI language. Every agent has its own set of beliefs that directs the agent’s reasoning and activity components. Agents can communicate their beliefs enabling the reaction of an agent to another agent’s communication (in terms of reasoning and/or activity execution). Brahms agents are also reactive to the detection of state changes in and of their world environment. Agents are situated in a modeled geographical world with objects that can be represented with state changing behavior. Object states are modeled as facts in the world, and agents can detect these facts in certain situations, making them beliefs for the agent and allowing them to react to them as soon as they are created. The Brahms engine or virtual machine (VM) is based on an activity-subsumption architecture, driving flexible reactive activity behavior of agents. The workings of this architecture will be discussed next.

The Activity Component in Brahms

Activities are the most important construct in the Brahms language. All agent behavior has to be modeled as an activity. There are three different types: primitive, composite and Java activities. All activities have a user-defined name representing a behavior defined by the modeler. According to our theory of activities (Sierhuis, 2001) (William. J. Clancey, 2002), the name of an activity should be the name of an observed behavior of a person in the real-world that the agent represents, but there is no rule in Brahms that states that the agent has to represent a person and that this has to be a person in the real world. It is the responsibility of the modeler to decide the relevance of the model to the system behavior that is being modeled. This allows the use of the Brahms language in any domain and for any purpose, including, but not restricted to, modeling social phenomena, human behavior, and software agent behavior. Activities have a defined beginning and end determining the duration of the activity. Just as people in the real world, when and how a Brahms agent performs an activity depends on the agent's context in a particular activity, and the actual performance and duration of the activity emerges during the execution of the model, based on the changed belief-state of an agent.

The design of an activity is one of the most important activities of a Brahms modeler. Here we describe the design of the Space Suit Donning composite activity from the EVA Prep model description from Table 1 (see Table 2).

Table 2. Scenario activity design

Notation:	
Object and attributes	– Italicized
<u>Areas</u>	– Underlined
Activities	– Bolded
Agents and groups	– Italicized and bolded
<Comments>	– In angular brackets
<Helpers assist the crew members who are behind (in procedure steps) first>	
<Each EVA_Crew has an associated conceptual object EVA_Procedure , which attribute values are facts (detected by helpers to know who is behind).>	
<Wherever “EVAPrepRoom” is specified as a location, always replace with a more specific one>	
<Below, an individual EVA_Crew member is denoted AgentC , and an individual EVA_Helper – AgentH . Selected individual objects are, for simplicity, category names suffixed with ‘C’ (e.g. “HelmetC”)>.	
SuitDonning <composite activity>	
EVA_Crew performs	
MovingToLocation (<u>SuitClosetArea</u>) <any better name?>	
FindingSuit <SuitC (suits – person-specific)>	
MovingToLocation (<u>ShelfArea</u>)	
SelectingBoots <BootsC, by size (assume any size is always available)>	
PuttingOnSuitBottom <step into suit and pull it up to pants level>	
broadcast CallingAvailableHelper	
< AgentC .needHelper = true>	
when AgentH .availableToHelp = true	
MovingToArea (<u>ChairCArea</u>)	
SittingDown	
PuttingOnBoots <including fastening boots>	
< AgentC contain BootsC>	
HavingGaitersPutOn	

```

        when AgentC contains GaitersC
            GettingUpFromChair

PuttingOnJacket <to wear under suit>
    <AgentC contain Jacket>
PuttingOnSuitTop
    <pulling suit up and donning the upper part over torso and arms>
    <suitTopOn = true>
HavingSuitBackZippedUp
    when suitBackZipped = true
        FindingTools
            <RZ has the Shovel,
            KQ has a Camera around her neck,
            VP has a Checklist around his neck>
        AttachingToolsToSuit
PuttingOnCamera <should've been brought in from the stateroom>
PuttingOnChecklist <both are put around neck>

EVA_Helper performs
    when AgentC.needHelper = true
        when not AgentH.location = EVAPrepRoom
            MovingToArea(EVAPrepRoom)
        when not (chair.location=EVAPrepRoom) & (chair.available=true)
            BringingChairs <composite activity>
                MovingToArea(WardroomTableArea)
                get SelectingChair
                    <AgentH contain ChairC>
                MovingToArea(EVAPrepRoom)
                MovingToArea(ChairCArea) <decided on in advance>
                put PuttingChairDown
                    <AgentH contain ChairC is false>
            communicate ShowingReadinessToHelp
                <AgentH.availableToHelp = true>

        when AgentC contains BootsC
            PuttingOnGaiters <over boots, with AgentC still sitting>
            <AgentC contain GaitersC>

        when AgentC.suitTopOn = true
            ZippingUpSuitBack
            <AgentC.suitBackZipped = true>

```

The design of the EVA Prep model is divided in a number of sub-models: Agent-, Object- and Geography Model. The Agent Model defines the agents, the agent groups and the activities. The Object Model defines all the objects in the physical world (e.g. a chair, the spacesuit), as well as the conceptual objects the agents use in the reasoning component (e.g. a procedure). The Geography Model defines conceptual locations that connect to actual locations in the world.

Agents and Groups

The scenario design in Table 2 shows that there are two types of FMARS habitat crewmembers involved in the spacesuit donning activity. The first group includes those habitation crewmembers that are to go on an EVA (the EVAGroup). Obviously, these habitation crewmembers are the ones that will have to don a spacesuit. The second group includes those habitation crewmembers who help the EVA crew to suit up and get ready for their EVA (the EVAHelpers). In Brahms an agent can be a member of multiple groups and inherit from all groups it is a member of. The Agent Model for the scenario in Table 2 is presented in Figure 8.

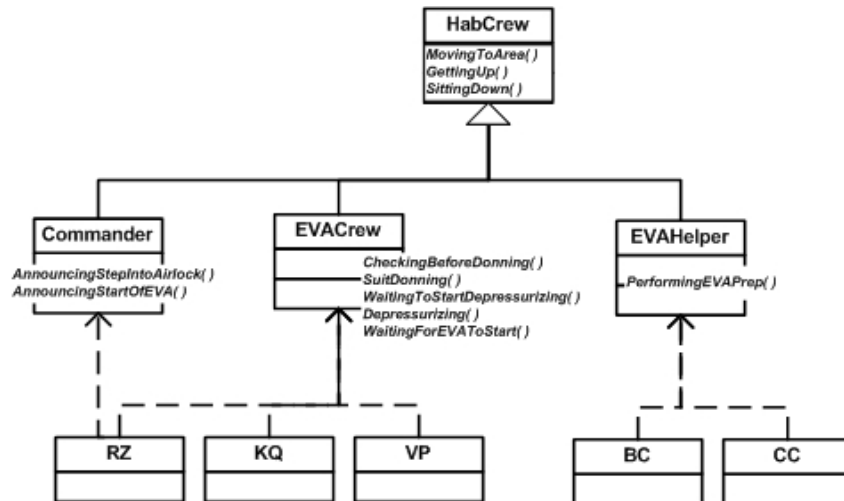


Figure 8. Agent Model

Figure 8 shows the groups and agents in an inheritance hierarchy with the names of the, by the agents inherited group activities. Agent RZ is a member of three groups, HabCrew, Commander and EVACrew, and inherits all the activities from these three groups (the compiler resolves the multiple-inheritance conflicts from the HabCrew group). Agents KQ and VP are members of HabCrew and EVACrew and thus inherit the activities from these two groups, but not those from the Commander group. Agents BC and CC are EVAHelper agents and also HabCrew agents, and thus inherit these group's activities. Table 2 decomposes the SuitDonning() composite activity into more detail. Figure 8 only shows the top-level activities, including the SuitDonning() activity of the EVACrew group.

Defining the inherited activities in the model does not immediately allow the agents to execute those activities. Defining the activities means that the agent has the potential to perform them, but when they will be performed has to be specified in situated-action rules called workframes. Workframes are also specified in the Agent Model, either at the group or agent level (they are not shown in Figure 8). Workframes are of the form

```

When (belief-conditions are true)
Do
    Activities, and
    conclude new beliefs for the agent and/or facts in the world
EndDo
  
```

Workframes are declarative production rules matching on the belief-set of the agent executing it. This means that if the belief-conditions in a workframe match a belief in the belief-set of the agent, the condition evaluates to true, and the variables in the condition are bound to the agent or object that is the object of the belief (see (Sierhuis et al., Submitted), (Sierhuis, 2001), and (van Hoof & Sierhuis, 2000) for a detailed explanation of precondition matching). Table 3 shows the PerformSuitDonning workframe in Brahms source code (bold characters are keywords).

Table 3. Workframe PerformSuitDonning

```

workframe PerformSuitDonning {
  variables:
    forone(Commander) commander;
    collectall(EVAHelper) evaHelper;
  detectables:
    detectable NoticeAvailableHelpers {
      detect((evaHelper.available = true))
      then continue;
    }
    detectable StartDepressurizing {
      detect((commander.stepIntoAirlock = true))
      then abort;
    }
  when(
    knownval(evaHelper.memberEVAHelper = true) and
    knownval(commander.memberCommander = true))
  do {
    CheckingBeforeDonning();
    SuitDonning();
    conclude((current.suitedUp = true));
    AnnouncingSuitedUp();
    WaitingToStartDepressurizing();
  }
}

```

The statements in the body of the workframe (do { } section of the workframe in Table 3) are executed in sequence from left to right and top to bottom. Figure 9 shows the agent execution timeline for agents RZ and CC from one simulation run, displayed in the Timeline View of the Composer application⁴. This timeline shows in what locations the agent is and is moving to at what time—this is shown as the top bar (EVAPrepRoom). You can see that agent RZ moves for a short time to a different location (Shower) during the CheckingBeforeDonning composite activity to fill his water bag.

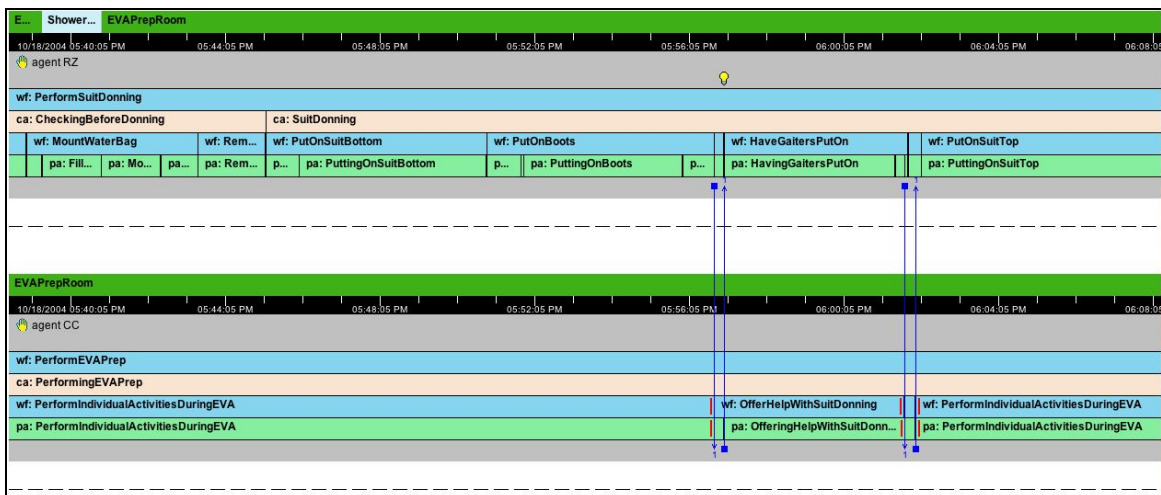


Figure 9. EVACrew and EVAHelper Agent Timeline

⁴ The Composer is an interactive design environment (IDE) for the Brahms language. It is a modeling, programming and simulation execution environment.

Bellow the location bar you find the time bar. This provides the time at which things occur. The Timeline View allows the user to zoom in and out of the timeline. The time bar in Figure 9 is at a zoom-interval of 2 minutes. This means that every white tick interval on the time bar represents 2 minutes of simulated time for the agent. It is not possible to see in Figure 9, but the total PerformSuitDonning workframe from Table 3 takes 1 hour, 33 minutes and 25 seconds. Bellow the black time bar you find the name of the agent, next to the agent icon that looks like a little hand. Bellow that you see the workframe-activity instantiation represented over time. This provides a hierarchical overview of the workframes and activities that the agent is executing at every simulation clock tick. It shows when and how long workframes and activities within it are being executed. For example, Figure 9 shows that agent RZ is executing the PerformSuitDonning workframe for the entire figure and agent CC is executing the PerformEVAPrep workframe at the same time. These activities are of type composite activity.

Composite activities are activities that are decomposed into lower-level subactivities, workframes and thoughtframes (see Figure 10). A primitive activity is an activity that is not further decomposed. It can be used to represent an action in the world that is not further decomposed. Primitive activities have a specified maximum or random duration. This is different from a composite activity in that it has a pre-specified duration. In contrast, the duration of a composite activity depends on the duration of the subactivities executed within it (note that thoughtframes have no duration—see the small light bulb at the top of the agent RZ timeline in Figure 9).

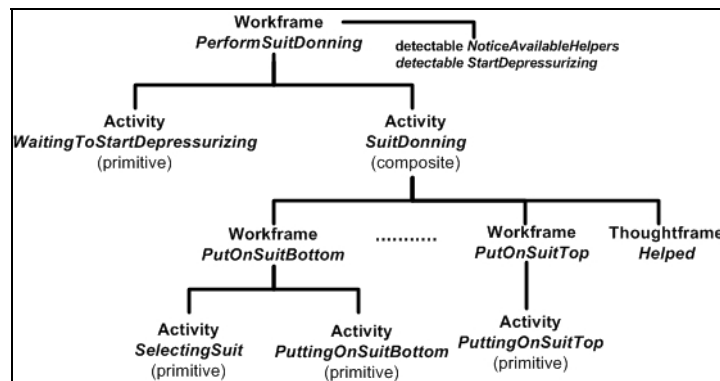


Figure 10. PerformSuitDonning Workframe-Activity Hierarchy

Primitive activity duration is determined at the start of its execution—either randomly chosen 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 instance⁵ (WFI) in which the activity is being called. Each WFI is in one of the states shown in Figure 11. The state of an agent’s activity behavior is defined by the combined sets of available, working, interrupted, and interrupted-with-impasse WFIs at any moment in time.

⁵ When a workframe (or thoughtframe) is fired (i.e. the preconditions are matched against beliefs in the agent’s belief-set) a workframe instance is created for every workframe variable context that matches all preconditions. Each workframe instance is now an independent version of the workframe and will be executed independently from each other, with different variable bindings (determined by the WFI-context).

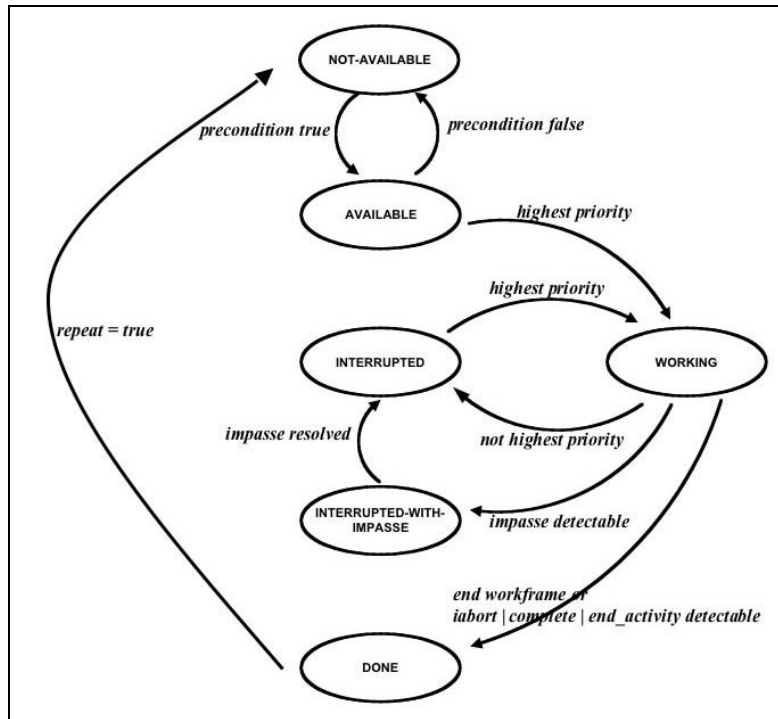


Figure 11. State transition diagram for workframe instances

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 activity execution sequence 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 priority of the workframe, if given, or the highest priority of the activities called within the workframe body. The default priority is always zero. The work selector process in the agent's inference engine determines which of the available, working and interrupted WFIs have the highest priority (see Figure 12). This one is moved to the working state and is executed by the work executor process. 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.

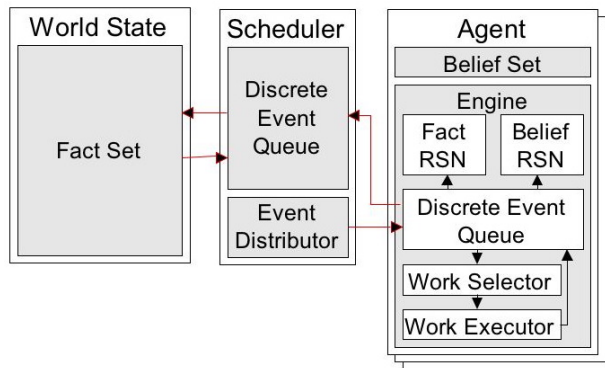


Figure 12. The Agent's Inference Engine

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. Table 3 shows the declaration of the NoticeAvailableHelpers and StartDepressurizing detectables. A detectable defines that if the agent detects a fact in the world this fact becomes a belief of the agent (this is accomplished via the event distributor in Figure 12: the fact is part of the fact set in the world state). 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 StartDepressurizing 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, reasoning or communication) that the EVACrew agent’s next subactivity is to step into the airlock, it will abort the working workframe, which means it will end the activity SuitDonning.

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

Activity Subsumption Architecture

An important aspect of the Brahms activity paradigm is that activities are not the same as functions and procedures in imperative languages (Pratt & Zelkowitz, 1996). Imperative languages use a computer memory-based program stack to keep track of function calls. When a function is executed, the function’s context is ‘pushed onto the program stack. When in a subfunction the program is not also still in the context of the ‘parent’ function. Thus the program cannot move execution back and forth between a function and its subfunctions that are called. Function execution is sequential and cannot be interrupted. Not so with activities. In contrast, Brahms activities stay active while they are being executed.

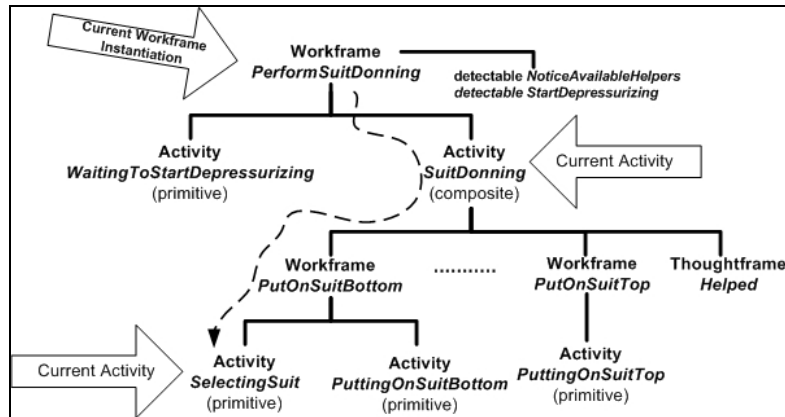


Figure 13. PerformSuitDonning Activity Subsumption

Thus, if a subactivity in a workframe of a composite activity gets executed, the ‘parent’ composite activity is still active (see Figure 13). All workframes, thoughtframes and detectables in the ‘parent’ activity are still being evaluated while the agent is executing the subactivity. This is part of the Brahms subsumption architecture (Brooks, 1991), and is based on the principle that humans are always multi-tasking by being in a hierarchy of activities at the same time. For example, the EVACrew member is also still in the activity of SuitDonning when it is in the activity of SelectingSuit. Thus, every workframe, thoughtframe or detectable in the current activity hierarchy is part of the agent’s context, and can be fired at any moment, changing the belief and behavioral state of the agent. However, at the same time workframes outside of the current activity tree also have the potential to fire, enabling an activity-context switch for the agent (e.g. the workframe PutOnSuitTop in Figure 13).

In a Brahms simulation, an agent may engage in multiple activities at any given time, but only one activity in one workframe is active at any one time. At each event the simulation engine determines which workframe should be selected as the current working, based on the priorities of available, current and interrupted work (see Figure 11). The state of an interrupted or impassed workframe is saved, so that the agent can continue an interrupted workframe with the activity that it was performing at the moment it was interrupted.

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

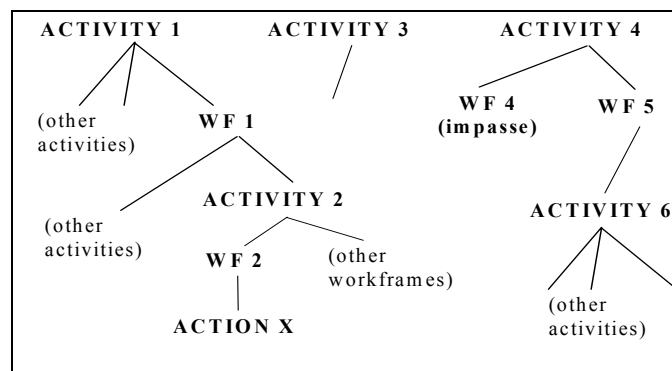


Figure 14. Multi-tasking in Brahms

An illustration of this is given in Figure 14. An agent (not shown) in a running model will have multiple competing 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 3 and Activity 4. Activity 2 similarly has competing workframes. Priority rankings led this agent to follow the path to Action X, but interruptions or reevaluations may occur at any time. Activity 3 has a workframe that is potentially active, but the agent is not doing anything with respect to this activity at this time. The agent is doing Activity 4, but reached an impasse in workframe WF4 and has begun an alternative approach (or step) in workframe WF5. This produced a subactivity, Activity 6, which has several potentially active workframes, all having less priority at this time than WF2.

The Brahms subsumption architecture allows two forms of multi-tasking. The first form is inherent in the activity-based paradigm; Brahms can simulate the reactive situated behavior of humans. An agent's context forces it to be active in only one low-level activity. However, at any moment an agent can change focus and start working on another competing activity, while queuing others. Having the simulation engine switch between current and interrupted work for each agent, simulates this type of multi-tasking behavior as represented in Figure 14. The second form is subtler. People can be working concurrently on many high- and medium-level activities in a workframe-activity hierarchy. While an agent can only execute one primitive activity in the hierarchy at a time (e.g. ACTION X in Figure 14), the agent is concurrently within all the higher-level activities in the workframe-activity hierarchy. For example, the agent in Figure 14 is concurrently within Activity 1, Activity 2 and primitive activity Action X, Activity 3, Activity 4 and Activity 6. It should be noted that while a workframe, and its associated activities are interrupted or impassed, the agent is still considered to be in the activity. The agent is conceptually within all current, interrupted and impassed activities.

To exemplify this subtle multi-tasking behavior of an agent, we draw the reader's attention to EVAHelper agent CC's timeline in Figure 9. The agent has two simultaneously competing workframes in the PerformEvaPrep composite activity, namely workframe PerformIndividualActivitiesDuringEva and OfferHelp-WithSuitDonning. The agent starts with executing the PerformIndividual-ActivitiesDuringEva workframe. This workframe gets interrupted when agent RZ asks for help (see communication line from agent RZ to agent CC in Figure 9). At that time agent CC starts executing workframe OfferHelpWithSuitDonning. When the agent is done with this workframe the interrupted workframe becomes active again, and the agent continues executing the PerformIndividualActivitiesDuringEva workframe there where it left off before its interruption. This workframe interruption is shown in the timeline in Figure 9 with red vertical bars at the beginning and end of the workframe's interruption.

The Reasoning Component in Brahms

The reasoning component consists of production rules for agents and belief conclusions in workframes. Production rules in Brahms are forward chaining inference rules acting on the beliefs of an agent. These rules are called thoughtframes, because using these rules an agent can 'think' and deduce new beliefs while in an activity. Each agent has a set of thoughtframes, a combination of thoughtframes locally declared and inherited. Table 4 shows a thoughtframe.

Table 4. Thoughtframe

```
thoughtframe Helped {
  repeat: true;
  variables:
    forone(EVAHelper) evaHelper;
  when(
    knownval(current.currentHelper = evaHelper) and
    knownval(evaHelper.helping = true) and
    knownval(current.needHelp = true) and
    knownval(current.beingHelped = false))
  do {
    conclude((current.beingHelped = true));
  }
}
```

A thoughtframe consists of a number of elements which we will describe using Table 4. First of all, a thoughtframe is used to infer new beliefs based on current beliefs in the belief-set of the agent. New beliefs are created when a thoughtframe executes. The conclude statement in the do-part or body of the thoughtframe creates a new belief for the agent. Table 4 shows a conclude statement of the form (O.A = v), where O = 'current', A = [the 'beingHelped' boolean attribute of the agent] and v is the outcome of a boolean expression that is evaluated before the belief is created (v= true).

When the agent has one or more beliefs that are matching all the preconditions the thoughtframe is immediately executed. Using this approach we can represent the forward-reasoning behavior of an agent; the conclude statement in one thoughtframe can trigger the execution of a subsequent thoughtframe or workframe, thus creating a 'forward chaining' of belief-set changes simulating the reasoning behavior of a person. Every time the agent gets a new belief, only those thoughtframes are evaluated that have a precondition that is a potential match on the newly created belief. This makes the reasoning behavior efficient, because for every belief change event in an agent only a small number of preconditions have to be evaluated.

The activity and reasoning components in Brahms are integrated in a way that allows the modeler to represent reasoning in the context of an activity. This situated reasoning is accomplished with composite activities. Composite activities consist of both workframes and thoughtframes. Figure 13 shows how the Helped thoughtframe is part of the SuitDonning composite activity. The agent can only execute the thoughtframe when performing the composite activity. Workframes can also create new beliefs (and facts) in the world (using a conclude statement). This allows the modeler to create belief changes and world-state changes (facts) of the agent, representing consequences of activity execution in a workframe. In other words, situated reasoning of an agent is done in context of the current activity and can be represented both as a consequence of the activity (as belief-consequences in workframes) and as a deliberative reasoning process during the activity (as thoughtframes).

3.2. The Animation Component—BrahmsVE

The animation component presents the Brahms agents and objects in a 3D virtual world. Just as the reasoning and behavioral components, the animation component is a model that is developed based on the scenario description. However, importantly, the animation component needs to be developed in close connection with what is in the behavioral model. The activity behaviors represented in the Brahms model need to be animated via the animation component. The Brahms agents and objects need to have an embodiment as bots inside the virtual world. With embodiment comes the notion of a geography model that needs to correspond with a cluster in the virtual world. A cluster consists of a collection of locales, and is a 3D model that corresponds to a Brahms Geography model. A locale is developed based on an animation storyboard created from the scenario description and the photo and/or video data available. The development cycle of all these models is represent in Figure 15. The next sections describe how these models are developed.

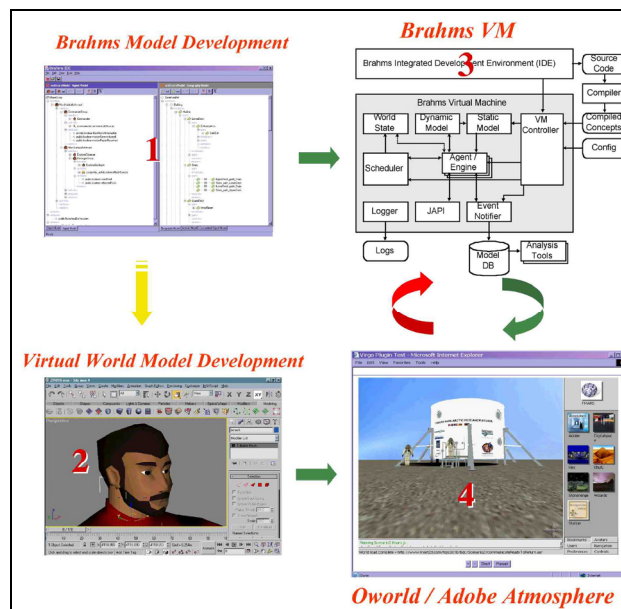


Figure 15. BrahmsVE Modeling Cycle

The Geography Model

The Geography Model specifies the location where agents perform activities and objects can be found. The Geography Model consists of a set of areas representing confined actual or conceptual spaces where agents and objects can be located. Areas can be contained inside other areas forming a conceptual hierarchical containment definition of space in the real-world. Area hierarchies enable the representation of spaces such as the FMARS habitat, containing floors that contain rooms and specific locations inside a room. Figure 16 shows the area hierarchy for the EVA preparation room on the lower deck of the FMARS habitat where the spacesuit donning activity is performed.

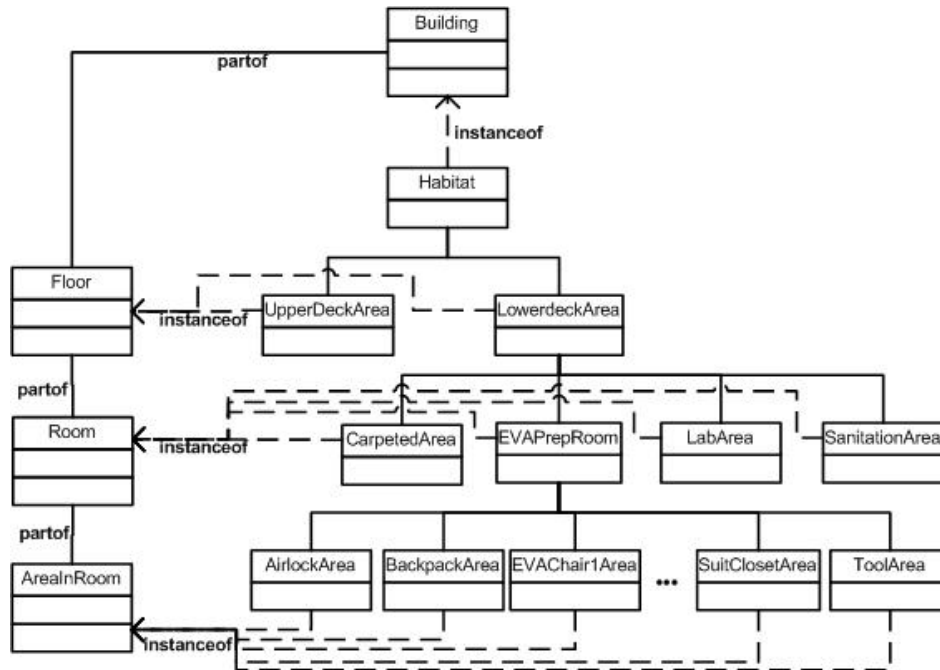


Figure 16. Habitat Geographical Area Hierarchy

The Brahms modeler is responsible for specifying the geography and placing agents and objects in the geography in their initial locations at the start of the scenario (see the agent inhabitants BC, CC, KQ, RZ and VP in Figure 17). The geography is to be visualized in a three-dimensional view by OWorld. OWorld displays the initial geography or scene, and is the virtual reality environment (see Figure 7) that consists of components that can visualize a Brahms geography in a three-dimensional view and can visualize the activity movement and behaviors of agents and objects in this view.

The Brahms geography model consists of Brahms source code not consisting of any specifics on how the geography, agents and objects are to be visualized in a three-dimensional view. The Brahms developer uses the Composer to design the geography and position the agents and objects within the geography. The Composer stores this information in a XML format parsable by Oworld (see Figure 17). The file contains information on how to organize and visualize the areas, agents and objects and where to position them.

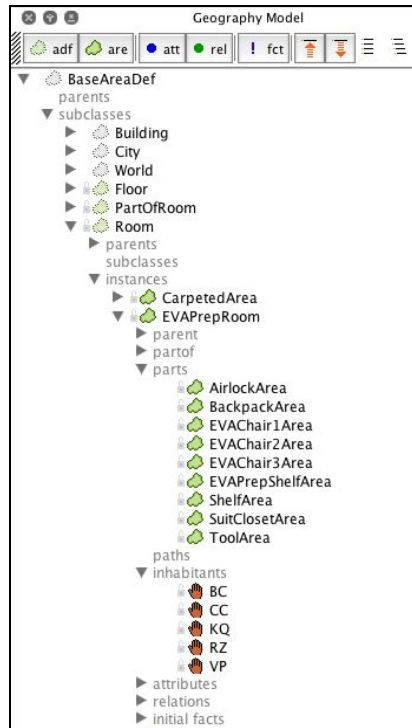
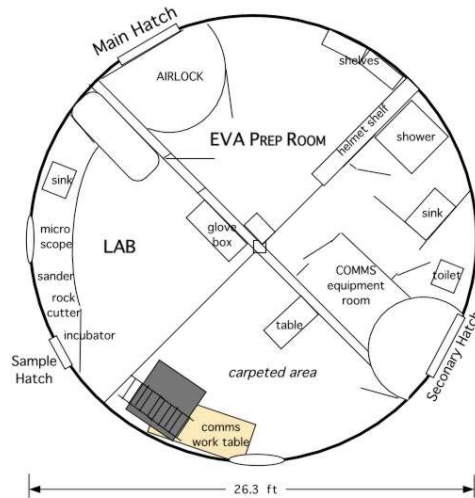


Figure 17. EVAPrepRoom Area Geography Model

The Virtual World Model—OWorld

Areas are worlds or locations in a world that can represent various types of living spaces, possibly organized in a containment hierarchy. Agents and objects are entities that have location, can move around and can interact with each other. Some of the areas in Brahms models are grouped into more high-level areas, where a number of sub-areas are located fairly closely together and in which agents and objects are confined to the high-level area. A high-level area like this has its own three-dimensional visualization and is called a Locale. An example of a locale is the lower deck of the FMARS habitat shown in Figure 18, first as a drawing, then as actual photographs from within the FMARS habitat and finally as a 3D locale inside Adobe Atmosphere. This locale is represented in Brahms as shown in Figure 16 and Figure 17. The collection of locales with their visualizations and positioning of areas, agents and objects is called a cluster. An example of a cluster is the complete geography model for the EVAPrep model. OWorld does not render a cluster all at once, but renders locales and a user generally views only one locale at a time.



Flashline Mars Arctic Research Station
Lower Deck, as built July 2000

(a)



Figure 18. (a) FMARS Lower deck Drawing (b) actual carpeted + lab areas (c) locale of Lowerdeck

In order for OWorld to render a locale, it needs to know how and where to render the various Brahms concepts. This information is specified using XML. OWorld loads in the cluster XML file and can render the default locale using the information specified in the file. OWorld will represent agents and objects as virtual bots. Areas will be represented as points. A point is a labeled location in a locale (a sub-area in the Brahms model). A point represents one specific X, Y, Z coordinate and is used to represent a named Brahms areas—parts—in a locale. The scene graph will have an accurate representation of the area and its size (see Figure 18c showing the carpeted-, lab- and EVA prep room). The point that represents the area merely serves as a point for agent movement used during a simulation to map an area to a location in OWorld. Brahms agents and objects move from one area to another without the use of a coordinate system, while OWorld moves the agent and object bots according to the VW coordinate system. The area to point mapping allows the generation of movement in the VW renderer (Adobe Atmosphere or Ogre). How this is actually done is described in the next section.

OWorld Events and Activity Animation

There are currently three types of events that can be used to visualize a Brahms model simulation in OWorld. These are described in Table 5.

Table 5. OWorld Event Types

Event Type	Description	Format
setVisible	A bot or icon can be set to be visible or invisible by using the setVisible event.	setVisible time bot visible
activity	A Brahms agent can execute an activity via its virtual representation (bot). This event is used to have the bot execute an animation script animating the activity of the bot in the virtual world. Examples are getting up, walking, standing somewhere, opening a door, talking to someone, etc.	activity activitytype starttime endtime who activityname arguments
setLabel	The label of a bot can be changed by using the setLabel action.	setLabel starttime endtime labeltype displaytext font fontsize colorR colorG colorB

Agent activities that are simulated in the Brahms VM are recorded as OWorld activity events and rendered in the VW by OWorld. For example, during the SuitDonning composite activity agents RZ and CC their activities (Figure 19) generate the activity events in Table 6 (only showing the events for the first half of the Figure 19).

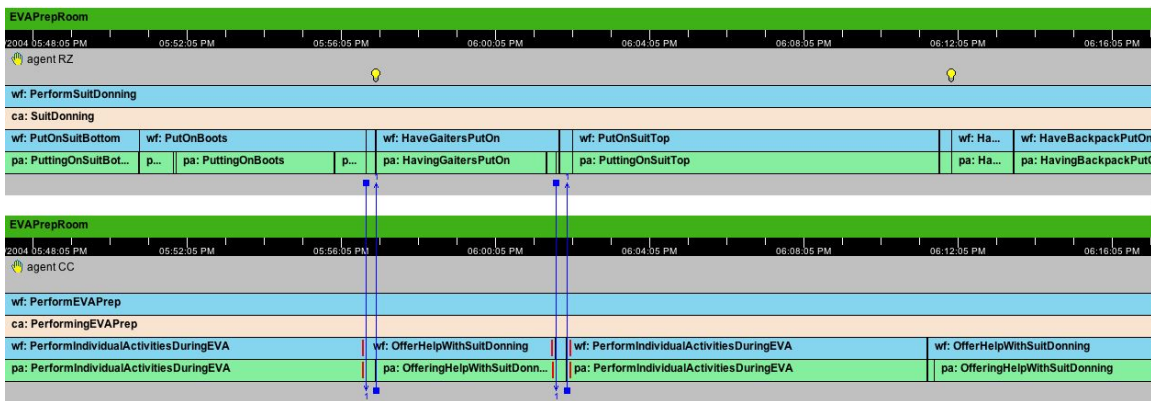


Figure 19. Agent RZ executing the CheckingBeforeDonning composite activity

Table 6. Brahms-OWorld events generated from the CheckingBeforeDonning activity

```
activity|primitive|803|1044|projects.EVAPrep.RZ|PuttingOnSuitBottom|
activity|get|1044|1088|projects.EVAPrep.RZ|SelectingBoots||projects.EVAPrep.BootsRZ|projects.EVAPrep.Shelf
Area
activity|primitive|1088|1093|projects.EVAPrep.RZ|SittingDown|
activity|primitive|1093|1358|projects.EVAPrep.RZ|PuttingOnBoots|
activity|primitive|0|1397|projects.EVAPrep.CC|PerformIndividualActivitiesDuringEVA|
activity|get|1358|1405|projects.EVAPrep.RZ|SelectingGaiters||projects.EVAPrep.GaitersRZ|projects.EVAPrep.S
helfArea
activity|primitive|1397|1414|projects.EVAPrep.CC|CommunicatingReadinessToHelp|
activity|primitive|1405|1417|projects.EVAPrep.RZ|CallingAvailableHelper|
activity|primitive|1417|1667|projects.EVAPrep.RZ|HavingGaitersPutOn|
activity|primitive|1414|1689|projects.EVAPrep.CC|OfferingHelpWithSuitDonning|
... etc.
```

The events entering OWorld are used to generate the animation of the bots in the VW, representing the agent's activities in the Brahms simulation. Animations are done using the Java-script scripting language. A script associated with the OWorld bot-class representing the agent animates each activity event. For example, agent movement is animated with the `actionHumanWalk` function shown in Table 7.

Table 7. Java-script code-fragment for animating an agent's move activity

```
actionHumanWalk.prototype.start=function(args)
{
    ...
    // Calculate the total length and use it to calculate the
    // total time the animation will take.
    this.totaltime = fTime;
    // Notify the sync server of the new position
    sendBrahmsString("interface|setSettable|" +this.owner.getTime()+ "|" +this.owner.m_name+
    "|Location|" + args[4].m_name );
    // Start up the walking animation
    this.owner.contAction.setTime( "walk", 0);
    this.owner.contAction.setWeightPercTarget( "walk", 1, 1 ); return this.totaltime;
}
```

The combination of the Brahms model of agent activities, communications and interactions, and the simulation with event output to Oworld driving the OWorld Java-script execution, generates bot animation of the agent's activities in the virtual world. Figure 20 shows screenshots of the bot animation of agent RZ and CC donning their space suit, corresponding the activities of the agent in the simulation model—the reader should compare the screenshots from Figure 20 with the agent's activity timeline in Figure 19.

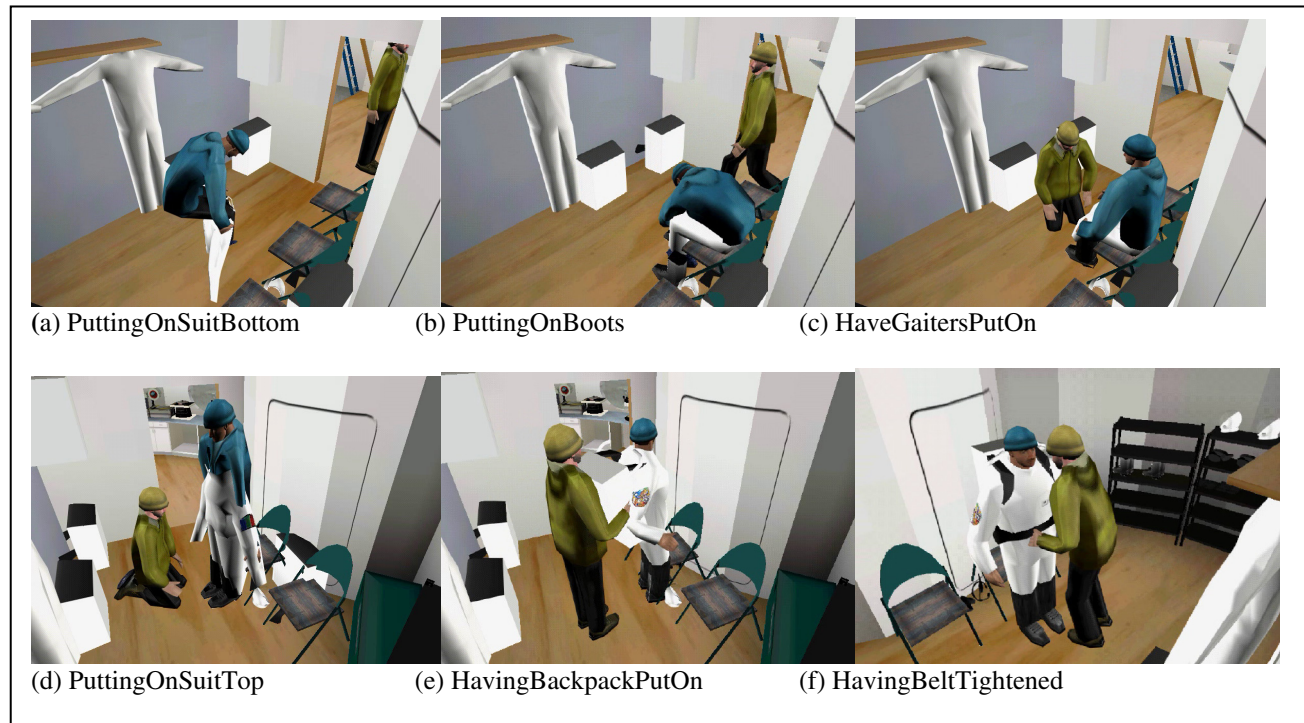


Figure 20. DonningSpacecruit Animation

4. Discussion: Ongoing and Future Work

The discussion thus far explained how the Brahms model simulation “drives” the animation of bots in the virtual world. Although important, this is only part of the story. To enable a true interactive virtual world, one in which the user can interact with the bots through their avatar, there needs to be a two-way communication between the bot’s activity model in Brahms and the virtual world. In other words, the fact that the user can enter the VW means that the world is not stable, but dynamic. Users’ avatars can enter and leave the world, changing it by adding and deleting objects in the world and interacting with bots in a way that was not foreseen by the bot and VW modelers. At the same time, animated bots are the embodiment of the agents in a 3D virtual world. The agent’s embodiment as a 3D-character in the virtual world constrains it in its capability to act, just as people are constraint in their actions in the real world by their environment. For example, just as people cannot walk through walls and other objects, neither can bots in the VW. This embodiment in a 3D virtual world enables the agent to get additional sensory information from it. For example, line of sight is an important constraint on what the agent can observe at any moment in time. In other words, activities of agents are constrained by the bot’s embodiment and sensory-input from the 3D world. The distinction between an agent’s “mind and body” disappears as we implement a tight coupling between the two. This coupling between cognition, motor-skills and sensory input instantiates our notion of situated activities. The 3D virtual world enables and constrains the agent’s activity behavior at the same time. Madsen and Granum describe the three-way interaction, between their VE server, low-level and high-level agent, as high-level sensory information, or percepts, (Madsen & Granum, 2001). We

are currently working on including this capability and in this last section we discuss how we are implementing this in the BrahmsVE environment.

Agent perception of the Virtual World: An integration of the VW and the agent behavior model

As mentioned above, representing and executing agent behavior is only one of the challenges we face when dealing with intelligent behavior in virtual worlds. A more difficult challenge is to integrate agent behavior within an always-changing virtual world. Some of these challenges are:

- How does an agent “see” things (objects and other agents) in the VW?
- How much and how far away can an agent “see” at any given moment?
- How does an agent know what it sees in the VW?
- How can we simulate sound in the VW?
- How does an agent “hear” things in the VW?
- How much and how far away can an agent “hear” at any given moment?
- How does an agent detect collisions with (“feel”) other agents and objects?
- How does an agent know what is in its vicinity?

We deal with these challenges by separating the capabilities needed for agent perception between generation of the phenomena, detection of the phenomena and agent reaction to the detected phenomena. The first two capabilities are implemented in OWorld, while the third capability is implemented with the standard Brahms concepts of detectables, thoughtframes, workframes and activities.

Events that can impact Brahms agent behavior:

1. End move: A Brahms agent moves from one area to another by executing a move activity. Traditionally, the Brahms VM determines how long the move takes, based either on a pre-specified activity time, or the calculation of the path, based on a shortest route algorithm. However, being within a 3D VW the actual path of the move is determined by the possible paths through the VW (doors, objects in the way, et cetera). Therefore, ending Brahms agents’ move activities is determined by OWorld, and the agent has to wait for a signal from OWorld that the bot arrived at its end location. Thus, OWorld will send an end-move event to the Brahms VM, which in turn has to react appropriately. Using this approach, one issue is; What happens if a move cannot be completed and an agent in OWorld is still en route? Brahms has no coordinate system, just areas. The agent might stop moving at a point in the VW that does not correspond with an existing area in the Brahms geography model. Should the Brahms VM send the agent back to the original location? Or more appropriately, should the bot stay at the location of the interrupted move, and “release” control back to the Brahms VM letting the agent model decide what to do next? It is now up to the Brahms modeler to include agent behavior that enables the agent to react appropriately. In this case, the issue is how to deal with the fact that a bot can stop anywhere in the VW and thus at locations in a locale that does not have a corresponding area in the Brahms model. In that case, how does the agent know where it is? One way to solve this problem is for OWorld to dynamically create a new area in the Brahms VM, and place the agent there. Using this approach the Brahms VM will handle the belief creations for the agent and the agent will now know where it is.

2. End activity: Activities have gestures associated with them that cost real time to animate in OWorld. In other words, the time to complete an activity is determined by the time it takes to animate the activity for the bot in the virtual world. The Brahms VM will have to wait for OWorld to tell it that the activity has indeed been completed. One way to deal with this is that the Brahms VM will send OWorld the “desired” end-time for the activity. OWorld will use this time to generate the animation duration and thus be able to determine the “speed” of the animation. As in the end move activity determination, what should happen when the agent in Brahms is interrupted while in its current activity before OWorld completed its animation? Stopping the animation “in the middle” is difficult, given the fact that the animation is generated using a script. At the moment, we are still working on finding an appropriate solution to this issue.

3. Collision events based on auras: “Seeing” and “hearing” of an agent are implemented with the notion of an aura. Figure 21 shows the vision and auditory aura of two bots, represented by the two small black circles. The auditory aura is a 360° circle, allowing the bot to “hear” sounds from any direction. The vision aura is an elliptic area in front of the bot, limiting the bot to only see what is in front of it. Figure 21 shows that both the vision and auditory auras of the two bots overlap.

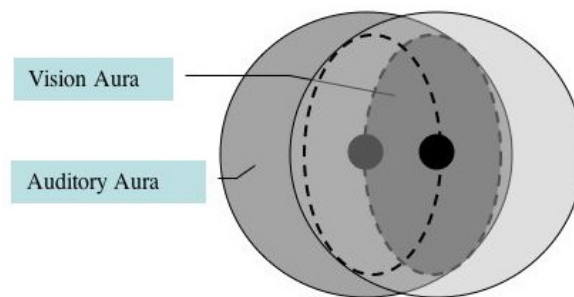


Figure 21. Vision and Auditory Auras of two bots

If the auras of agents overlap a collision event is send to the Brahms VM. Brahms then generates an appropriate fact (“hearing” a sound, and “seeing” an object or another agent in the VW). Such facts can then be detected by the agents using the standard Brahms detectables, allowing the agents to react appropriately. For example, an agent can interrupt a move to initiate a “conversation” with an agent it “sees”. However, this could make the animation choppy, because of the time needed to do the collision processing. Should OWorld wait for Brahms to send a ‘collision processed’ event back to OWorld for each interruption, so that it can then continue with the appropriate animation? If not, a moving bot might be “out of sight” before the collision response takes effect. Should Brahms indicate to OWorld what types of collisions it is interested in for an activity, i.e. whenever a detectable becomes active the VM notifies OWorld of that detectable and through some mapping it can be linked to types of collisions? This optimizes the collision processing. If an agent is never interested in talking to somebody during a certain activity, it does not need to be bothered with collision detection of auras overlapping.

4. Field of vision (the vision aura): All objects that are visible by the agent in OWorld should be “communicated” to the Brahms agent. Brahms’ current auto-detection mechanism should be shut off (normally used when entering an area). OWorld completely drives what an agent sees and therefore implicitly detects. Again, should the Brahms agent notify OWorld of the objects it is

interested in, making object detection explicit in the model and thus more efficient? Doing this would mean that the agent “decides”, explicitly, which objects it will “see”. This does not represent real life, since people do not consciously decide, before entering an area, what objects they will notice.

5. How do we map new event info from OWorld to Brahms? We ensure that whatever is in the VW needs a representation in the Brahms model, so that their identifiers can be used for the mapping? This means every object, agent and user avatar in the VW requires a corresponding representation in the Brahms model. An agent cannot detect (i.e. “see” or “hear”) an object, bot or avatar that does not have a corresponding representation in the Brahms model. This allows a user entering the VW as an avatar to either be detected by the agents or not. The avatar cannot be detected as long as a corresponding agent is not created inside the Brahms VM.

Displaying mental state in the Virtual World

How can the end user understand what a bot is “thinking” about? This problem we had not foreseen until our first experiment with virtual worlds. For training applications it is important that the end-user (i.e. the trainee or trainer) gets a good understanding of what is happening in the training session. Without the ability to immediately understand what the bots are doing and “thinking,” understanding the situation is sometimes very difficult. To alleviate this otherwise inevitable situation, we devised a way for the agents to communicate mental state to the end-user. First off, we allow the agents to speak to the end-user by generating text-to-speech. However, with a lot of bots within the user’s view it can become difficult to know which bot is speaking. To solve this problem, BrahmsVE allows the display of a text-balloon next to the bot that is speaking. This way both text and speech are observed. A more difficult problem is to communicate an activity’s relevant mental state of an agent to the end-user. Figure 22 shows how we communicate to the end-user that the Personal Satellite Assistant (PSA) agent has found the drill it was asked to look for?

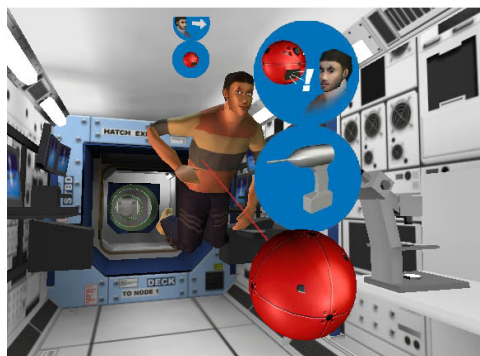






Figure 22. PSA reports the location of the drill, while the astronaut looks at the PSA

We devised a number of status icons that communicate the mental state of an agent to the end-user. Throughout the simulation the mental status of the agents is reported in a text buffer and indicated by convenient status icons projected above the active agents. The visual language for these status icons is described in Table 8 and Table 9.






An activity icon is an indicator of the activity the agent is currently performing. Table 8 shows the activity icons that were developed for the International Space Station (ISS) model, in which an astronaut agent asks a PSA agent to locate a drill in the ISS.

Table 8. Activity Icons (top icon)

PSA & Astronauts 	The Agent is looking at, and tracking, the Target.
	The Agent is moving to a new location.
PSA Only 	The PSA is currently scanning its surroundings for its Target.
Power Lead 	The PSA is currently recharging. The PSA is reporting the tool location.

The target icon represents what object the agent is currently focused on. Table 9 shows the target icons that were developed for the ISS model. Status icons are made visible by a setVisible event from the BrahmsVM to OWorld.

Table 9. Target Icon (bottom icon)

Icons	Target
	Drill, Flashlight or Wrench tools
	PSA is receiving commands (via laptop)
	Agent sees PSA or Astronaut
	PSA is affected by blower fan
	PSA has identified and is utilizing power station

We are currently implementing both the agent virtual world perception capabilities mentioned above, as well as the mental state display capabilities. The BrahmsVE will allow us to develop rich virtual world environments in which bots are not just simple script-driven bots, but can animate complex activity behavior and reasoning and interact with the end-user. We are at the very beginning of enabling the development of just-in-time training applications for space missions. BrahmsVE will allow us to experiment with richer virtual worlds in which end-users can start to interact with complex behavioral bots via their avatars. The Brahms language is a previously tested rich multi-agent language for modeling complex bot-behaviors, and by integrating this language with OWorld we can enable the development of complex virtual worlds on the web. As the next step, after the BrahmsVE environment is sufficiently complete, we will start our research of how just-in-time mission training applications could and should be developed.

Acknowledgements

The work described in this chapter was done from 1999-2002, in close collaboration between the Brahms team at NASA Ames Research Center and DigitalSpace Corporation. DigitalSpace received its funding for this work from a Small Business Technology Transfer (STTR) grant and two Small Business Innovation Research (SBIR) grants. The Brahms team received funding for this work from NASA's Computing, Information and Communications Technology Program (CICT), in the Human-Centered Systems area of the Intelligent Systems (IS) project.

References

- ActiveWorlds. (2004). <http://www.activeworlds.com>. Activeworlds Corporation [2004, August 28].
- Atmosphere. (2004). <http://www.adobe.com/products/atmosphere/main.html>. Adobe [2004, August 28].
- Blumberg, B. (1996). Old Tricks, New Dogs: Ethology and Interactive Creatures. Unpublished PhD thesis, Massachusetts Institute of Technology, Cambridge, MA.
- Brooks, R., A. (1991). Intelligence without representation. *Artificial Intelligence*, 47, 139-159.
- Brooks, R. A. (1997). From Earwigs to Humans. *Robotics and Autonomous Systems*, Vol. 20(2-4), 291-304.
- Clancey, W. J. (2001). Field Science Ethnography: Methods for Systematic Observation on an Expedition. *Field Methods*, 13(3), p. 223-243.
- Clancey, W. J. (2002). Simulating "Mars on Earth"—A Report from FMARS Phase 2. In F. Crossman & R. Zubrin (Eds.), *On to Mars: Colonizing a New World*: Apogee Books.
- Clancey, W. J. (2002). Simulating Activities: Relating Motives, Deliberation, and Attentive Coordination. *Cognitive Systems Research*, 3(3), 471-499.
- Clancey, W. J., Sachs, P., Sierhuis, M., & van Hoof, R. (1998). Brahms: Simulating practice for work systems design. *International Journal on Human-Computer Studies*, 49, 831-865.
- Clancey, W. J., Sierhuis, M., Damer, B., & Brodsky, B. (in press). Cognitive modeling of social behaviors. In R. Sun (Ed.), *Cognition and Multi-Agent Interaction: From Cognitive Modeling to Social Simulation*. New York: Cambridge University Press.
- d'Inverno, M., Luck, M., Georgeff, M., Kinny, D., & Woodbridge, M. (2004). The dMARS Architecture: A specification of the Distributed Multi-Agent Reasoning System. *Autonomous Agents and Multi-Agents Systems*, Kluwer Academic Press, Vol. 9(1/2), 5-53.
- Damer, B. (1997). *Avatars! : Exploring and Building Virtual Worlds on the Internet*. Berkeley, CA: Peachpit Press.
- Huber, M. J. (1999). JAM: A BDI-theoretic mobile agent architecture. Paper presented at the Proceedings of the Third International Conference on Autonomous Agents (Agents'99), Seattle, WA.
- Lave, J. (1988). *Cognition in Practice*. Cambridge, UK: Cambridge University Press.
- Madsen, C. B., & Granum, E. (2001). Aspects of Interactive Autonomy and Perception. In L. Qvortrup (Ed.), *Virtual Interaction: Interaction in Virtual Inhabited 3D Worlds* (pp. 182-208). London: Springer Verlag.
- Madsen, C. B., Pirjanian, P., & Granum, E. (1999). Can finite state automata, numeric mood parameters and reactive behavior come alive? Paper presented at the Proceedings of the Workshop on Behaviour Planning for Life-Like Characters and Avatars, Spain.

- Ogre. (2004). <http://www.ogre3d.org/>. Ogre3d [2004, 11/29/2004].
- Pratt, T. N., & Zelkowitz, M. Z. (1996). Programming Languages: Design and Implementation (3rd. ed.). Englewood Cliffs, NJ.: Prentice Hall.
- Schön, D. A. (1982). The Reflective Practitioner: How Professionals Think in Action: Basic Books.
- Sierhuis, M. (2001). Modeling and Simulating Work Practice; Brahms: A multiagent modeling and simulation language for work system analysis and design. Amsterdam, The Netherlands: University of Amsterdam, SIKS Dissertation Series No. 2001-10.
- Sierhuis, M., Clancey, W. J., & Hoof, R. v. (Submitted). Brahms: A multiagent modeling environment for simulating work practice in organizations. Journal for Simulation Modelling Practice and Theory, Elsevier, The Netherlands, Special issue on Simulating Organisational Processes.
- Suchman, L. A. (1987). Plans and Situated Action: The Problem of Human Machine Communication. Cambridge, MA: Cambridge University Press.
- van Hoof, R., & 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.

