

6. CASE STUDY 1: APOLLO 12 ALSEP-OFFLOAD

In this chapter, I report on the results of the *descriptive modeling* case study (Sierhuis 2000) (Sierhuis et al. 2000a) (Sierhuis et al. 2000c) (Sierhuis et al. 2000d). A descriptive model is an abstraction of an existing or historical system, and preserving the relations between system states—system morphism. I describe the development of a Brahms model and simulation of the ALSEP Offload activity that was part of the ALSEP instrument deployment during the Apollo 12 Lunar mission.

This chapter is divided into a number of sections that could be read or skipped independent of the other sections. Section 6.1 gives a short introductory description of the ALSEP Offload task that the astronauts performed during the Apollo 12 mission. Sections 6.2, 6.3, and 6.4 should be read together. They describe the design of the agent-, object-, and geographical models of the Brahms model. Section 6.5 describes the design of the activity model. Section 6.6 describes the behavioral model. This section describes how the workframes of an agent are executed, and explains the relationship between time, beliefs, workframes, activities and detectables during the simulation of the model. Section 6.7 describes how the communication between the lunar surface astronauts, as well as the lunar surface astronauts and the Capcom agent in mission control is simulated. Included in the communication model is the simulation of the communication time delay between the Moon and the Earth. Section 6.8 describes in more detail how we can model the interaction between people and artifacts. I describe this by explaining how we can model the interaction with a photo camera, while performing the activity of taking a photograph. Section 6.9 describes the process of verifying and validating the Apollo 12 ALSEP Offload model. In this section I describe how the Brahms model and simulation is verified and validated against the available historical Apollo 12 data. Last, section 6.10 describes my conclusions for this case study.

Goals and Objectives

The goal of this experiment was to investigate the use of the Brahms-language in order to *describe an existing work practice*. The challenge I faced in this experiment was to investigate if the theory of modeling work practice, implemented in the Brahms language (Chapter 4), is sufficient to describe the work practice in the chosen domain. The objectives of this first experiment were:

1. Being able to represent the people, things, and places relevant to the domain.
2. Represent the actual behavior of the people, second by second, over time.
3. Show which of the tools and artifacts are used when, and by whom to perform activities.
4. Include the communication between co-located and distributed people, as well as the communication tools used, and the effects of these communication tools on the practice.

The domain I chose for this experiment is the work practice of the Apollo 12 astronauts in the deployment of the Apollo Lunar Surface Experiments Package (ALSEP) on the Moon. The reasons for choosing this domain are the following:

1. The work performed by the astronauts requires unique and highly skilled individuals. The complexity of the work to be described is high enough to argue that if we can model this type of work practice within Brahms, we can model most other work practices as well.
2. The ALSEP deployment process is performed by a relatively small number of people. This has a positive impact on the modeling and simulation effort, in terms of the time it takes to develop the model, as well as the time it takes to simulate the model.
3. The ALSEP deployment work is distributed over the people involved, and is collaborative in nature.

4. There is no work product “flowing” through the work process. This means that this type of work is not easily represented in a workflow model. Being able to model this type of work in Brahms supports the argument for developing Brahms.
5. In order to develop a descriptive model of an existing work practice, we need to have access to a significant amount of data about the actual work. This often means a long observational and/or ethnographical study of the participants. This takes an enormous amount of effort and is a grounded research process in and of itself. However, the Apollo project has been well documented by NASA and numerous institutions, and writers (Compton 1989) (Wilhelms 1993) (Chaikin 1994) (Godwin 1999). Specifically, there is a significant library of video and audiotapes taken during the actual missions (NASA 1972)]. This allows us to develop, verify and validate our models using independent data from the real events.
6. Although a Human Mission to Mars is not an official NASA supported activity at this point in time, more and more researchers in and outside of NASA are informally studying what it would take to have humans go to Mars and do scientific work for an extended period of time. We know very little about how people can or should work on Mars. The only reference point we have about humans working on extra-terrestrial planets is the work that humans did on the Moon during the Apollo project. Developing models of the work practices on the Moon might allow us to extrapolate these models and investigate people working on Mars, before we can physically go there.
7. There is, to certain extent, a real possibility that before we will go to Mars we will first go back to the Moon. The reasons to do this might be of a scientific or a commercial nature. Regardless of the reason to go back to the Moon, a model that describes the existing, but mostly forgotten work practices for deploying instruments on the Moon is self-evident.

6.1 APOLLO 12 AND THE ALSEP OFFLOAD

One of the biggest objectives of the Apollo 12 mission was to deploy the Apollo Lunar Surface Experiments Package (ALSEP). It would be the first time to deploy the ALSEP on the moon. The earlier Apollo 11 mission only deployed a preliminary version, called the EASEP (Early Apollo Surface Experiment Package). The ALSEP consisted of a number of independent scientific instruments that were to be deployed on the moon. The instruments were data collection devices for different scientific experiments about the moon’s internal and external environment. By deploying similar ALSEP instruments over multiple Apollo missions (A12, 14, 15, 16 and 17), the ALSEP deployments created an array of data gathering instruments at different locations on the lunar surface. Table 6-1 shows a list of deployed instruments by mission.

Table 6-1. ALSEP experiments for Apollo missions

	A 12	A 13	A 14	A 15	A 16	A 17
Passive Seismic Experiment (PSE)	X	(X) ³²	X	X	X	
Active Seismic Experiment (ASE)			X		X	
Suprathermal Ion Detector Experiment (SIDE)	X		X	X		
Solar-Wind Spectrometer	X			X		
Lunar Surface Magnetometer (LSM)	X			X	X	
Cold Cathode Gage (CCG)	X	(X)	X	X		
Charged Particle Lunar Environment Experiment (CPLEE)		(X)	X			
Solar Wind Spectrometer (SWS)				X		
Heat Flow Experiment (HFE)		(X)		X	(X)	X
Lunar Ejecta and Meteorites Experiment (LEAM)						X
Lunar Seismic Profiling Experiment (LSPE)						X
Gravimeter, Lunar Surface (LSG)						X
Lunar Atmosphere Composition Experiment (LACE)						X

To deploy the ALSEP on the lunar surface, the astronauts had to accomplish three high-level tasks. First, they had to *offload* the ALSEP from the Lunar Module (LM). Second, they had to *traverse* with the ALSEP packages to the deployment area, away from the LM. Third, they had to *deploy* each ALSEP instrument onto the surface. In this chapter, I discuss the development of a work practice model for the first task, the *ALSEP Offload*.

³² The round brackets mean that these were planned experiments that failed to be deployed properly.

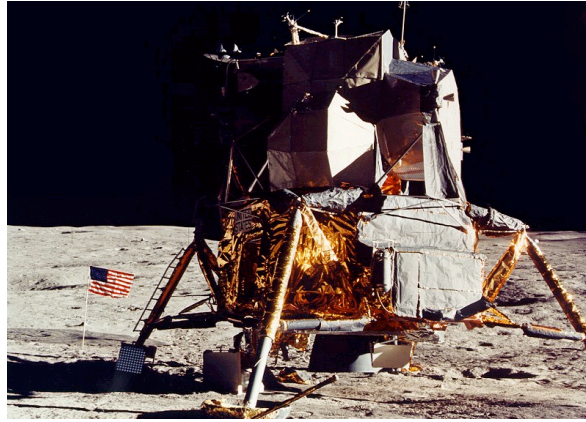


Figure 6-1. SEQ Bay and RTG Cask located on the side of the LM

All the ALSEP instruments and tools, used for deployment, were stored on two sub-pallets (“packages”) in the Scientific Equipment Bay (SEQ Bay) during flight. Figure 6-1 shows the SEQ Bay located on the LM, on the opposite side of the ladder from which the astronauts descended to the lunar surface.

The offload consisted of a number of specified (sub-)activities that were trained extensively and assigned to each of the astronauts. The order in which these tasks were to be performed, and whether the Lunar Module Pilot or the Commander was to perform the task, i.e. the plan, was the same for all five missions. Figure 6-2 shows the plan and start-time for the Apollo 12 ALSEP Offload.

CDR	LMP
1-15 ALSEP OFFLOAD	1-15 ALSEP OFFLOAD
	OPEN SEQ BAY DOOR
REMOVE PKG 1	
REMOVE PKG 2	
STOW BOOMS	DEPLOY HTC
UNSTOW UHT (2)	UNSTOW CASK TOOLS
CONNECT BAR	
TIP PKG 2	
REMOVE SIDE	LOWER CASK
	FUEL RTG
CLOSE DOORS	CONNECT PKGS
(BACK WALL OF SEQ BAY)	
REST/CHECK EMJ	REST/CHECK EMJ
	SUR-47
	Basic Date October 27, 1969
	Changed _____

Figure 6-2. Apollo 12 Surface Checklist 47 for the ALSEP Offload

The order in which the astronauts were to perform their tasks was pre-specified and trained. In other words, offloading the ALSEP was a highly choreographed collaborative activity performed by two astronauts working in parallel.

However, even though this high-level task was planned and choreographed up front, the plan did not include the situational variations, the actual communication and collaborative activities between the astronauts, and the communication between and coordination of activities by the Manned Spaceflight Center (MSC) in Houston. MSC, also known as Mission Control, kept track of where the astronauts were on the plan, solving unplanned problems, and monitoring and communicating life support status for the astronauts. Central in this collaborative activity is the person who played the role of Capsule Communicator (CapCom). The CapCom was the “voice” of Houston and the only person in direct communication with the astronauts. This communication happened through the voice-loop (see section 6.7 on modeling the voice-loop).

The work practice of the ALSEP Offload, or any work practice for that matter, consists of more than the sequence and distribution of tasks. What constitutes the practice of the ALSEP Offload is the way the actual plan is carried out; The situational activities of the collaborators, the way they react to their environment, the way they communicate, what is said, the way they “know” how to do their tasks given the situation. It is *situated action* (Suchman 1987). A choreographed play “executed” during the performance, planned and trained, but always different.

In the next sections, I will describe how the ALSEP Offload work is modeled in a model of work practice. The model is not a model of the problem-solving knowledge of each individual involved in this task. Instead, it is a model of the behavior of the individuals. It describes how the collaboration, coordination, and communication between the three individuals happen, and make this a fluent event. The activities of one individual are like the movements of a musician in a symphony orchestra. The communication between individuals is like the interleaved notes that seem to “tell” each musician what to play next. The artifacts and tools are like the instruments of the musician. The environment of the Moon and Mission Control is like the symphony hall. The Brahms “symphony” that is being played is planned and scored on a piece of paper (i.e. the astronaut’s checklist). The orchestra has trained the piece many times (i.e. the astronaut training on Earth). However, what comes out in the performance is due to their practice, the concert hall (i.e. the Moon), and the way they play together that specific evening (i.e. EVA 1 on Apollo 12).

6.2 THE AGENT MODEL

One of the most relevant design issues for any Brahms model is the design of the agents and the groups they belong to. The *Agent Model* describes to which groups the agents belong and how these groups are related to each other.

Designing an Agent Model is similar to the design of an Object Model in object-oriented design (Rumbaugh et al. 1998). Just as the class-hierarchy in an Object Model, we need to design the group-hierarchy in the Agent Model. As a rule of thumb, we identify the communities of practice of which the agents in the model are members, and abstract them to a common denominator for all agents. All agents are members of this abstract group. The most abstract group is called the *Base Group*. This group exists in the Brahms Base library. It contains all attribute and relation definitions that are needed by default, such as the name of the agent, the group membership relation, and the location of the agent. We specialize this group, until we have identified all the similarities and differences between the agents. It should be noted, again, that groups and agents can be members of multiple groups.

Figure 6-3 shows the Agent Model design. We start with defining our agents. Each agent represents a person in our domain, e.g. Ed Gibson, Pete Conrad, Al Bean, and Dick Gordon. We generalize the community all four agents belong to as the group of *ApolloAstronauts*.

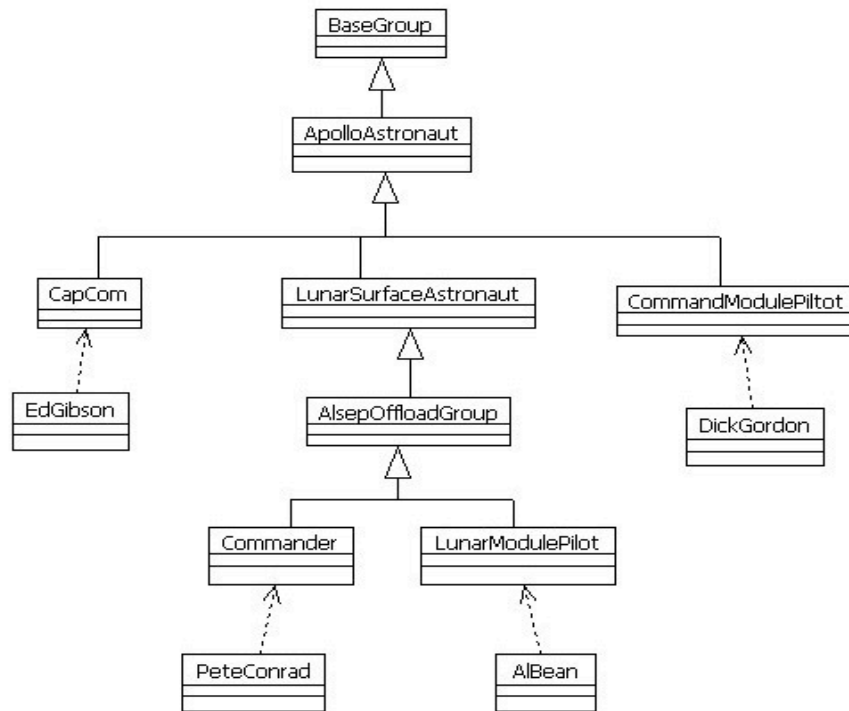


Figure 6-3. Apollo Agent Model design

The *Capsule Communicator* (CapCom) was always an astronaut. In the case of Apollo 12, Ed Gibson was a civilian chosen with the fourth group of astronauts in 1965. The role of the CapCom was to be the only person in Mission Control to talk directly to the astronauts. Dick Gordon, the *CommandModulePilot* (CMP), was a Navy Captain chosen with the third group of astronauts in 1963. The role of the CMP was to fly the Command Module (CM), named “Yankee Clipper”, circling in orbit around the moon while the Lunar Module (LM), named “Intrepid”, was on the moon. Pete Conrad, the Apollo 12 *Commander* (CDR), also a Navy Captain, was chosen with the second group of astronauts in 1962. Last, the *LunarModulePilot* (LMP) Al Bean, who was also in the Navy, was chosen with the third group of astronauts in 1963.

I represent the *role* of each of the astronauts as a group. This way I can represent role specific attributes and activities at the group level. The *AsepOffloadGroup* is a *functional* group in the sense that it does not specify a specific role, but a task of the agent. This group represents all work activities and attributes that have to do with the ALSEP Offload task in one group. This way the group represents the community of agents that can perform the ALSEP Offload task. For Apollo 12, both the CDR and the LMP trained for the ALSEP Offload activities, and both of them could, if necessary, perform the ALSEP Offload task by themselves, and therefore belong to the group ALSEPOffloadGroup. Thus, the Commander and LunarModulePilot groups are members of the group AsepOffloadGroup. Since both the CDR and the LMP were working on the surface there are tasks that both astronauts needed and/or could perform. The ALSEP Offload task was one of them, but there were others as well. All the activities that needed to be performed by all astronauts on the lunar surface are represented in the *LunarSurfaceAstronaut* group. Such activities include taking photographs and changing the cooling of their space suit. In conclusion, we can describe the group hierarchy of Apollo astronauts in three sub-groups, CapCom, CommandModulePilot, and LunarSurfaceAstronaut. The LunarSurfaceAstronaut group has the AsepOffload group as a subgroup, which in turn is subdivided into the subgroups Commander and LunarModulePilot.

Figure 6-4 shows the Brahms source code of the group and agent definitions, as shown in Figure 6-3.

```
// Groups
group BaseGroup { ... }

group ApolloAstronaut memberof BaseGroup { ... }

group CapCom memberof ApolloAstronaut { ... }

group LunarSurfaceAstronaut memberof ApolloAstronaut { ... }

group CommandModulePilot memberof ApolloAstronaut { ... }

group AsepOffloadGroup memberof LunarSurfaceAstronaut { ... }

group Commander memberof AsepOffloadGroup { ... }

group LunarModulePilot memberof AsepOffloadGroup { ... }

// Agents
agent PeteConrad memberof Commander { ... }

agent AlBean memberof LunarModulePilot { ... }

agent DickGordon memberof CommandModulePilot { ... }

agent EdGibson memberof CapCom { ... }
```

Figure 6-4. Brahms source code of the agent model

Figure 6-5 shows the Agent Model in the Brahms Model Builder. The Brahms Model Builder application allows the modeler to create and compile the Brahms model. Figure 6-5 shows the group and agent hierarchy compiled from the source code in Figure 6-4. Each group has a number of “folders” underneath it. Each folder is a different category of model elements for the group. The “Member Groups” folder contains all the subgroups that are a member of the group. The “Member Agents” folder contains all agents that are members of the group. The rectangles around the groups and agents are not part of the GUI, but are added for clarification purposes, so that the reader can easily identify them in the figure. The colors show the different group-levels. At the top, in black, you see the BaseGroup. All the other groups are a subgroup of the BaseGroup group, and are therefore shown in the “Member Group” folder. The yellow rectangles show the four agents, while the other colors show the intermediate groups in the group hierarchy (the colors are only visible in a color reprint).

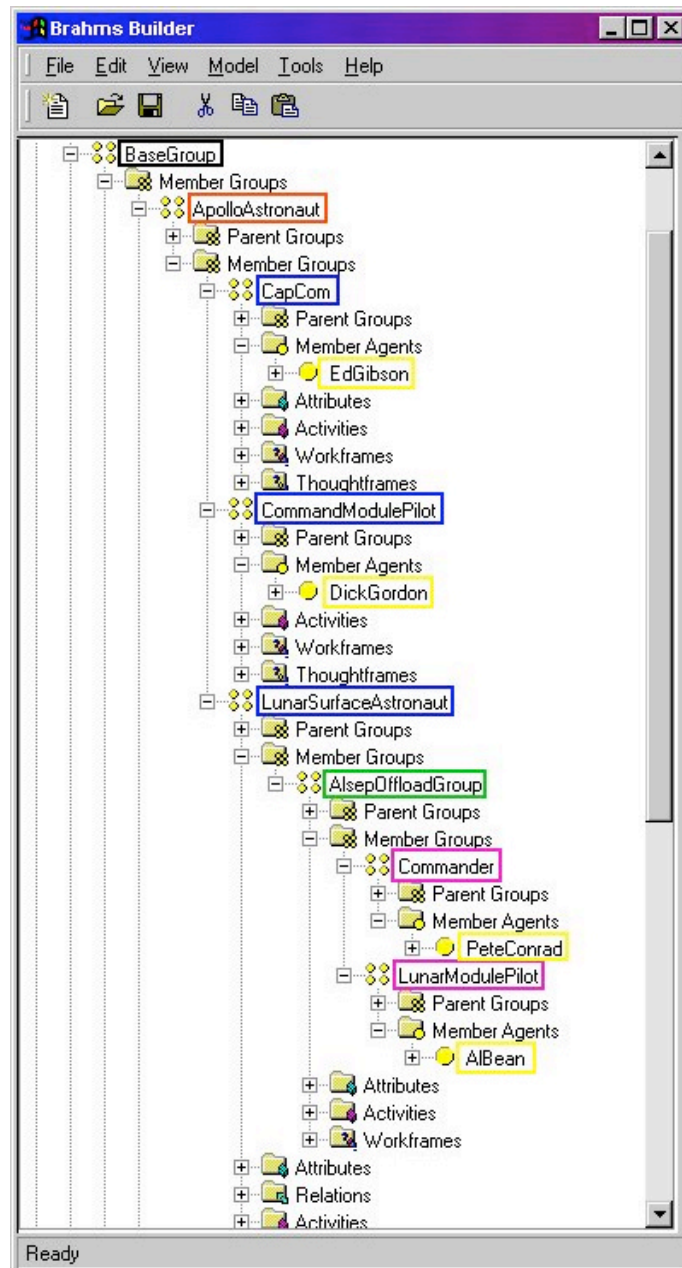


Figure 6-5. Agent Model in the Brahms Model Builder

6.3 THE OBJECT MODEL

After the Agent Model, the next model that needs to be designed is the Object Model. In this model we design the class-hierarchy of all the domain objects. Figure 6-6 shows the Object Model design in UML (Rumbaugh et al. 1998) for the Apollo 12 domain objects and artifacts. As with the Agent Model, the root-class of the class hierarchy is the class *BaseClass*. All other classes and objects inherit from this *BaseClass* class.

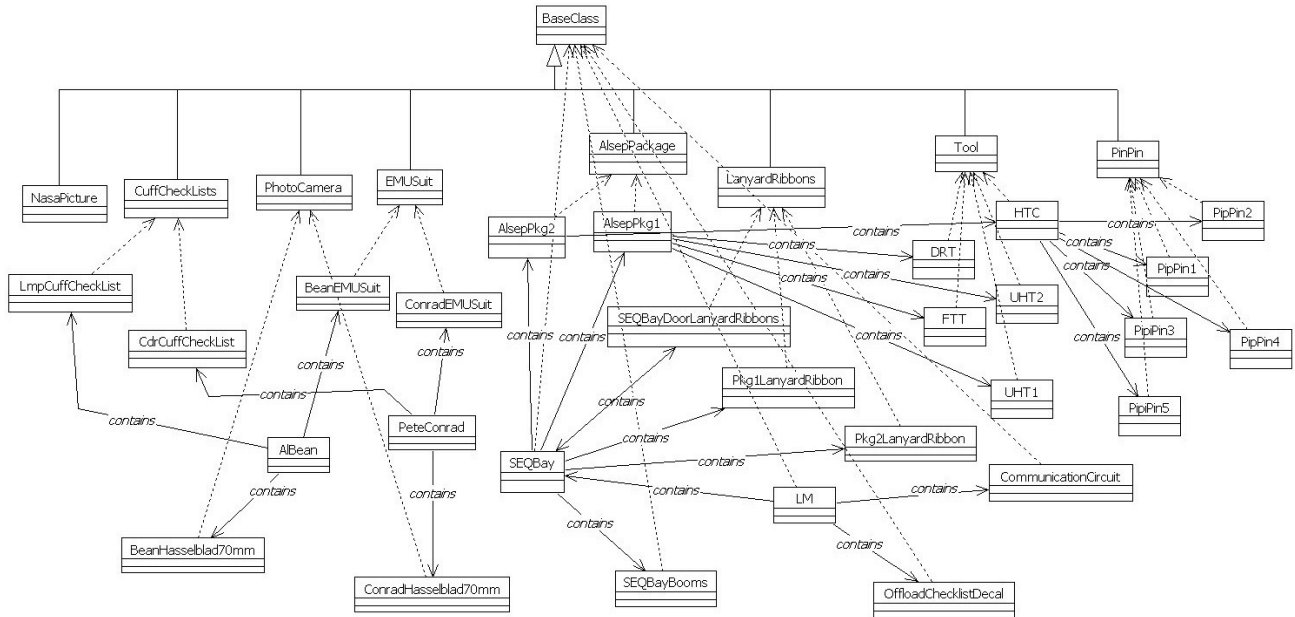


Figure 6-6. Apollo Object Model design

The objects with the dotted arrows pointing up represent the *object instances* of a class. The solid arrows show the *built-in contains relation*. This relation represents objects contained in other objects. This relation has a pre-defined semantic meaning, which is discussed in section 6.4.4.

The objects with the dotted arrows pointing up represent the *object instances* of a class. The solid arrows show the *built-in contains relation*. This relation represents objects contained in other objects. This relation has a pre-defined semantic meaning, which is discussed in section 6.4.4.

Figure 6-7 shows the Brahms model source code for the *LM* and *SEQBay* objects. Both the *LM* and *SEQBay* objects are instances of the class *BaseClass*. Besides representing the corresponding artifacts on the Apollo 12 mission, the source code also specifies the *initial location* of these objects within the Geography Model (see section 6.4). Both objects are located in the *SEQBayArea* area. Furthermore, the objects declare the *attributes* with which we can describe the different aspects of these objects. Although we could describe any number of aspects of an object, such as the color, height, et cetera, we only declare those attributes that are relevant. To model the fact that the astronauts inspect the *LM* and the *SEQ Bay's* exterior appearance after the landing, we declare the attribute *exteriorAppearance* as a type *symbol* attribute. Using this attribute we can represent the state of the exterior of these objects. Both the *LM* and the *SEQBay* objects have an *initial fact* describing the state of their exterior appearance after the landing on the moon as an initial fact for the simulation, e.g.

(the exteriorAppearance of current = SEQBayExteriorLooksGood).

The status of the door of the *SEQBay* is modeled with the *door* attribute of type *symbol* that can have a value of *closed* or *open*. The door is in the initial state (i.e. an initial fact) of being closed, e.g. *(the door of current = closed)*. This represents the door of the *SEQ Bay* being closed at the start of the *ALSEP* offload. Next, we model the objects that are located within the *LM* and *SEQ Bay*. This is represented with the *contains* relation (see Figure 6-6). This relation is declared in the *BaseClass* class, and inherited by the *LM*

and SEQBay objects. The fact that the SEQBay is located on the outside of the LM is represented as an initial fact in the LM object, i.e.

(current contains SEQBay).

The object LM represents the Apollo 12 Lunar Module, named Intrepid. This is the Lunar Module in which the astronauts landed in Surveyor cater. For this model the only relevant object that is part of the LM and that is relevant for the ALSEP Offload is the SEQBay, positioned on the outside of the LM. The SEQBay contains a number of artifacts that are relevant during the ALSEP offload activity. These artifacts are also modeled as Brahms objects in the model (see Figure 6-7 and Figure 6-8).

```
// Apollo 12 objects
object LM instanceof BaseClass {
  display: "Intrepid";
  location: SEQBayArea;
  attributes:
    public symbol exteriorAppearance;
  initial_facts:
    (the exteriorAppearance of current = LmExteriorLooksGood);
    (current contains SEQBay);
}

object SEQBay instanceof BaseClass {
  location: SEQBayArea;
  attributes:
    public symbol door;
    public symbol exteriorAppearance;
  initial_facts:
    (the exteriorAppearance of current = SEQBayExteriorLooksGood);
    (the door of current = closed);
    (current contains AsepPkg1);
    (current contains AsepPkg2);
    (current contains OffloadChecklistDecal);
    (current contains SEQBayDoorLanyardRibbons);
    (current contains Pkg1LanyardRibbons);
    (current contains Pkg2LanyardRibbons);
    (current contains SEQBayBooms);
}
```

Figure 6-7. Apollo 12 LM and SEQ Bay Brahms objects

First, there are the LanyardRibbon objects. These objects are used to open the SEQBay door (SEQBayDoorLanyardRibbons) and lower the ALSEP packages (Pkg1LanyardRibbons and Pkg2LanyardRibbons), respectively. The lanyard ribbons are rope-like artifacts the astronauts pull on to open the door and lower the packages. The two main objects are the ALSEP packages, AsepPkg1 and AsepPkg2. These are the packages the astronauts have to lower from the SEQ Bay and position on to the lunar surface. The SEQ Bay also contains booms (SEQBayBooms). These artifacts are rail extension structures at the top of the SEQ Bay. When the astronaut pulls on the package lanyard ribbons, the ALSEP package comes out attached to the booms. The packages are automatically released from the booms, after which the astronaut slowly lowers them to the lunar surface by releasing the tension on the lanyard ribbons. The last artifact of interest in the SEQ Bay is the OffloadChecklistDecal object. This is the decal that is shown in Figure 6-2, and is a decal that shows the activities and their order for offloading the ALSEP. It is there as a reminder for the astronauts.

Figure 6-8 shows the objects mentioned above, as well as the objects that are contained in each of them. One last interesting note to make is that of the pippin objects. Pippins were used to fasten objects to the ALSEP packages and other artifacts. The HTC (Hand Tool Carrier) object is fastened on AsepPkg2 with five pippins. The fact that the pippins fasten the HTC is modeled by having them be contained in both the AsepPkg2 object and in the HTC object. Removing the HTC from AsepPkg2 means to first “remove” the pippin objects from both the HTC and the AsepPkg2 objects, before the HTC object can be removed from the AsepPkg2 object.

```

// Apollo 12 objects
object SEQBayDoorLanyardRibbons instanceof LanyardRibbons { }

object Pkg1LanyardRibbons instanceof LanyardRibbons { }

object Pkg2LanyardRibbons instanceof LanyardRibbons { }

object AsepPkg1 instanceof AsepPackage {
  initial_facts:
    //carries objects
    (current contains DRT);
    (current contains FTT);
    (current contains UHT1);
    (current contains UHT2);
}

object AsepPkg2 instanceof AsepPackage {
  initial_facts:
    //carries objects
    (current contains PipPin1);
    (current contains PipPin2);
    (current contains PipPin3);
    (current contains PipPin4);
    (current contains PipPin5);
    (current contains HTC);
}

object HTC instanceof Tool {
  initial_facts:
    //carries objects
    (current contains PipPin1);
    (current contains PipPin2);
    (current contains PipPin3);
    (current contains PipPin4);
    (current contains PipPin5);
}

object PipPin1 instanceof PipPin { }
object PipPin2 instanceof PipPin { }
object PipPin3 instanceof PipPin { }
object PipPin4 instanceof PipPin { }
object PipPin5 instanceof PipPin { }

object DRT instanceof Tool { } //Dome Removal Tool

object FTT instanceof Tool { } //Fuel Transfer Tool

object UHT1 instanceof Tool { } //Universal Handling Tool

object UHT2 instanceof Tool { }

object OffloadChecklistDecal instanceof BaseClass { }

```

Figure 6-8. Apollo 12 contained artifacts

Now that the agents and artifacts are represented, the next section describes the geography model in which the agents and artifacts are located during the simulation.

6.4 THE GEOGRAPHY MODEL

In Brahms we model geographical locations using two concepts, *area-definitions* and *areas*. Area-definitions are user-defined types of areas. Areas are instances of area-definitions. Thus an area is an instance of a specific location in the real world that is being modeled. Furthermore, areas can be *part-of* other areas. With this representation scheme we can represent any location at any level of detail.

For the Apollo 12 ALSEP Offload activity, the following locations are important; Earth, the Manned-Space Center (MSC), the Moon, the Apollo 12 landing-site (“Surveyor Crater”), the area where the SEQ Bay is located, the ALSEP deployment area, an area away from the SEQ Bay to place artifacts after offloading,

and last, the lunar orbit and the Command Module (“Yankee Clipper”). Figure 6-9 shows the geography model design.

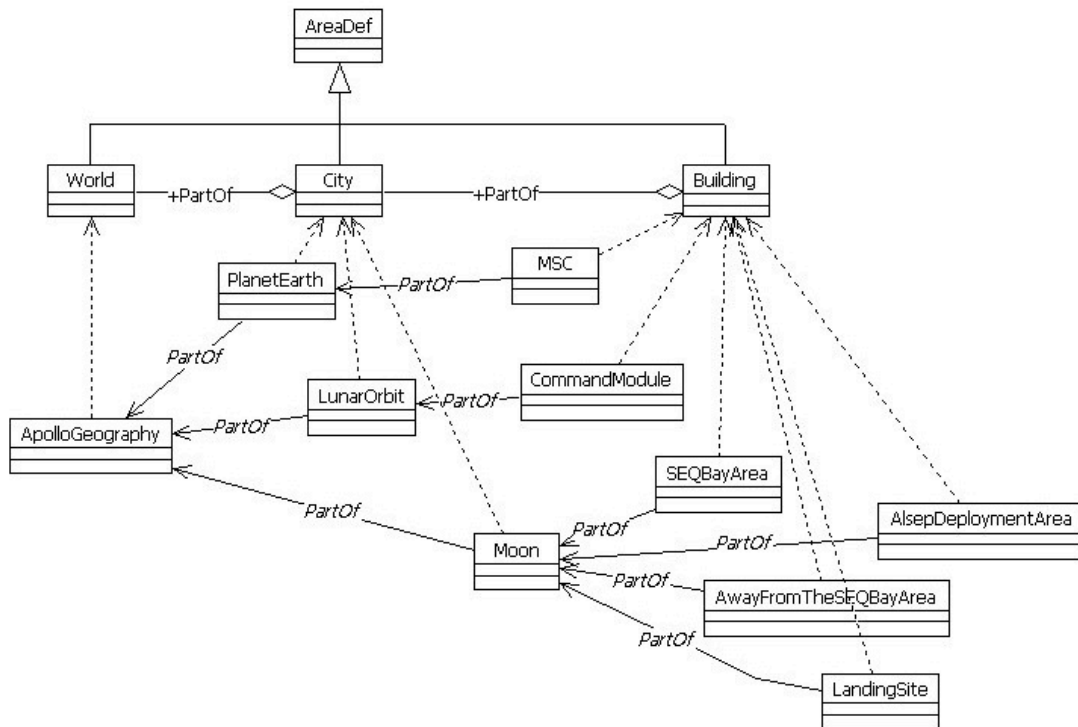


Figure 6-9. Apollo Geography Model design

Figure 6-10 shows the Brahms source code of the area definitions (areadef) and area objects (area). The *area definition* types used to represent the area-instances are World, City and Building.

```

areadef World { }
areadef City { }
areadef Building { }

area ApolloGeography instanceof World { }

// back on Earth!
area PlanetEarth instanceof City partof ApolloGeography { }
area MissionControlCenter instanceof Building partof PlanetEarth { }

// on the Moon!!
area Moon instanceof City partof ApolloGeography { }
area LunarOrbit instanceof City partof ApolloGeography { }
area SEQBAYArea instanceof Building partof Moon { }
area AwayFromTheSEQBAYArea instanceof Building partof Moon { }
area AIslepDeploymentArea instanceof Building partof Moon { }

// Apollo 12 Geography
area CommandModule instanceof Building partof LunarOrbit {
  display: "Yankee Clipper";
}
area LandingSite instanceof Building partof Moon {
  display: "Surveyor Crater";
}

```

Figure 6-10. Geography Model Brahms source code

It does not seem logical to give the area-definitions the names “World”, “City”, and “Building,” and indeed it is not. The reason for this is the limitation of the current Brahms simulation engine³³. The current engine only accepts three types of areas, namely World, City and Building. Also, in the current engine there can only be one world-area. This limitation stems from the fact that our initial designed use of Brahms was for work practice models for the type of work that is performed within buildings, such as the more traditional office-work. This creates an obvious limitation in our representational needs for this extra-terrestrial work domain. First, the work happens in two different worlds, namely on Earth and on the Moon. We therefore would like to create two world-areas in our model. However, because of the current limitation of the engine we need to create the Earth and the Moon as type city-areas, being part of one world. We therefore create one world-area called *ApolloGeography*. This area represents the total “world” for our simulation. An area of type World can contain only areas of type City, therefore the Earth and Moon are areas of type City. Now we have our two planets—Earth and Moon—represented as cities. Secondly, the work on the Moon does not happen within buildings. However, we can only represent areas within a city-area as a type Building area. Thus, the Moon, being of type City, can only have areas of type Building located within it. We therefore represent the locations in which the astronauts perform their work as building-areas. A third “city” is created namely the orbit of the Command Module around the Moon. Since we are not concerned about the location of the Command Module with respect to the Moon and the Earth, we represent the orbit as a city-area within our world. The reason for this is that the Command Module Pilot “lives” within this area. It is therefore easier to locate the Command Module Pilot within his “building” location.

The geographical areas are hierarchically represented as instances of Buildings, which are part of Cities, which in turn are part of the World. This leads to the Compiled Geography Model as represented in Figure 6-11³⁴.

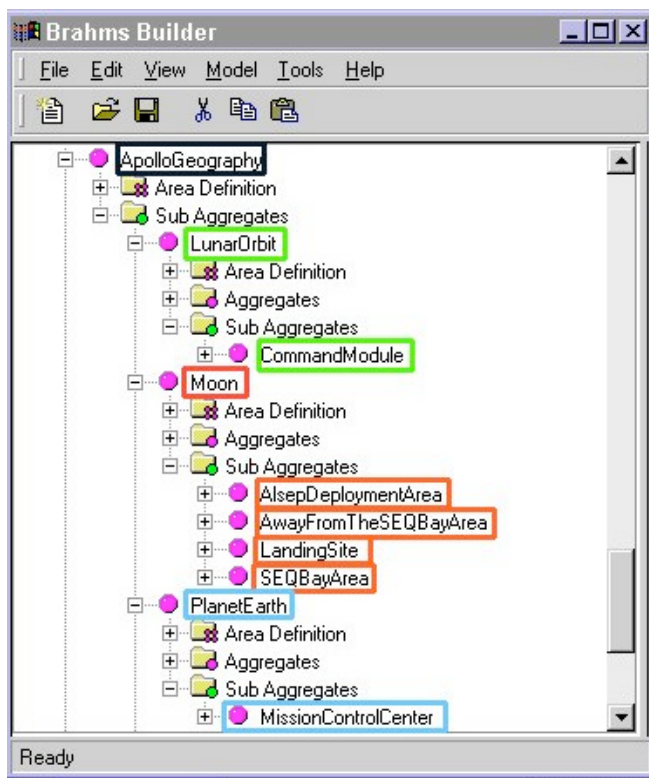


Figure 6-11. Apollo 12 ALSEP compiled Geography Model

³³ We have re-implementing the engine in Java.

³⁴ Figure 6-11 is a part screen capture from the Brahms Builder application.

6.4.1 Initial locations

Each agent and object has an initial location in one of the lowest-level areas, (CommandModule, AwayFromTheSEQBayArea, AlsepDeploymentArea, LandingSite, SEQBayArea, or MissionControlCenter). Initial locations are locations in which an agent or object is placed during the initialization phase of the simulation. This way each agent and object starts out being located in a geographical location (an area). To define an initial location for an agent the modeler uses the *location* attribute at the group or individual agent level. Figure 6-12 shows the initial location for each agent.

```
agent PeteConrad memberof Commander {
    location: LandingSite;
...
}
agent AlBean memberof LunarModulePilot {
    location: LandingSite;
...
}
agent DickGordon memberof CommandModulePilot {
    location: CommandModule;
...
}
agent EdGibson memberof CapCom {
    location: MissionControlCenter;
...
}
```

Figure 6-12. Agent initial location

6.4.2 Movement

Agents and objects can move from one area to another. Moving from one location to another removes the agent from the starting location and moves the agent to the new location. This is accomplished by having the agent perform a move-type activity. The time the activity is active (i.e. the activity duration-time) determines how long it takes the agent to move from location A to location B. Figure 6-13 shows an example of a move-activity.

```
move Moving(Building loc, int pri, int dur) {
    priority: pri;
    max_duration: dur;
    resources: MoveActivity;
    location: loc;
}
```

Figure 6-13. Move activity source code

The move-activity *Moving* starts in the area the agent is located at the moment the move-activity gets activated, and ends at the new area location given by the *loc* parameter. When both agents, PeteConrad and AlBean, perform the activity *Moving(SEQBayArea, 0, 5)* they both move independently from the LandingSite area (Surveyor Crater), their initial location, to the SEQBayArea in 5 seconds, as shown in Figure 6-14.

Movement

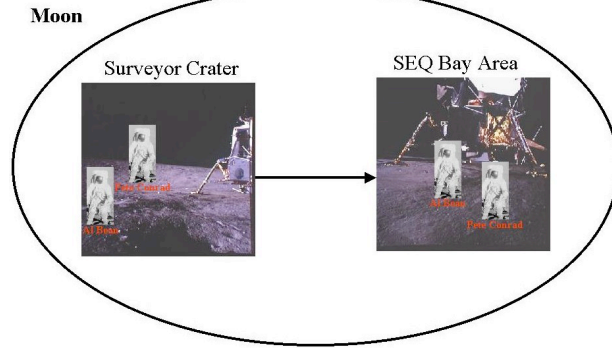


Figure 6-14. Pete Conrad and Al Bean moving to the SEQBayArea

Figure 6-15 gives a from-above view of the LM landing site and the ALSEP Offload Area of activity (the SEQBayArea in the model) from the Apollo 14 Press Kit (NASA 1971).

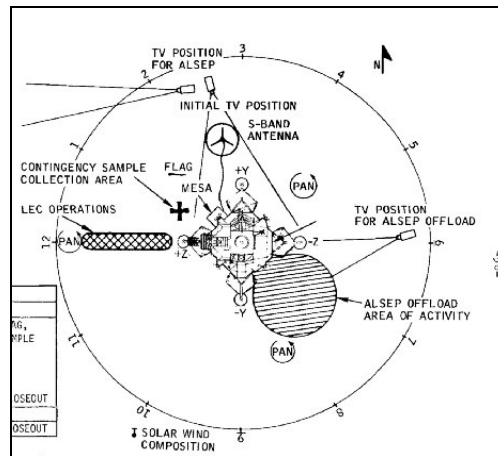


Figure 6-15. Apollo 14 Landing site and ALSEP Offload area of activity

6.4.3 Detecting agents and objects

As both agents arrive at their new location area they will immediately detect facts about the location of other agents and objects that are also in the area they arrive at. The simulation engine automatically creates beliefs for the agent from the facts about other objects and agents that are in the same location. The agents already in that location will get the belief that the agent that arrived is now also in the location. This way, agents will always notice other agents and objects that are in the location the same area.

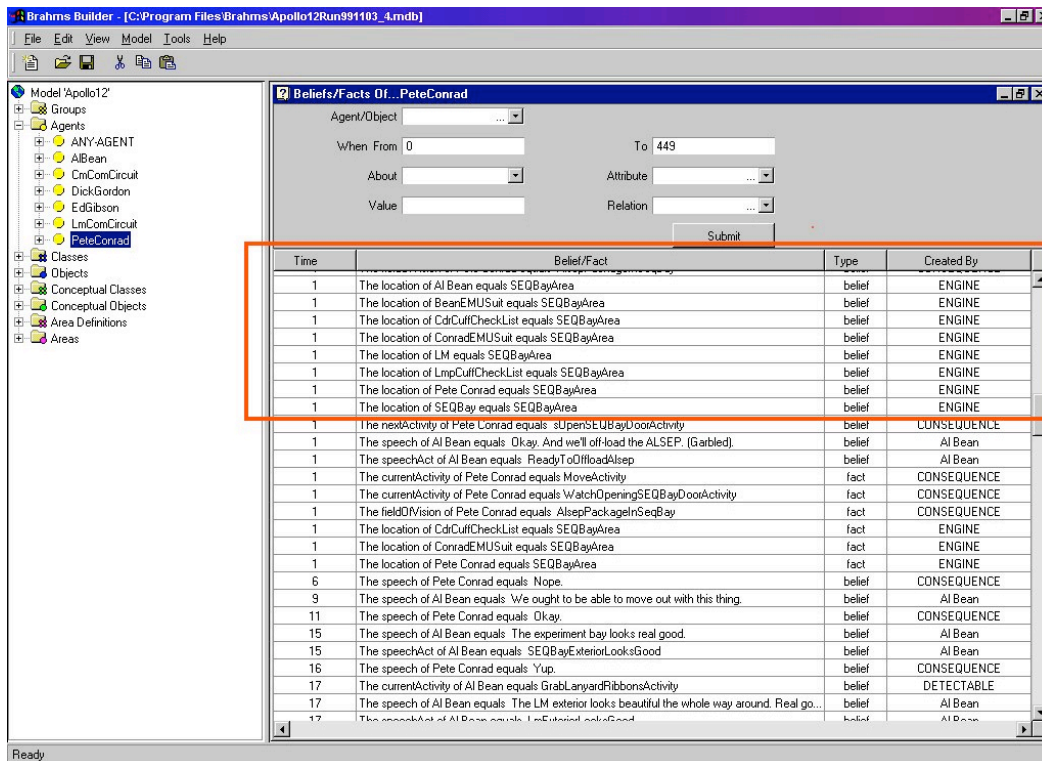


Figure 6-16. Pete Conrad's location beliefs

Figure 6-16 shows the beliefs and facts of the PeteConrad agent in the Brahms Builder application. By opening a simulation history database³⁵ the modeler can investigate what happened during a specific simulation run. Figure 6-16 shows all the beliefs the agent PeteConrad received and the facts it created during the specific simulation run. The columns show the time the agent created the belief or fact, the type (belief/fact) and how it was created (Created by). The (red) rectangle shows the *location* beliefs agent PeteConrad received at time one second into the simulation, created by the simulation engine (Created By: ENGINE). The agent received these beliefs due to the move activity *Moving* that moved the agent from the LandingSite area to the SEQBayArea. As you can see in Figure 6-16, at the moment the PeteConrad agent arrived at the SEQBayArea location it noticed (i.e. received the beliefs) that AI Bean, his EMU space suit and cuff checklist, the LM and the SEQBay, and he himself are all in the SEQBayArea location. Figure 6-16 also shows other beliefs and facts of the PeteConrad agent. The *Created By* column shows who or what created the belief/fact for the agent. *ENGINE* means that the simulation engine created the belief/fact, *CONSEQUENCE* shows that a consequence in a workframe or thoughtframe of the agent created the belief/fact. *DETECTABLE* shows a detectable in a workframe created the belief. The name of an agent or object in the column shows that that agent or object communicated the belief to the agent.

6.4.4 Containment relation

During this case study I ran into a Brahms language limitation. To model the movement of agents and objects correctly I had to add the notion of *containment* to the language (see Figure 6-6). An agent or object can “carry” other agents and objects. Consequently, when an agent or object moves locations all the objects or agents that are “carried” by the moving agent or object should also move to the new location.

This is best explained with a simple example from the domain. As shown by the *contains-relation* in the object model in Figure 6-6, the lunar surface astronauts carry their EMU suit and their cuff checklist. As the astronauts move from location to location we want these carried objects to move with them, without having to specify this moving behavior separately for these objects. Instead, to accomplish this automatically we specified a built-in semantic relation called *contains*.

³⁵ a Microsoft® Access database

When the simulation engine executes a move-activity for an agent (or object) it first checks which objects or agents the moving agent contains. The simulation engine checks this by finding existing *facts* of the form:

Fact: (*[moving agent-or-object]* contains *[contained agent-or-object]*)

For every such fact the *contained agent-or-object* is moved as well. To simulate that an agent or object is no longer containing another object or agent the above containment-fact needs to be negated:

Fact: (*[moving agent-or-object]* contains *[contained agent-or-object]* is false)

Such a negation undoes the containment, and the previously contained agent or object will not be moved in case the agent or object moves.

Following is a small example of the use of the containment relation in the Apollo 12 model. Consider the following scenario; while the LMP agent is offloading an ALSEP package from the SEQ Bay, the CDR agent needs to move the first ALSEP package (AsepPkg1) out of the way, so that the LMP can put the second ALSEP package down. Figure 6-17 shows the source code of the activity.

```
1.   conclude((current contains AsepPkg1), bc:100, fc:100);
2.   MovePkgOutOfTheWay(AsepPkg1, AwayFromTheSEQBayArea, 100, 5);
3.   conclude((current contains AsepPkg1 is false), bc:100, fc:100);
4.   Move(SEQBayArea, 10, 5);
```

Figure 6-17. Moving contained object source code

In step 1, the agent “picks up” the object AsepPkg1. This is modeled by creating a contains-relation fact (fc:100). A belief is also created for the agent (bc:100), because it is obvious that the agent knows he picked up the object. Next, in step 2, he performs a move-activity that moves both him and the contained objects to the AwayFromTheSEQBayArea area. Then, in step 3, the agent “sets down” the AsepPkg1 object. This is modeled by negating the previously created containment fact and belief. Last, step 4 moves the agent, and its current contained objects, back to the SEQBayArea area. Consequently, the AsepPkg1 object remains in the AwayFromTheSEQBayArea area.

Figure 6-18 shows the simulation output of the execution of the MovingPkg1OutOfTheWay workframe described above³⁶. The focus in the picture is on the area within the (red) rectangle. The picture shows the activity time-line of the CDR (agent PeteConrad) and that of the ALSEP package (AsepPkg1) being moved. It can be seen that agent PeteConrad is performing step 2 and step 4 from Figure 6-17. The two rectangle boxes with the text “mv:”³⁷ and “mv: Move” in it, show the duration of the two move activities. It can be seen that after step 2 (rectangle with text “mv:”) the location of both the agent PeteConrad and the object AsepPkg1 has changed from the SEQBayArea area to the AwayFromTheSEQBayArea area³⁸. After agent PeteConrad has performed step 4 (rectangle with text “mv: Move”), only agent PeteConrad (and its contained objects not shown in Figure 6-18) has moved back to the SEQBayArea area. Consequently, due to step 1 and step 3 (the creation and negation of the containment fact), not shown in the figure but executed nonetheless, object AsepPkg1 has been moved out of the way.

³⁶ Figure 6-18 is a screen dump of the AgentViewer tool that shows the result of a simulation. This interface is described in section 6.9.5, as well as the loose insert that is provided.

³⁷ Due to a lack of space in the rectangle, the name of the move activity “MovePkgOutOfTheWay” is not shown.

³⁸ Figure 6-18 only shows the “Away” text in the agent location bar, again, because of space limitation for the complete text string.

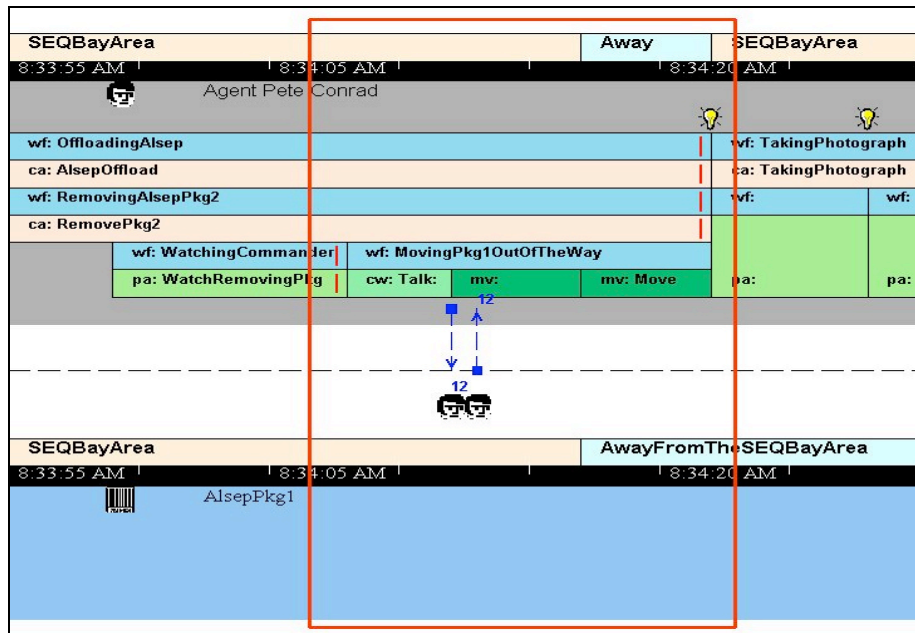


Figure 6-18. Moving contained object simulation

6.5 THE ACTIVITY MODEL

In this section, I describe the ALSEP Offload activities that are performed on the lunar surface, and I describe the Brahms model of the Apollo 12 ALSEP Offload. This model represents a part of the work practice of the Apollo 12 lunar surface astronauts as they performed the ALSEP Offload activity. As shown in section 6.2, there are four people relevant to the Apollo 12 ALSEP Offload; the lunar surface astronauts; Pete Conrad the Commander (CRD), Al Bean the Lunar Module Pilot (LMP), as well as Ed Gibson the Capsule Communicator (CapCom), and Dick Gordon the Command Module Pilot (CMP).

There are three separate areas where these four people are located during the Apollo ALSEP Offload activity (described in section 6.4). The CapCom sits in the Mission Control Center (MCC) located in the Manned Spaceflight Center in Houston, Texas³⁹. His main function is to listen to and communicate directly over the voice-loop with the astronauts. The CDR and LMP are the astronauts on the lunar surface and are located at or near the area of the SEQ Bay, which is located on the backside of the Lunar Module (LM) “Intrepid”. The CMP is orbiting around the moon in the Command Module (CM) “Yankee Clipper.” The main characters in the ALSEP Offload activity are CDR Pete Conrad, and the LMP Al Bean. Their work activities were planned and trained according to the ALSEP Offload checklist (see Figure 6-2).

Figure 6-19 shows that although the sequence of removing ALSEP packages during the mission was planned, there were more activities performed in practice. After the LMP identified that it is time to start the ALSEP Offload, he walks to the SEQ Bay and picks up the SEQ Bay door lanyard from outside of the SEQ Bay, and uses it to pull the SEQ Bay door open. The CDR is watching the LMP opening the door, and is not as is suggested in the plan “doing-nothing”. Once the SEQ Bay door is open, the CDR grabs the lanyard for lowering the first ALSEP package. He walks back from the SEQ Bay with the lanyard in his hand. Meanwhile, the LMP is warm and decides to lower the temperature in his EMU suit (Extra-vehicular Mobile Unit suit, i.e. his space suit). The CDR pulls the lanyard to move the first ALSEP package from the SEQ Bay and lowers it to the ground. While the CDR is performing this activity, the LMP is watching him. When the LMP sees a nice reflection in the CDR’s helmet visor he decides to take a couple of photographs of the CDR. After the CDR has lowered the first ALSEP package to the surface, it is the LMP’s turn to get the second ALSEP package out of the SEQ Bay (compare Figure 6-2 and Figure 6-19). The LMP performs the same activities as the CDR to lower the second ALSEP Package to the lunar surface. While lowering the second ALSEP package, it is the CDR who is watching the LMP. However, when the LMP is lowering the

³⁹ During the Apollo days the NASA center in Houston was called the Manned Spaceflight Center (MSC). Today it is referred to as Johnson Space Center (JSC).

package the CDR notices that the first ALSEP package is in the way, and mentions that he will take the first package and carry it away from the SEQ Bay area. Once he has done that, and is back at the SEQ Bay, he will take three photographs. One photograph of the first ALSEP package as he placed it away from the SEQ Bay, and two more photographs of the LMP lowering the second ALSEP package from the SEQ Bay. During these activities of the two astronauts on the lunar surface the CapCom is listening to the conversation of the astronauts.

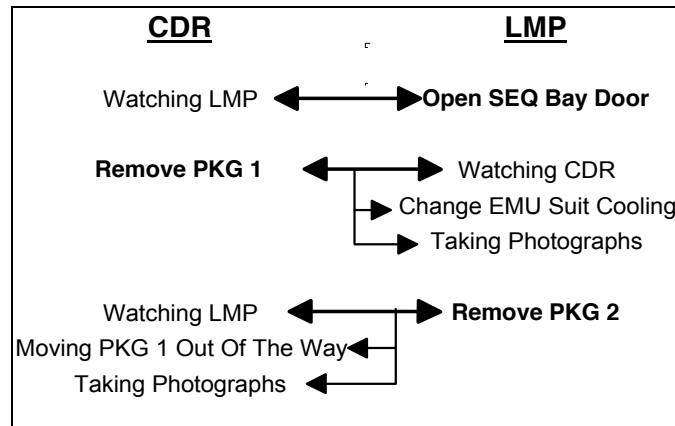


Figure 6-19. Activities in practice

There were activities that the astronauts performed that were not planned or trained. This has to do with the nature of what happens in practice, and is precisely what we want to capture in the model, since it defines how the work actually happened, i.e. the work in practice.

6.5.1 Data sources

I have identified the similarities and differences between the planned activities and the real performed activities during the mission by studying the transcribed data of the communications between the CDR, LMP and CapCom from the Apollo Lunar Surface Journal (Jones 1997). These communication transcriptions have been my major source of data for the Apollo 12 mission. Another source of information has been the Apollo 12 Press Kit (NASA 1969) and the Apollo 12 NASA Mission Reports (Godwin 1999). Unfortunately, there is no video data for the Apollo 12 ALSEP Offload available. This is due to the fact that an unforeseen problem with the TV camera lens and the bright sun on the Moon left the TV camera incapacitated from the beginning of the first EVA, right before the ALSEP Offload. Nevertheless, an accurate account of what happened during the ALSEP Offload can be derived from the second by second verbal communication between the astronauts, in combination with the mission plans. Also, there is video data available from the Apollo 14 ALSEP Offload activities. Although the specifics are somewhat different, the opening of the SEQ Bay door and lowering the ALSEP packages are similar, and the video is therefore a good source for filling in the gaps found in the transcribed communication data. Furthermore, the mission photographs are available as well, and provide some extra visual data.

6.5.2 The Apollo 12 ALSEP offload model

To reiterate, the goal of this modeling experiment is to *describe* the work activities of the lunar surface astronauts of the Apollo 12 mission as they are offloading the two ALSEP packages from the SEQ Bay. The hypothesis is that with Brahm's we can describe (model and simulate) the work practice of these Apollo astronauts.

The data paints an accurate picture of the two lunar surface astronauts communicating. However, the data does not provide an accurate description of the activities of the LMP and CDR. Although the data provides some of the communication from the CapCom and the CMP, there is no detail data of the activities of the CapCom and the CMP. However, I will show that the model proves the hypothesis, by accurately modeling and simulating the work practice of the Apollo 12 *lunar surface astronauts* during the ALSEP Offload, while including, where possible, some of the activities of the CapCom and the CMP.

The model describes the activities listed in Figure 6-2: Open SEQ Bay Door, Remove PKG-1, Remove PKG-2, Deploy Hand Tool Carrier, Unstow Cask Tools, Stow Booms, Unstow Universal Handling Tools, and Close SEQ Bay Door. The activity start- and end-times are computed from the Apollo Lunar Surface Journal (see Table 6-2) (Jones 1997).

Table 6-2. ALSEP Offload activity timetable

Activity	Performer	GET Begin Time	GET End Time	Total Time
1. Open SEQ Bay Door	LMP	116:31:34	116:32:22	0:00:48
2. Remove PKG-1	CDR	116:32:22	116:33:53	0:01:31
3. Remove PKG-2	LMP	116:33:53	116:34:44	0:00:51
4. Deploy Hand Tool Carrier	LMP	116:34:44	116:38:46	0:04:02
5. Unstow Cask Tools	LMP	116:34:44	116:36:25	0:01:41
6. Stow Booms	CDR	116:34:44	116:36:25	0:01:41
7. Unstow UHT	CDR	116:34:44	116:36:25	0:01:41
8. Close SEQ Bay Door	CDR	116:36:25	116:36:49	0:00:24

Figure 6-20 shows the activities Table 6-2 in the Brahms model. The model is viewed within a tree-view. Figure 6-20 shows the *AlsepOffload Group* in the Groups folder of the *Apollo 12 Model*. The parent groups of a group are positioned under the Parent Groups folder. The parent group of the *AlsepOffload* group is the *LunarSurfaceAstronaut* group (see also Figure 6-3), which means that the *AlsepOffload* group inherits all elements from that group. The subgroups of a group are positioned under the Member Groups folder. The subgroups are the *Commander* and *LunarModulePilot* groups, according to the design of the Agent Model (see Figure 6-3). The *PeteConrad* agent is a member of the *Commander* group, while the *AlBean* agent is a member of the *LunarModulePilot* group. Consequently, both the *PeteConrad* and *AlBean* agent inherit all the model elements defined in the *AlsepOffload* group, as well as all model elements inherited by the *AlsepOffload* group from its parent groups. This means that both agents can theoretically perform all the ALSEP offload activities. In reality this was also the case, since both astronauts trained the ALSEP offload activities together on Earth many times before the mission. If, for some reason, one astronaut would not be able to perform his planned activity, the other could perform it for him. This was shown in later missions, when some activities were performed by the astronaut who was not planned to perform the activity (e.g. during the ALSEP Offload on the Apollo 15 mission).

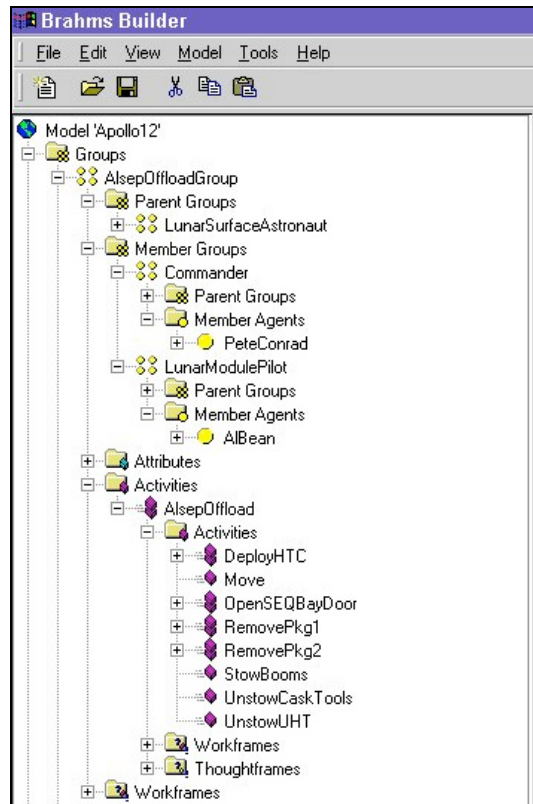


Figure 6-20. The Brahms ALSEP Offload group and activities model

The ALSEP offload activities from Table 6-2 are modeled as sub-activities of the *AlsepOffload composite-activity*, and can be seen in Figure 6-20 under the Activities Folder of the *AlsepOffloadGroup*. In the next sections, I will describe these activities in more detail, and will explain the Brahms model accordingly.

6.5.3 The open SEQ Bay door activity

The ALSEP Offload starts at 116 hours 31 minutes and 34 seconds ground-elapsed time (GET)⁴⁰, with the LMP announcing that they're starting the offload of the ALSEP (see Table 6-2 and Figure 6-21). The next activity is for the LMP to open the SEQ Bay door. In this section, I describe how I modeled this activity in Brahms, based on the available Lunar Surface Journal data (Jones 1997). Figure 6-21 is the transcription from the actual voice loop communication between the CDR and the LMP during the opening of the SEQ Bay door (Jones 1997, Apollo 12 ALSEP Off-load).

116:31:34 Bean: Okay. And we'll off-load the ALSEP. (Garbled).

116:31:39 Conrad: Nope. (Pause)

116:31:42 Bean: We ought to be able to move out with this thing.

116:31:44 Conrad: Okay.

116:31:48 Bean: The experiment bay looks real good.

116:31:49 Conrad: Yup.

116:31:50 Bean: The LM exterior looks beautiful the whole way around. Real good shape. Not a lot that doesn't look the way it did the day we launched it.

⁴⁰ The ground-elapsed time (GET) was the time clock in Houston that was started at the moment of launch.

116:32:02 Conrad: (Possibly pulling a lanyard to open the SEQ bay doors) Light one. (Pause)

116:32:12 Bean: Okay. Here we go, Pete. Ohhhhh, up they go, babes. One ALSEP. (Pause)

[They have raised the doors that cover the cavity where the ALSEP packages are stowed.]

116:32:22 Conrad: There it is.

Figure 6-21. Apollo 12 LSJ: ALSEP Offload transcription (Jones 1997) (with permission)

There are three high-level (sub)activities that one can identify in this *OpenSEQBay* activity. First, there is a communication to MSC in Houston that they are ready to offload the ALSEP. This is the communication starting at 116:31:34. The issue to solve for the modeler is when this activity ends and the next activity begins. From the CDR communication at 116:32:02 we can infer that this is the time that the LMP actually opens the SEQ Bay door by pulling at the SEQ Bay door lanyard ribbons. So, we could start the activity of raising the SEQ Bay door around that time. However, from the video of the Apollo 14 ALSEP Offload it can be shown that before the LMP can pull the lanyard ribbons he has to grab them from the SEQ Bay, walk back until the ribbons are tight, and only then pull the ribbons to raise the SEQ Bay door. These activities have to happen before 116:32:02.

Table 6-3 shows the activities and sub-activities of the *Open SEQ Bay Door* activity for both LMP and CDR, mapped onto the communication transcribed in the Apollo LSJ. The actual names of the activities and sub-activities are more or less arbitrary, and conceptualize the modeler's *interpretation* of the observations of the Apollo 12 communication data and the Apollo 14 video data. However, these data and observations are strong evidence that these are the actual activities that are performed during the *OpenSEQBay* activity.

Table 6-3. Open SEQ Bay door activity

LMP		CDR	
Communicate Ready To Offload		Watching Opening SEQ Bay Door	
Activity	Communication	Communication	Activity
<i>Communicate Open Door</i>	116:31:34 Bean: Okay. And we'll off-load the ALSEP. (Garbled).		<i>Watch Opening SEQ Bay Door</i>
<i>Inspect SEQ Bay</i>		116:31:39 Conrad: Nope. (Pause)	
	116:31:42 Bean: We ought to be able to move out with this thing.		
		116:31:44 Conrad: Okay.	
	116:31:48 Bean: The experiment bay looks real good.		
		116:31:49 Conrad: Yup.	
Raising SEQ Bay Door			
Activity	Communication		
<i>Grab Lanyard Ribbons</i>	116:31:50 Bean: The LM exterior looks beautiful the whole way around. Real good shape. Not a lot that doesn't look the way it did the day we launched it.		
<i>Walk Back To Pull Ribbons Tight</i>			
<i>Pull Lanyard Ribbons</i>		116:32:02 Conrad: (Possibly pulling a lanyard to open the SEQ bay doors) Light one. (Pause)	
	116:32:12 Bean: Okay. Here we go, Pete. Ohhhhh, up they go, babes. One ALSEP. (Pause)		
		116:32:22 Conrad: There it is.	

6.6 THE BEHAVIORAL MODEL

The activities from Table 6-3 are implemented in the Brahms model as the *OpenSEQBayDoor* composite-activity. Figure 6-22 shows this activity, its sub-activities and workframes.

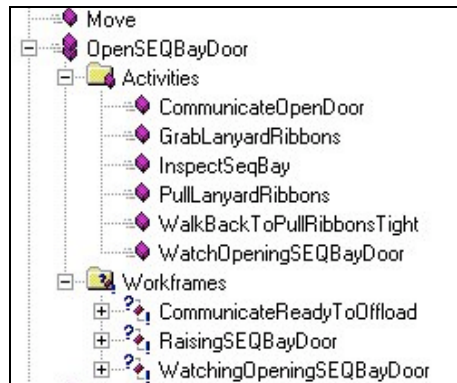


Figure 6-22. The OpenSEQ BayDoor composite-activity, sub-activities, and workframes

Each (sub)activity is “executed” by a workframe, which means that when an agent executes the workframe the activity is performed within the context of that workframe. As the first activity during the ALSEP offload, the CDR and LMP start walking to the area of the SEQ Bay. Walking to the SEQ Bay area to start opening the SEQ Bay door is modeled by the *Move* activity, seen at the top of Figure 6-22. Now that we defined the sub-activities and workframes of the *OpenSeqBayDoor* activity the question is; how do the CDR and LMP agents start this activity during the simulation? Figure 6-23 shows the workframes of the *AlsepOffload* activity that both agents can execute to offload the ALSEP.

The first workframe to fire—the highest-level workframe, but lowest in Figure 6-23—is the *OffloadingAlsep* workframe, which executes the *AlsepOffload* activity. Executing the *AlsepOffload* activity enables all the workframes, shown in Figure 6-22, it to potentially fire for the agent. Each of these workframes will execute lower-level activities, which are subsumed by the higher-level *AlsepOffload* activity.

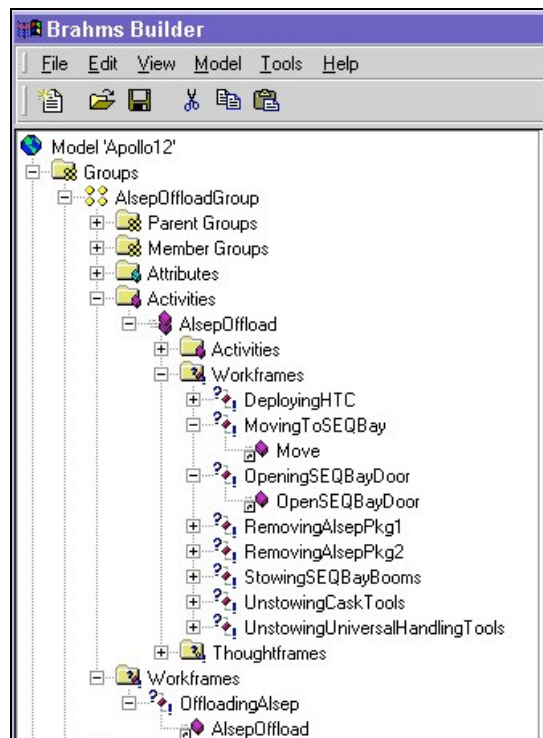


Figure 6-23. The AlsepOffload workframes

We can represent the relationship between workframes executing activities, containing other workframes that execute activities, etc, in a workframe-activity subsumption hierarchy as shown in Figure 6-24.

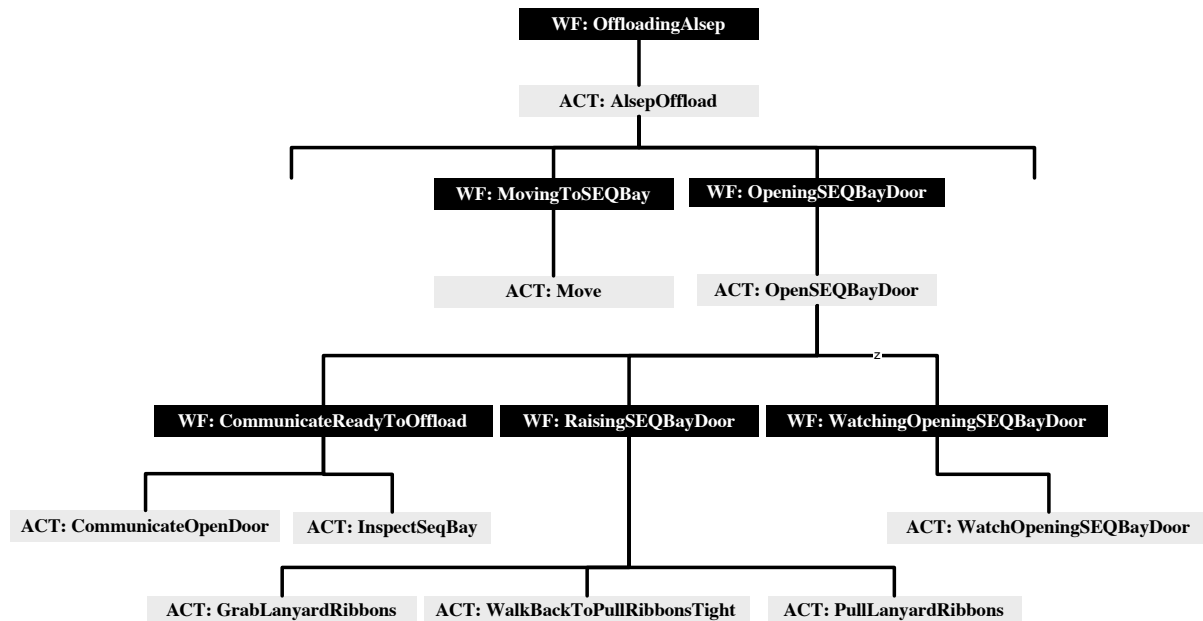


Figure 6-24. AlsepOffload workframe-activity subsumption hierarchy

Only one primitive activity can be active at any given time (i.e. at any clock-tick). This means that the order in which workframes at the same level in the hierarchy fire depends on two things; first, the conditions of the workframe that are to be matched to the beliefs of the agent, and second, the priority of the activities within the workframes.

6.6.1 Representing the work context

Figure 6-25 represents the parallel sequential order of the activities of the CDR and LMP from Table 6-3 and Figure 6-22. However, Figure 6-25 does not represent how the CDR and LMP came to do what they did. The question is not if we can describe the sequential activities of each of the astronauts, but rather, what makes the astronauts do what they do at each moment in time. What influence does the specific Apollo 12 situation have on when and how they do things? What influence do they have on each other's activity? Are they merely executing the OpenSeqBayDoor plan? Or, are they deciding what to do based on their personal knowledge of that plan? If so, we can represent the knowledge of the plan for each individual agent, and be done. This is the traditional knowledge-based systems approach, in which we represent the knowledge "inside people's heads" as production rules. However, what makes Al Bean know that he needs to open the door now, and what makes Pete Conrad know that he has to just watch the commander. What makes them react?

As much as it has to do with their knowledge of the plan for opening the SEQ Bay door, e.g. the steps that they have to go through, it is also a function of the *situation*, i.e. the situation specific context which they are part of. To start the opening door activity they not only need to know what is the right activity to be performing at that moment (according to the plan), but they also need to know that they need to go to the SEQ Bay. To go to the SEQ Bay they need to know where the SEQ Bay is. Once they are at the SEQ Bay, they can see if the door of the SEQ Bay is already open or not. They need to know that the ALSEP packages are located inside the SEQ Bay, and where the lanyard ribbons are located, et cetera. All this has to do with the context of the Apollo 12 mission.

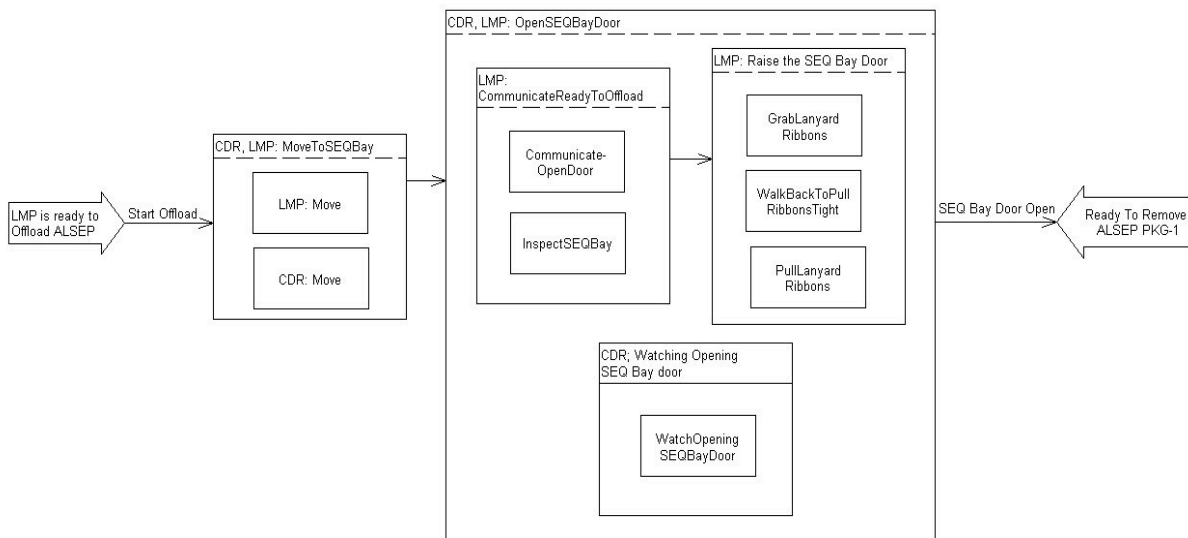


Figure 6-25. Open SEQ Bay door activity sequence model

In Brahms, we model context using three different modeling concepts. First, we model the geographical places at which people perform the work we are interested in (see the section on the Geographical Model, section 6.4). Second, we model all the objects and artifacts that are important in the work. In the case of opening the SEQ Bay door, we model the LM, the SEQ Bay, the ALSEP packages, as well as the lanyard ribbon used to pull open the SEQ Bay door (see the section on the Object Model, section 6.3). Third, we model the state of the world in terms of facts that can be detected by our agent astronauts. For example, when the astronauts walk over to the SEQ Bay, they immediately detect the state of the SEQ Bay door; is it open or closed? They notice the location of all objects in their surroundings. These are the world-facts that trigger the agents to react in certain ways, given the activity they are currently performing. For example, Al Bean would not always open the SEQ Bay door when he comes to the SEQ Bay and notices that the door is closed. He will only do this when he is in the activity of offloading the ALSEP, in particular when his next activity is to open the door. This is where the plan interacts with the situation specific context.

Why does Pete Conrad watch the LMP? What makes him perform that activity? His plan does not say to perform that activity (see Figure 6-2). Rather, this activity is a reaction on the LMP's activity of opening the door during the ALSEP Offload. It is the collaboration between the two astronauts that makes Pete Conrad watch his partner. He sees his partner grabbing the lanyard ribbons. He therefore knows what activity his partner is performing. It is a reaction to the situation and the context, as well as the fact that he is done performing his previous activity.

Figure 6-25 does not represent this context. The influence the context has on the sequence of the activities within Figure 6-25 determines the transitions. The interesting parts of Figure 6-25 are the transitions between the activities. What makes the model go from one state to another? This is what we want to uncover in the understanding of the work practice of the ALSEP offloading.

6.6.2 A narrative description of what happens in practice

Following is my interpretation of what happened during the opening of the SEQ Bay door and why the LMP and CDR do what they do:

When they are ready to offload the ALSEP, they first have to walk over to the SEQ Bay,

Once they arrive at the SEQ Bay, the two astronauts can see the SEQ Bay and can immediately notice that the door of the SEQ Bay is still closed. Of course, they both know that the SEQ Bay contains the two ALSEP packages, and since they are in the activity of offloading the packages they first need to open the door.

This triggers them to start the activity of opening the SEQ Bay door, and since the Apollo 12 ALSEP Offload plan states that the LMP, Al Bean, is to open the SEQ Bay door, he is the one that announces that they will now start with the ALSEP offload.

Since the LMP is the one who is to perform the activity of opening the SEQ Bay door, he is taking the first action and the next activity that is performed is the LMP inspecting the SEQ Bay. However, this is not a planned activity. It seems very likely that this activity is performed based on the astronauts' knowledge that mission control is interested in knowing how the SEQ Bay and the LM have withstood the long travel to the Moon. It is therefore a very important piece of information that the LMP communicates to Mission Control at this point.

Next, the LMP is ready to start raising the door. To do this he needs to grab the lanyard ribbon with which to pull open the door of the SEQ Bay. This means he must know where the lanyard ribbon is to pick-up the ribbon. He then walks back with the ribbon. He needs to tighten the ribbon to have enough leverage to pull the ribbon. Meanwhile, the CDR is standing close by and is watching the LMP, ready to help in case it is needed. Even though the offload plan does not specify any activity for the CDR at this moment, it is logical to infer that the CDR's objective is to closely watch what is happening just in case something happens. This is an activity in and of itself. The CDR would not do anything else even though he could. It seems that the two astronauts always know what high-level activity the other is performing. This means they are always ready to help each other.

After the SEQ Bay door is all the way open, the LMP lets the lanyard ribbon drop to the lunar surface.

I created this narrative, based on my analysis of the available mission data. Based on this short description of what is happening and what makes the two astronauts do what they do, we can list those elements in the context that are most important to include in the model of work practice of the astronauts.

- The SEQ Bay area location near the LM where this all takes place.
- The SEQ Bay and the fact that the SEQ Bay is part of the LM located in the SEQ Bay area location.
- The fact that the exterior of the LM and SEQ Bay are in good condition.
- The two ALSEP packages and the fact that they are located inside the SEQ Bay.
- The door of the SEQ Bay, and that it is closed.
- The lanyard ribbon with which to open the SEQ Bay door.
- The fact that both astronauts detect each other's activity.
- The fact that the LMP needs to carry the lanyard ribbon, and thus must know where this ribbon is located for him to pick it up.
- The fact that after the LMP has completed the activity of opening the door, the SEQ Bay door is open.
- The fact that after the SEQ Bay door is open the lanyard ribbon's location is the lunar surface, because the LMP lets it fall to the surface.
- The fact that both astronauts are noticing all these events and become aware of them, and react to them appropriately.

The challenge is to include these independent context elements into the model. Being able to include these elements in the model is what makes a Brahms model different from the sequential model of Figure 6-25. A sequential model, such as Figure 6-25, can only be executed in the pre-specified order, and does not allow for variations based on context. However, work practice is not the rigid execution of a pre-specified activity sequence. In practice, the sequence of activities depends on the situation. Is the door already open? Are the

packages inside the SEQ Bay or are they already on the ground, et cetera? In the next section, I will describe how these contextual and situational elements are included in the model.

6.6.3 Executing the OffloadingAsep workframe

In this section, I describe the execution of the OffloadingAsep workframe (Figure 6-26). The OffloadingAsep workframe is the highest-level workframe (see Figure 6-23) in the AsepOffload group (see Figure 6-24). Both LunarSurfaceAstronaut agents (LMP and CDR) inherit this workframe, and independently, can execute this workframe in order to start the ALSEP offload.

```

workframe OffloadingAsep {
  repeat: false;
  variables:
    collectall(LmVoiceLoop) vlcoms;
    forone(LunarSurfaceAstronaut) pagt;
    forone(EMUSuit) emusuit;

  detectables:
    detectable DetectPartnerActivity {
      when (whenever)
        detect((the currentActivity of pagt = the currentActivity of pagt));
    }

    detectable DetectCoolingLevel {
      when (whenever)
        detect((the coolingLevel of emusuit = value));
    }

    detectable NoticeAsepPkg1LocationChange {
      when (whenever)
        detect((the objectLocation of AsepPkg1 = anylocation));
    }

    detectable NoticeAsepPkg2LocationChange {
      when (whenever)
        detect((the objectLocation of AsepPkg2 = anylocation));
    }

  when (knownval(the currentConceptualActivity of current = AsepOffload) and
    not(the name of current = the name of vlcoms) and
    knownval(the communicationType of vlcoms = LmVoiceLoop) and
    knownval(the partner of current = pagt) and
    knownval(current contains emusuit))
  do {
    AsepOffload(vlcoms, pagt);
  }
}

```

Figure 6-26. AsepOffloading workframe

6.6.3.1 Variable bindings and preconditions

In order for the agents to execute a workframe (or thoughtframe) all preconditions of the workframe must evaluate to true. The Brahms scheduler will test each precondition and match the precondition to the beliefs of the agent. If there is a belief that matches the precondition, the precondition evaluates to true. The AsepOffloading workframe in Figure 6-26 uses three variables within the preconditions to bind to objects and agents in the model. The first variable, *vlcoms* (i.e. voice-loop communicators), is used to match to the list of all agents (a “collectall” variable) who are members of the group LmVoiceLoop (see the “communicationType” precondition), except for the agent itself (see the “not” precondition). This variable is passed as a parameter to the AsepOffload activity, where it is used to communicate to all the agents who are listening to the voice loop (see section 6.7). The second variable, *pagt*, is used to bind to the partner of the agent in the “partner” precondition. In case the agent executing the workframe (i.e. *current*) is the CDR, *pagt* is bound to the LMP agent (i.e. AIBean). In case the agent executing the workframe is the LMP, *pagt* is bound to the CDR agent (i.e. PeteConrad). This is because the LMP agent, AIBean, has an initial-belief

(the partner of current = PeteConrad),

while the CDR agent, PeteConrad, has an initial-belief

(the partner of current = AlBean).

For each of these two agents the precondition matches, and the *pagt* variable gets bound to the matched agent. The third variable, *emusuit*, is bound to the EMUSuit object of the *current* agent in the “contains” precondition.

All the above-mentioned preconditions are not used to actually guard the workframe from firing. These are all preconditions that are used to bind the variables to the appropriate objects and agents. The only real guard for the workframe is the “currentConceptualActivity” precondition. Not until the agent has the belief that his current (conceptual) activity is to offload the ALSEP, will this workframe fire. This shows that writing workframe preconditions has all the similar precondition control characteristics, such as the ordering, as writing preconditions for production rules in traditional expert systems (Clancey 1983), (Clancey 1988) and (Clancey 1992).

6.6.3.2 Detectables

The workframe in Figure 6-26 contains four detectables that are active as long as the agent is executing the *AlsepOffload* activity within the workframe (this is due to the “whenever” condition in each detectable). All four detectables have a “continue” action part (this is the default action of a detectable). This means that all the detectables are defined in the workframe so that the agent executing the workframe will detect any of the facts that match the detect-conditions, while performing the *AlsepOffload* activity without disrupting the activity itself. This makes it possible for the agent to notice certain facts in the world and react to them, because the facts turn into beliefs for the agent. This allows for the following reactive behavior on the part of the agent:

- The *DetectPartnerActivity* detectable makes sure that during the ALSEP offload activity the CDR and LMP are always aware of the activity their partner is performing. This enables the agent to react to their partner’s activity. There are multiple reasons for modeling that the astronauts on the lunar surface are always aware of this. The first one is that their activities are well choreographed and trained, and the second reason is that it is part of the NASA policy that there is a “buddy system” for EVA work performed by astronauts. This “buddy system” is a safety precaution. This way there is always someone who can help out. This means that the two lunar surface astronauts were very much in tune with what their partner was doing, even if they would be working on their own activity.
- The *DetectCoolingLevel* detectable models the fact that both astronauts are always aware of the cooling-level of their space suit. The fact that the astronauts are wearing their space suit makes this obvious. This detectable allows the agents to react to the cooling-level, and change the level of cooling accordingly.
- The *NoticeAlsepPkg1(2)LocationChance* detectables speak for themselves. Whenever the location of either ALSEP package is changed, the agent will notice this, and can react accordingly. This is used to simulate the fact that when one of the agents lowers the ALSEP package from the SEQ Bay, both agents will become aware of the fact that the ALSEP package has changed its location from the SEQ Bay to the SEQ Bay area (i.e. the lunar surface). This belief is used to start/stop the activities for offloading the actual packages

6.6.3.3 Workframe execution

Following is a description of how Brahms executes the *AlsepOffloading* workframe during simulation of both the AlBean and the PeteConrad agent. The workframe is executed at time $t=0$ for both agents. This means that both agents are executing an instance of the workframe (Workframe Instantiation or WFI) at the same time. I’ll show the WFI for both agents by repeating the workframe from Figure 6-26, but then showing the bindings of the variables in preconditions, consequences, detectables, and activity parameters. Figure 6-27 and Figure 6-28 show the WFI for the agents AlBean and PeteConrad respectively.

```

workframe OffloadingAIslep {
  repeat: false;
  variables:
    collectall(LmVoiceLoop) vcoms; => (PeteConrad, LmComCircuit)
    forone(LunarSurfaceAstronaut) pagt; => PeteConrad
    forone(EMUSuit) emusuit; => BeanEmuSuit

  detectables:
    detectable DetectPartnerActivity {
      when (whenever)
        detect((the currentActivity of PeteConrad = the currentActivity of PeteConrad));
    }

    detectable DetectCoolingLevel {
      when (whenever)
        detect((the coolingLevel of BeanEmuSuit = value));
    }

    detectable NoticeAIslepPkg1LocationChange {
      when (whenever)
        detect((the objectLocation of AIslepPkg1 = anylocation));
    }

    detectable NoticeAIslepPkg2LocationChange {
      when (whenever)
        detect((the objectLocation of AIslepPkg2 = anylocation));
    }

  when (knownval(the currentConceptualActivity of AIBean = AIslepOffload) and
        not(the name of AIBean = the name of (PeteConrad, LmComCircuit)) and
        knownval(the communicationType of (PeteConrad, LmComCircuit) = LmVoiceLoop) and
        knownval(the partner of AIBean = PeteConrad) and
        knownval(AIBean contains BeanEmuSuit))
  do {
    AIslepOffload((PeteConrad, LmComCircuit), PeteConrad);
  }
}

```

Figure 6-27. AIslepOffloading WFI for agent AIBean

```

workframe OffloadingAsep {
  repeat: false;
  variables:
    collectall(LmVoiceLoop) vlcoms; => (AlBean, LmComCircuit)
    forone(LunarSurfaceAstronaut) pagt; => AlBean
    forone(EMUSuit) emusuit; => ConradEmuSuit

  detectables:
    detectable DetectPartnerActivity {
      when (whenever)
      detect((the currentActivity of AlBean = the currentActivity of AlBean));
    }

    detectable DetectCoolingLevel {
      when (whenever)
      detect((the coolingLevel of BeanEmuSuit = value));
    }

    detectable NoticeAsepPkg1LocationChange {
      when (whenever)
      detect((the objectLocation of AsepPkg1 = anylocation));
    }

    detectable NoticeAsepPkg2LocationChange {
      when (whenever)
      detect((the objectLocation of AsepPkg2 = anylocation));
    }

    when (knownval(the currentConceptualActivity of PeteConrad = AsepOffload) and
      not(the name of PeteConrad = the name of (AlBean, LmComCircuit)) and
      knownval(the communicationType of (AlBean, LmComCircuit) = LmVoiceLoop) and
      knownval(the partner of PeteConrad = AlBean) and
      knownval(PeteConrad contains ConradEmuSuit))
    do {
      AsepOffload((AlBean, LmComCircuit), AlBean);
    }
  }

```

Figure 6-28. AsepOffloading WFI for agent PeteConrad

The next two sections describe how the CDR and LMP agents are both performing the AsepOffload activity, and in doing so collaborating in opening the SEQ Bay door.

6.6.4 Performing the AsepOffload activity

After the firing of the OffloadingAsep workframe both agents execute the AsepOffload composite activity (see Figure 6-29). For each agent, the simulation engine changes the agent Activity-Context Tree (ACT) based on the workframes and thoughtframes in the composite activity that execute. An ACT consists of WFI's and the current activity context of the selected workframe⁴¹. In this section, I will show how the simulation engine scheduler schedules the activities for each of the lunar surface astronaut agent. To do this, I first provide the source code of the workframes of the AsepOffload composite activity. Next, I will show the ACT for both the AlBean and the PeteConrad agent for two simulation events (steps). I will show the change in the ACTs as the beliefs of the agents and the world-facts change over time, due to the workframe execution and the agent's reasoning, interaction with other agents/objects and their environment.

⁴¹ Only one workframe instantiation can be fired at any time, which means that there is always only one current activity and therefore only one current activity-context.

```

workframe MovingToSEQBay {
    repeat: false;
    detectables:
        detectable DetectSEQBayDoor {
            when (100)
                detect((the door of SEQBay = value));
        }
    when (not(the agentLocation of current = SEQBayArea))
    do {
        conclude((the currentActivity of current = MoveActivity), bc:100, fc:100);
        Move(SEQBayArea, 5, 1);
        conclude((the nextActivity of current = OpenSEQBayDoorActivity), bc:100, fc:0);
    }
}

workframe OpeningSEQBayDoor {
    repeat: false;
    variables:
        collectall(AlsepPackage) alseppkgs;

    when (knownval(the agentLocation of current = SEQBayArea) and
        knownval(the door of SEQBay = closed) and
        not(the objectLocation of alseppkgs = SEQBayArea))
    do {
        conclude((the currentActivity of current = OpenSEQBayDoorActivity), bc:100, fc:100);
        OpenSEQBayDoor(vlcoms);
        conclude((the door of SEQBay = open), bc:100, fc:100);
    }
}

```

Figure 6-29. Workframes within the composite AlsepOffload activity

1. STEP 1: time $t = 0$

For each Lunar Surface Agent, the scheduler checks the preconditions of all the workframes and thoughtframes in the AlsepOffload activity, based on the agent's current belief set.

AlBean:

Current Belief Set:

$t=0 \Rightarrow$ BELV: The currentConceptualActivity of AlBean = AlsepOffload
 $t=0 \Rightarrow$ BELV: The agentLocation of AlBean = Surveyor Crater
 $t=0 \Rightarrow$ BELV: The agentLocation of PeteConrad = Surveyor Crater
 $t=0 \Rightarrow$ BELV: The agentLocation of DickGordon = Yankee Clipper
 $t=0 \Rightarrow$ BELV: The agentLocation of EdGibson = MissionControlCenter
 $t=0 \Rightarrow$ BELV: The partner of AlBean = PeteConrad
 $t=0 \Rightarrow$ BELV: SEQBay contains AlsepPkg1
 $t=0 \Rightarrow$ BELV: SEQBay contains AlsepPkg2
 $t=0 \Rightarrow$ BELV: SEQBay contains OffloadChecklistDecal
 $t=0 \Rightarrow$ BELV: SEQBay contains Pkg1LanyardRibbons
 $t=0 \Rightarrow$ BELV: SEQBay contains Pkg2LanyardRibbons
 $t=0 \Rightarrow$ BELV: SEQBay contains SEQBayDoorLanyardRibbons
 $t=0 \Rightarrow$ BELV: AlBean contains BeanEMUSuit
 $t=0 \Rightarrow$ BELV: AlBean contains LmpCuffCheckList
 $t=0 \Rightarrow$ BELV: BeanEMUSuit contains BeanHasselblad70mm
 $t=0 \Rightarrow$ BELV: PeteConrad contains ConradEMUSuit
 $t=0 \Rightarrow$ BELV: PeteConrad contains CdrCuffCheckList
 $t=0 \Rightarrow$ BELV: ConradEMUSuit contains ConradHasselblad70mm

Precondition Matching:

workframe MovingToSEQBay:

Prec: not(the agentLocation of current = SEQBayArea)
 TRUE, based on BELV: The agentLocation of AlBean = Surveyor Crater

workframe OpeningSEQBayDoor

Prec: knownval(the agentLocation of current = SEQBayArea)
 FALSE, based on BELV: The agentLocation of AlBean = Surveyor Crater
 Prec: knownval(the door of SEQBay = closed)
 FALSE, based on NO belief about the door of SEQBay

Prec: not(the objectLocation of alseppkgs = SEQBayArea)
 TRUE, based on NO belief about the location of AlosePkg1 and AlosePkg2

Activity-Context Tree:

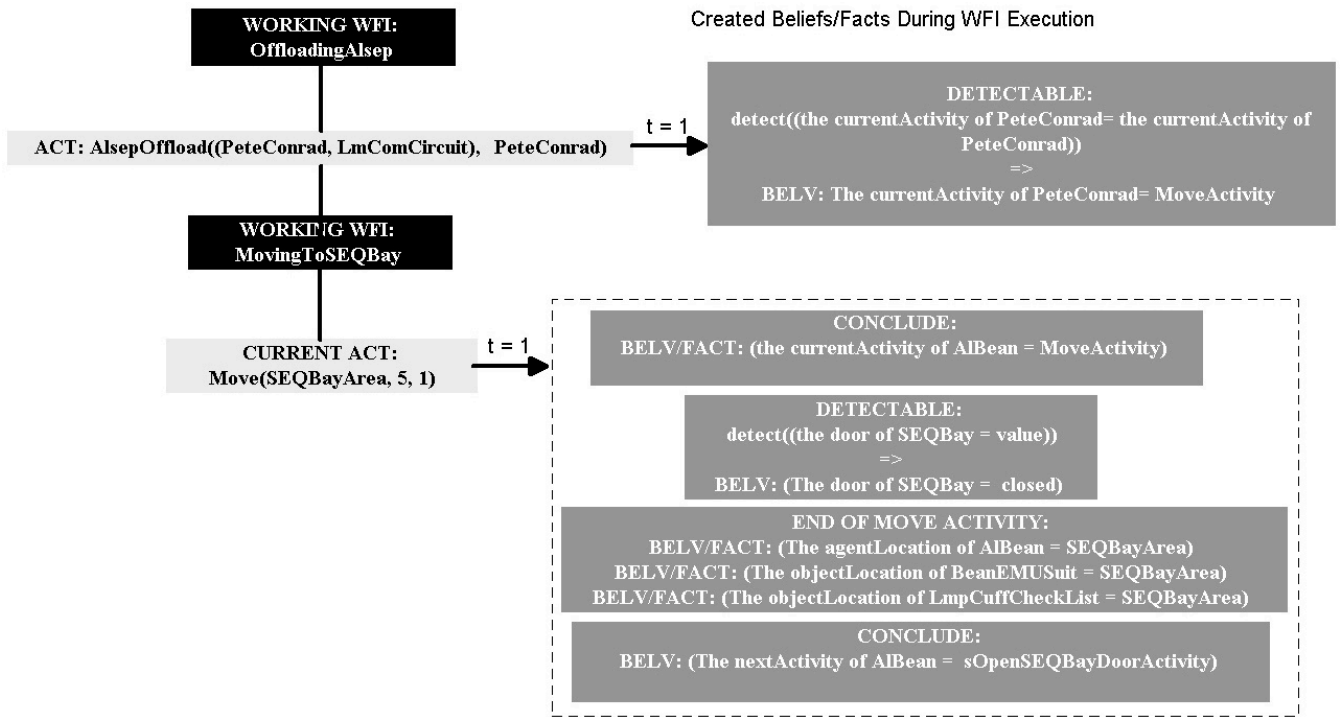


Figure 6-30. AlBean's Step 1 Activity-Context Tree

PeteConrad:

Current Belief set:

- t=0 => BELV: The currentConceptualActivity of PeteConrad = AloseOffload
- t=0 => BELV: The agentLocation of PeteConrad = SurveyorCrater
- t=0 => BELV: The agentLocation of AlBean = SurveyorCrater
- t=0 => BELV: The agentLocation of DickGordon = YankeeClipper
- t=0 => BELV: The agentLocation of EdGibson = MissionControlCenter
- t=0 => BELV: The partner of PeteConrad = AlBean
- t=0 => BELV: SEQBay contains AlosePkg1
- t=0 => BELV: SEQBay contains AlosePkg2
- t=0 => BELV: SEQBay contains OffloadChecklistDecal
- t=0 => BELV: SEQBay contains Pkg1LanyardRibbons
- t=0 => BELV: SEQBay contains Pkg2LanyardRibbons
- t=0 => BELV: SEQBay contains SEQBayDoorLanyardRibbons
- t=0 => BELV: AlBean contains BeanEMUSuit
- t=0 => BELV: AlBean contains LmpCuffCheckList
- t=0 => BELV: BeanEMUSuit contains BeanHasselblad70mm
- t=0 => BELV: PeteConrad contains ConradEMUSuit
- t=0 => BELV: PeteConrad contains CdrCuffCheckList
- t=0 => BELV: ConradEMUSuit contains ConradHasselblad70mm

Precondition Matching:

workframe MovingToSEQBay:

Prec: not(the agentLocation of current = SEQBayArea)
 TRUE, based on BELV: The agentLocation of PeteConrad = SurveyorCrater

workframe OpeningSEQBayDoor

Prec: knownval(the agentLocation of current = SEQBayArea)
 FALSE, based on BELV: The agentLocation of PeteConrad = SurveyorCrater
 Prec: knownval(the door of SEQBay = closed)
 FALSE, based on NO belief about the door of SEQBay
 Prec: not(the objectLocation of alseppkgs = SEQBayArea)

TRUE, based on NO belief about the location of AalsepPkg1 and AalsepPkg2

Activity-Context Tree:

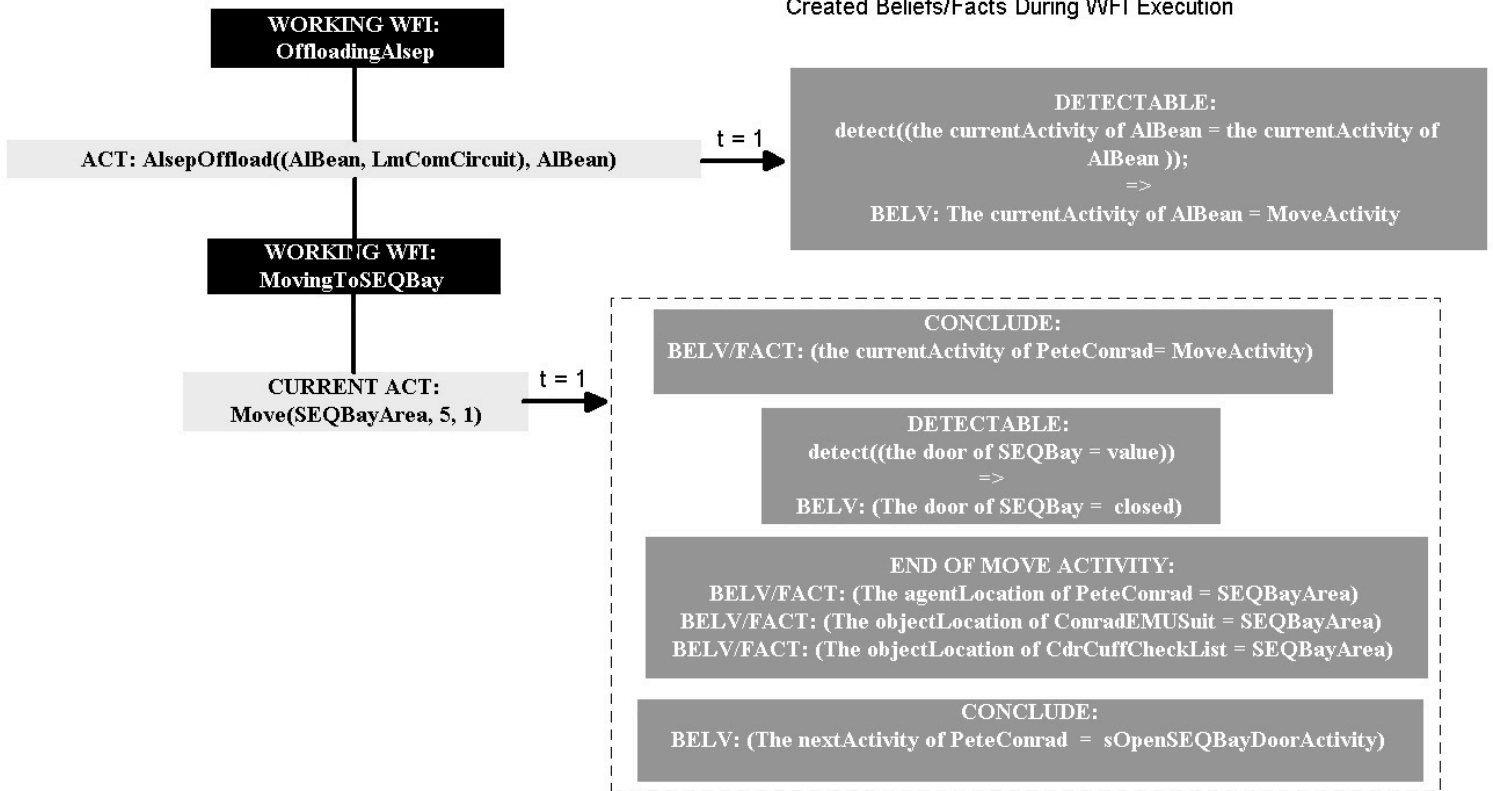


Figure 6-31. PeteConrad's Step 1 Activity-Context Tree

Both agents are executing a move-activity from their current location (i.e. Surveyor Crater) to the SEQBayArea, as can be seen in Figure 6-30 and Figure 6-31. As the agents move to the new location, the objects that they are containing (i.e. cuff checklist and EMU suit) are automatically moved with them to the new location. As the agents arrive in the new location, they detect that the SEQ Bay door is still closed. Also shown in Figure 6-30 and Figure 6-31, the agents automatically notice (i.e. the engine automatically creates the beliefs for the agents) the location of all other objects and agents that are also in the new location; i.e. the location of the other agent, its cuff checklist and EMU Suit, the LM, and the SEQ Bay. Both agents also detect each other's current activity, through the DetectPartnerActivity detectable in the AalsepOffload activity. Lastly, the agents receive a belief about their next activity to open the SEQ Bay door.

When the simulation clock has increased by one, the following (partial) situation exists:

2. STEP 2: time t = 1

For each Lunar Surface Agent, the scheduler checks the preconditions of all the workframes and thoughtframes in the AalsepOffload activity, based on the agent's current belief set.

AlBean:

Current Belief Set:

- t=1 => BELV: The currentActivity of AlBean = OpenSEQBayDoorActivity
- t=1 => BELV: The nextActivity of AlBean = OpenSEQBayDoorActivity
- t=1 => BELV: The door of SEQBay = closed
- t=1 => BELV: The currentActivity of PeteConrad = MoveActivity
- t=1 => BELV: The objectLocation of SEQBay = SEQBayArea
- t=1 => BELV: The objectLocation of LM = SEQBayArea
- t=1 => BELV: The agentLocation of AlBean = SEQBayArea
- t=1 => BELV: The objectLocation of BeanEMUSuit = SEQBayArea
- t=1 => BELV: The objectLocation of LmpCuffCheckList = SEQBayArea

t=1 ⇒ BELV: The agentLocation of PeteConrad = SEQBayArea
 t=1 ⇒ BELV: The objectLocation of ConradEMUSuit = SEQBayArea
 t=1 ⇒ BELV: The objectLocation of CdrCuffCheckList = SEQBayArea
 t=1 ⇒ BELV: The currentActivity of AIBean = MoveActivity
 t=0 ⇒ BELV: The currentConceptualActivity of AIBean = AIslepOffload
 t=0 ⇒ BELV: The agentLocation of DickGordon = YankeeClipper
 t=0 ⇒ BELV: The agentLocation of EdGibson = MissionControlCenter
 t=0 ⇒ BELV: The partner of AIBean = PeteConrad
 t=0 ⇒ BELV: SEQBay contains AIslepPkg1
 t=0 ⇒ BELV: SEQBay contains AIslepPkg2
 t=0 ⇒ BELV: SEQBay contains OffloadChecklistDecal
 t=0 ⇒ BELV: SEQBay contains Pkg1LanyardRibbons
 t=0 ⇒ BELV: SEQBay contains Pkg2LanyardRibbons
 t=0 ⇒ BELV: SEQBay contains SEQBayDoorLanyardRibbons
 t=0 ⇒ BELV: AIBean contains BeanEMUSuit
 t=0 ⇒ BELV: AIBean contains LmpCuffCheckList
 t=0 ⇒ BELV: BeanEMUSuit contains BeanHasselblad70mm
 t=0 ⇒ BELV: PeteConrad contains ConradEMUSuit
 t=0 ⇒ BELV: PeteConrad contains CdrCuffCheckList
 t=0 ⇒ BELV: ConradEMUSuit contains ConradHasselblad70mm

Precondition Matching:

workframe MovingToSEQBay:

Prec: not(the agentLocation of current = SEQBayArea)
 FALSE, based on BELV: The agentLocation of AIBean = SEQBayArea

workframe OpeningSEQBayDoor

Prec: knownval(the agentLocation of current = SEQBayArea)
 TRUE, based on BELV: The agentLocation of AIBean = SEQBayArea
 Prec: knownval(the door of SEQBay = closed)
 TRUE, based on BELV: The door of SEQBay = closed
 Prec: not(the objectLocation of alseppkgs = SEQBayArea)
 TRUE, based on NO belief about the location of AIslepPkg1 and AIslepPkg2

Activity-Context Tree:

As the move-activity in Step 1 (Figure 6-30) ends, in the next clock-tick (t=1) the ACT for agent AIBean changes. The agent is still within the OffloadAIslep activity, because there are still workframes that are in the working-state. The MovingToSEQBay workframe has finished executing, its preconditions are false, and its repeat-variable has the value "false". Therefore, the working WFI is finished and stops. However, the preconditions of the OpeningSEQBayDoor workframe have become true in the same clock-tick (t=1), and a new working WFI for this workframe is created (see Figure 6-32). Next, the composite activity OpenSEQBayDoor in this WFI gets executed. Consequently, the preconditions of all workframes in it are checked. It turns out that for agent AIBean, the preconditions of two of the three workframes evaluate to "true". This means that WFI's are created for both the RaiseSEQBayDoor and CommunicateReadyToOffload workframes, and their state becomes "available". Since there can only be one WFI working at that level in the ACT, the engine solves the conflict by comparing the priorities of the two available WFI's. The priority of a WFI is equal to the priority of the highest activity priority within it. In this case, the priority of the RaiseSEQBayDoor WFI is zero (0) and that of the CommunicateReadyToOffload WFI is ten (10). Consequently, the CommunicateReadyToOffload WFI becomes the working WFI, and its first activity Talk the current activity, i.e. the agent's activity that is being executed.

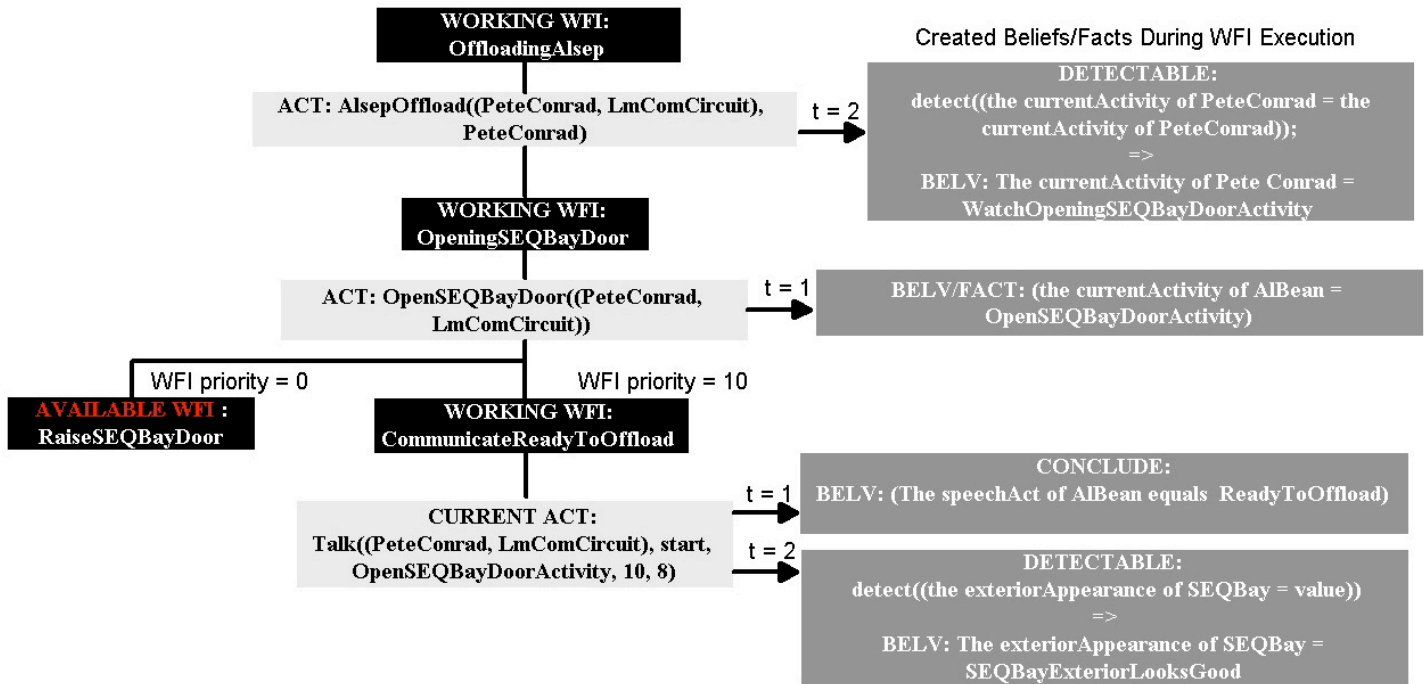


Figure 6-32. AlBean's Step 2 Activity-Context Tree

Also shown in Figure 6-32 are the detectables in the workframes `OffloadingAIslep` and `CommunicateReadyToOffload` firing in step 2 ($t=1$). However, in both cases the beliefs are not created until $t=2$, as a result of detecting the facts at $t=1$. This occurs at $t=2$ and not $t=1$, due to the clock-based simulation engine. The current activity `Talk` starts execution at $t=1$. This means that all the detectables in the working WFI's are checked at $t=1$. The beliefs are not created until the next clock-tick, $t=2$.⁴²

PeteConrad:

Current Belief Set

- $t=1 \Rightarrow$ BELV: The currentActivity of PeteConrad = WatchOpeningSEQBayDoorActivity
- $t=1 \Rightarrow$ BELV: The currentActivity of PeteConrad = OpenSEQBayDoorActivity
- $t=1 \Rightarrow$ BELV: The nextActivity of PeteConrad = OpenSEQBayDoorActivity
- $t=1 \Rightarrow$ BELV: The door of SEQBay = closed
- $t=1 \Rightarrow$ BELV: The currentActivity of AlBean = MoveActivity
- $t=1 \Rightarrow$ BELV: The fieldOfVision of PeteConrad = AIslepPackageInSeqBay
- $t=1 \Rightarrow$ BELV: The objectLocation of SEQBay = SEQBayArea
- $t=1 \Rightarrow$ BELV: The objectLocation of LM = SEQBayArea
- $t=1 \Rightarrow$ BELV: The agentLocation of AlBean = SEQBayArea
- $t=1 \Rightarrow$ BELV: The objectLocation of BeanEMUSuit = SEQBayArea
- $t=1 \Rightarrow$ BELV: The objectLocation of LmpCuffCheckList = SEQBayArea
- $t=1 \Rightarrow$ BELV: The agentLocation of PeteConrad = SEQBayArea
- $t=1 \Rightarrow$ BELV: The objectLocation of ConradEMUSuit = SEQBayArea
- $t=1 \Rightarrow$ BELV: The objectLocation of CdrCuffCheckList = SEQBayArea
- $t=1 \Rightarrow$ BELV: The currentActivity of PeteConrad = MoveActivity
- $t=0 \Rightarrow$ BELV: The currentConceptualActivity of PeteConrad = AIslepOffload
- $t=0 \Rightarrow$ BELV: The agentLocation of DickGordon = YankeeClipper
- $t=0 \Rightarrow$ BELV: The agentLocation of EdGibson = MissionControlCenter
- $t=0 \Rightarrow$ BELV: The partner of PeteConrad = AlBean
- $t=0 \Rightarrow$ BELV: SEQBay contains AIslepPkg1
- $t=0 \Rightarrow$ BELV: SEQBay contains AIslepPkg2
- $t=0 \Rightarrow$ BELV: SEQBay contains OffloadChecklistDecal
- $t=0 \Rightarrow$ BELV: SEQBay contains Pkg1LanyardRibbons
- $t=0 \Rightarrow$ BELV: SEQBay contains Pkg2LanyardRibbons
- $t=0 \Rightarrow$ BELV: SEQBay contains SEQBayDoorLanyardRibbons
- $t=0 \Rightarrow$ BELV: AlBean contains BeanEMUSuit
- $t=0 \Rightarrow$ BELV: AlBean contains LmpCuffCheckList

⁴² In our new Java-based discrete event simulation engine the beliefs will be created at the same clock-tick, i.e. $t=1$.

- t=0 => BELV: BeanEMUSuit contains BeanHasselblad70mm
- t=0 => BELV: PeteConrad contains ConradEMUSuit
- t=0 => BELV: PeteConrad contains CdrCuffCheckList
- t=0 => BELV: ConradEMUSuit contains ConradHasselblad70mm

Precondition Matching:

workframe MovingToSEQBay:

Prec: not((the agentLocation of current = SEQBayArea)
 FALSE, based on BELV: The agentLocation of PeteConrad = SEQBayArea

workframe OpeningSEQBayDoor

Prec: knownval((the agentLocation of current = SEQBayArea)
 TRUE, based on BELV: The agentLocation of PeteConrad = SEQBayArea
 Prec: knownval((the door of SEQBay = closed)
 TRUE, based on BELV: The door of SEQBay = closed
 Prec: not((the objectLocation of alseppkgs = SEQBayArea)
 TRUE, based on NO belief about the location of Aseppkg1 and Aseppkg2

Activity-Context Tree:

As the PeteConrad agent also comes into the SEQBayArea location, he also starts working on the OpenSeqBayDoor activity. Potentially the agent can execute the same workframes as AlBean. However, due to the belief-set of the agent PeteConrad, it will fire the WatchingOpeningSEQBayDoor workframe, which therefore becomes the working WFI.

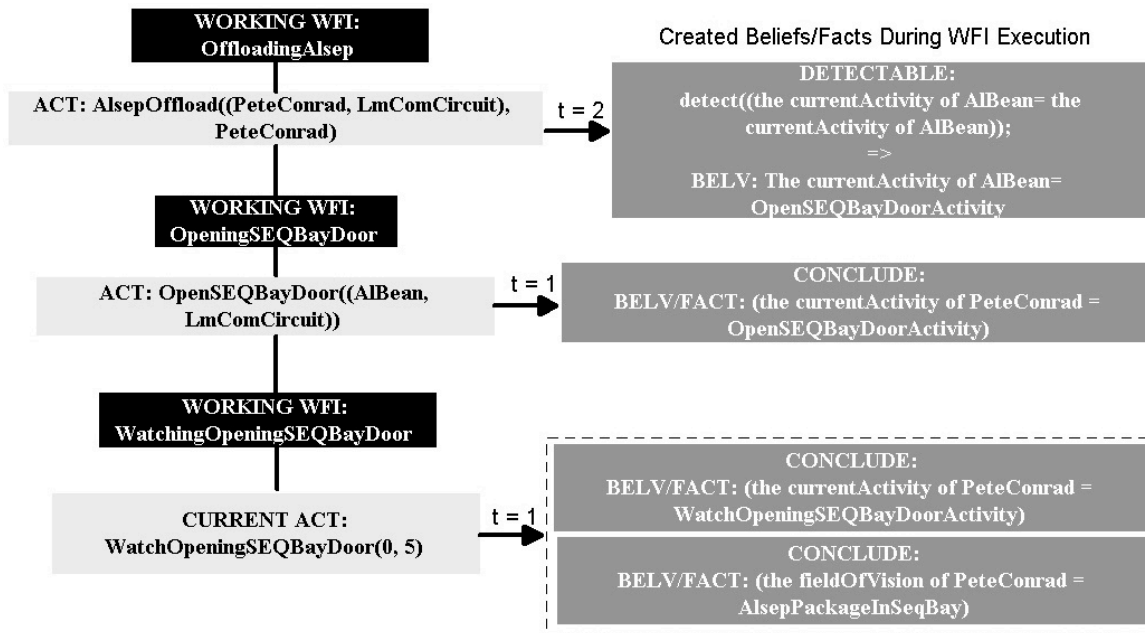


Figure 6-33. PeteConrad's Step 2 Activity-Context Tree

6.6.5 Viewing the simulation results

In this section I show the results of the simulation of the OpenSEQBayDoor activity, as described in the previous sections. Figure 6-34 shows the ACTs of the AseppOffload activity performed by both the AlBean and the PeteConrad agent, as described in section 6.6.4, as well as the communication between the two agents. While performing the AseppOffload composite activity, both agents are within the OpenSEQBayDoor activity. While AlBean is performing the activities within the CommunicateReady and the RaisingSEQBayDoor workframe, the PeteConrad agent is performing the activities within the WatchingOpenSEQBayDoor workframe. The grain-size of the simulation is one second. This means that the simulation engine changes the ACT for every agent and object every second of simulated time. We can therefore say that the simulation is a second by second model of the work practice of the lunar surface astronauts. Figure 6-34 also shows the location the agent was in when performing the activity. As an

overlay, the dotted arrows show the communication of beliefs between agents AlBean and PeteConrad. The direction of the arrows show the direction in which the beliefs are being communicated, while the little square box at the start of the arrow shows the agent that is performing the communication.

Figure 6-34 is a screen shot from the AgentViewer application⁴³. The AgentViewer application takes as input a Brahms Simulation History database⁴⁴. This history database contains the historical situation-specific model data of a particular simulation run. The AgentViewer application creates a graphical representation of the activity of agents and objects during a simulation.

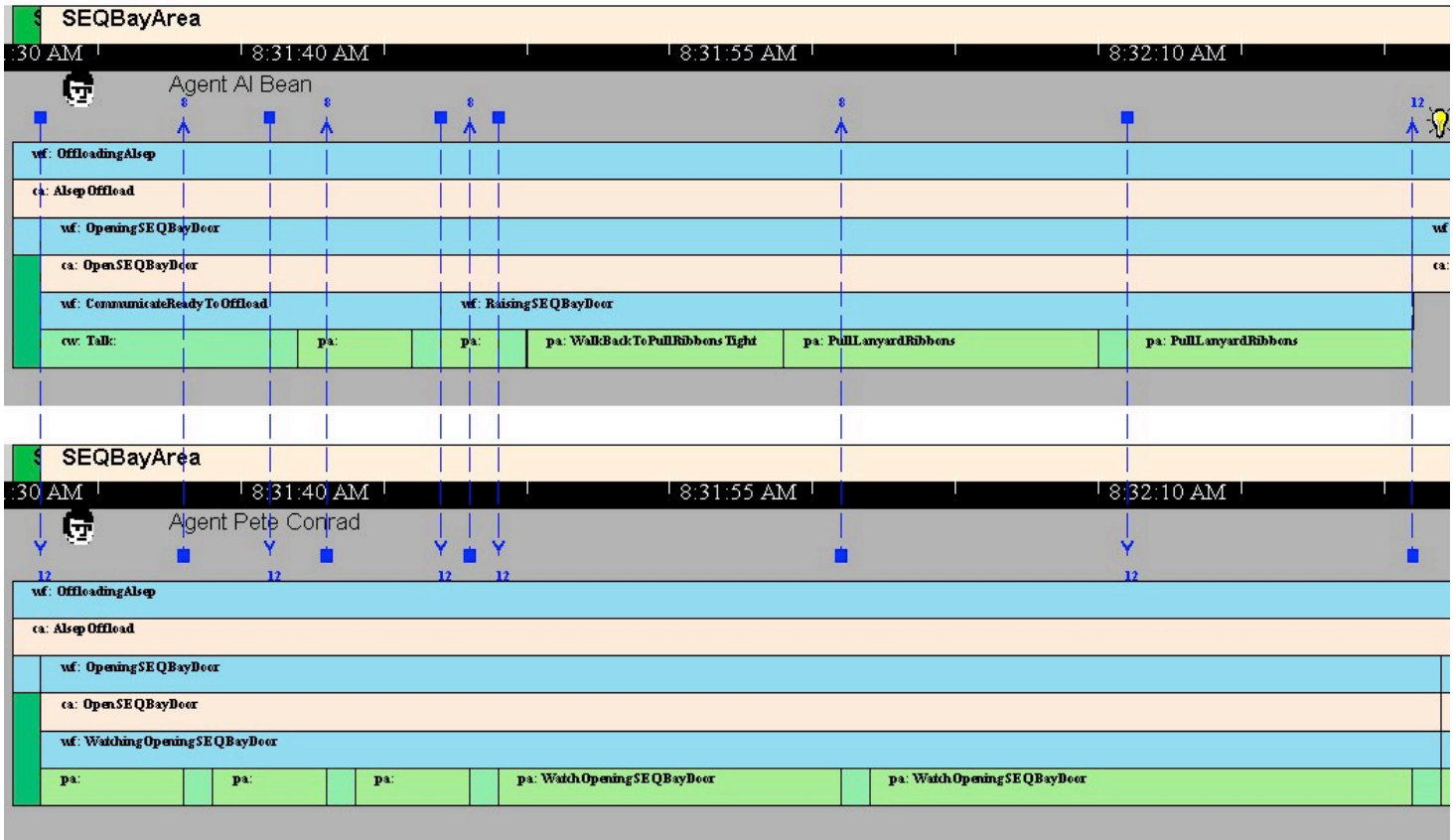


Figure 6-34. AlsepOffload activity agent timeline

Grouping a number of important data about the activity of the agent during the simulation into an agent workframe-activity hierarchy shows the ACT of an agent or object, at any time during the simulation. Each agent's ACT consists of a number of "bars." Each bar is an object that can be manipulated in the AgentViewer.

At the top of each agent's ACT there is the *location bar*. The location bar shows the movement of the agent throughout its activities. When an agent changes location the color of the location bar changes⁴⁵. In Figure 6-34 both agents start in the same initial location This is the Apollo 12 LandingSite area (Surveyor Crater). You can see that the next location both agents are in is the SEQBayArea location.

The next bar in the agent's ACT is the *time-line bar*. This bar shows the simulation time. Figure 6-34 shows that the AlsepOffload activity starts just after 8:31:30 AM (in fact the simulation clock starts at time 8:31:32 AM). Each thin white line in the time-line bar shows a 5-second interval. Consequently, Figure 6-34 shows an activity interval of about 50 seconds (from 8:31:30 AM until about 8:32:23 AM).

⁴³ The AgentViewer application is a stand-alone Visual Basic application we developed for viewing the results of a simulation.

⁴⁴ The history database is a complex relational database containing the simulation data preserving their relationships.

⁴⁵ An agent does not effectively change its location until the simulation engine has finished a move activity and consequently positions the agent into the new location. The agent's location during the move activity stays unchanged, even though the agent is moving, and should thus not be in any location. Brahms is not modeling the movement of agents during the execution of a move-activity.

The third bar is the agent's *name bar*, with the name of the agent and an agent icon). This bar shows which agent or object⁴⁶ is being displayed

The fourth and final bar is the *workframe-activity bar*. This is the bar that shows the execution of the workframes, activities, and thoughtframes for the agent. Workframes are represented as blue bars that start with the letters "wf", for workframe, and the name of the workframe (if it fits within the graphics block). Underneath a workframe bar there can either be a flesh-colored bar, or a green-colored bar. A flesh-colored bar represents a composite activity, and starts with the letters "ca", for composite activity. A green-colored bar is a primitive-, move-, communicate-, or create-object activity. These are always the lowest level activities. Each type of primitive activity is indicated by a different shade of green. Other than a color indication; a primitive activity is indicated by the letters "pa", for primitive activity; a move activity by the letters "mv", for move; a communicate activity by the letters "cw", for communicating-with; a create-object activity by the letters "co", for create-object. When the size of the graphics block is large enough to contain the name of the activity it is shown as well. If not, the name is shown when the user moves the mouse over the activity or workframe box.

6.7 VOICE-LOOP COMMUNICATION

One of the most important aspects of work practice is the way people communicate. The communication to and from the Apollo Lunar Surface was made possible by the Extra-Vehicular Communication System (EVCS). The EVCS was a communication relay system that communicated voice from the astronauts via their EMU suits to the LM and via the LM, using a S-Band antenna, to mission control. The voice of the CapCom was communicated back to the LM and the astronauts via the same system (see Figure 6-35). This way the lunar surface astronauts and CapCom were in constant two-way communication. The CMP and CapCom had a similar communication system via the CM. The two lunar surface astronauts were operating their communication system in dual mode, which meant that they were always able to hear each other. However, the CMP was not in direct communication with the lunar surface astronauts, and was therefore not always able to hear them.

In this section I describe how the EVCS, or as I have named it, the voice-loop communication has been modeled in Brahms.

6.7.1 Communication delay

Conversational overlaps are a normal part of human dialog, and humans are pretty well apt to deal with this phenomenon. However, the communication delay from Earth to the Moon is significantly larger than the face-to-face or phone communication on Earth. The one-way delay, to Earth and to the Moon, is one and a quarter (1.25) second. This means a minimum of two and a half (2.5) seconds round-trip communication delay. If one of the astronauts made an utterance, the CapCom would hear the utterance one-and-a-quarter second later. If the CapCom would respond immediately, the astronauts would not hear this response until one and a quarter second later, which means a total of, at minimum, two and a half seconds.

⁴⁶ Objects have a different object icon.

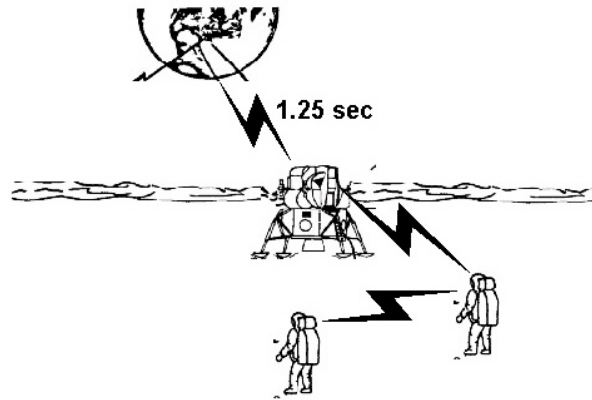


Figure 6-35. Extra-Vehicular Communication System

During the Apollo missions the communication delay sometimes lead to problematic communication patterns, as is shown in the example from Apollo 17 from the section "Journal Preparation and Structure" in the Apollo LSJ (Jones 1997).

In the following example, we imagine CapCom Bob Parker giving Gene Cernan instructions on parking the Rover. Before Cernan hears Parker, he starts to make a comment about where he is parked. He then stops talking, listens to Parker (who doesn't stop talking), responds, and then continues with his comment.

Parker: Gene, just a reminder that we want a Rover (garbled)...

Cernan: Bob, we've stopped next to...(Hears Parker)

Parker: ...(heading) of 045; and, when you get out, we'll need readouts.

Cernan: (Responding to Parker) Okay, Bob. We've parked next to one of the fresh craters that shows up on the map.

Generally, when someone's utterance ends with ellipses and his next utterance begins with ellipses, the reader should infer that the speaker kept talking under the overlapping remark. When someone's utterance ends with ellipses but his next utterance does not begin with ellipses, the reader should infer either a break in thought or a pause to listen. Unintelligible dialog is indicated by the editorial comment "garbled". Unintelligible dialog is often associated with overlapping conversations and in this illustration, on the continuation of Parker's utterance I have indicated the likely missing word. "

Although in this example Eric Jones is referring to the transcription as done for the Apollo LSJ, the fact of the matter is that if one wants to model the communication utterances of the astronauts and their impact on work practice, we have to model the one and a quarter delay for each communication event.

6.7.2 Modeling the communication to Earth

The voice-loop in context of the Apollo missions is the inter-communication system between the astronauts on the Moon and the CapCom at MSC in Houston.

There is a significant difference between voice-loop communication and face-to-face (f-2-f) communication. First, and foremost, f-2-f communication is bounded to geographical location of the agents. This means that the agents have to be in the same location to be able to engage in a f-2-f communication activity. This is referred to as Same Time/Same Place (STSP) communication (Chapter 3.2.4.4). In voice-loop communication there is no restriction on the geographical location of the engaging agents. The agents can be in any location, indeed even on Earth and on the Moon.

Another difference is the fact that in a voice-loop communication there is no need to “go to” somebody before a communication can take place. This is similar to a phone communication (Same Time/Different Place (STDP) communication). However, different from a phone communication, there is no need to “call” someone before the communication can start. Therefore, a voice-loop communication is a combination of f-2-f and phone communication, it is a STSP/DP communication form.

To model the communication delay over the EVCS, I have developed a *voice-loop communication model* that includes the LM communication circuit as an additional agent with behavior. When a lunar surface agent makes an utterance (i.e. performs a communication transfer), this utterance is communicated to his partner and to the LM communication circuit agent. The LM communication circuit agent (LmComCircuit) communicates the utterance to the CapCom agent with a delay of one second⁴⁷. The result is that the CapCom agent will receive the communicated belief a second (a clock-tick) later, while the partner on the lunar surface will receive the belief instantaneous, i.e. at the moment of the communication.

Figure 6-36 shows how the voice-loop communication model lets the lunar surface agent AlBean communicate to both his partner PeteConrad and CapCom EdGibson that he is ready to start with the offload activity. First, the agent that speaks, AlBean in this case, has to have a belief about what needs to be spoken. Figure 6-36 shows this belief about the *speechAct* attribute being created in the *CommunicateReadyToOffload* workframe:

```
conclude((the speechAct of current = ReadyToOffloadAIspe));
```

The agent AlBean can now communicate this belief in the *Talk* communicate-activity:

```
Talk(vlcoms, start, OpenSEQBayDoorActivity, 10, 8);
```

This *Talk* activity transfers the belief at the start of the activity to all the agents bound to the *vlcoms* variable (PeteConrad and LmComCircuit in this case). The *Talk* activity is part of the *VoiceLoopCommunicator* group shown in Figure 6-37. Every member of the *VoiceLoopCommunicator* group, which the AlBean agent (as well as the PeteConrad and LmComCircuit agents) is a member of, inherits this *Talk* activity and will therefore be able to communicate its current belief about the *speechAct* attribute. Consequently, both the PeteConrad and the LmComCircuit agent receive AlBean’s belief about the *speechAct* attribute. Next, the LmComCircuit agent performs the *SendComToEarth* activity, which actually transfers the *speechAct* belief it just received from AlBean to the agents bound to the *vlagts* variable.

```
SendComToEarth(AlBean, vlagts);
```

The *vlagts* variable is bound to just the EdGibson agent (since he is the only agent with the *communicationType* equal to “MscVoiceLoop,” meaning he is the only agent listening to the voiceloop in MSC). The *SendComToEarth* activity, shown in Figure 6-36, has a duration of one second and transfers the *speechAct* belief at the end of the activity. Consequently, this describes the communication delay from the Moon to Earth. For longer delays one would simply increase the duration of *SendComToEarth* activity, making this a general model for voice-loop communication with communication delay.

⁴⁷ The Brahms clock grain-size cannot be set to 1.25 seconds, but has to be set to an integer number.

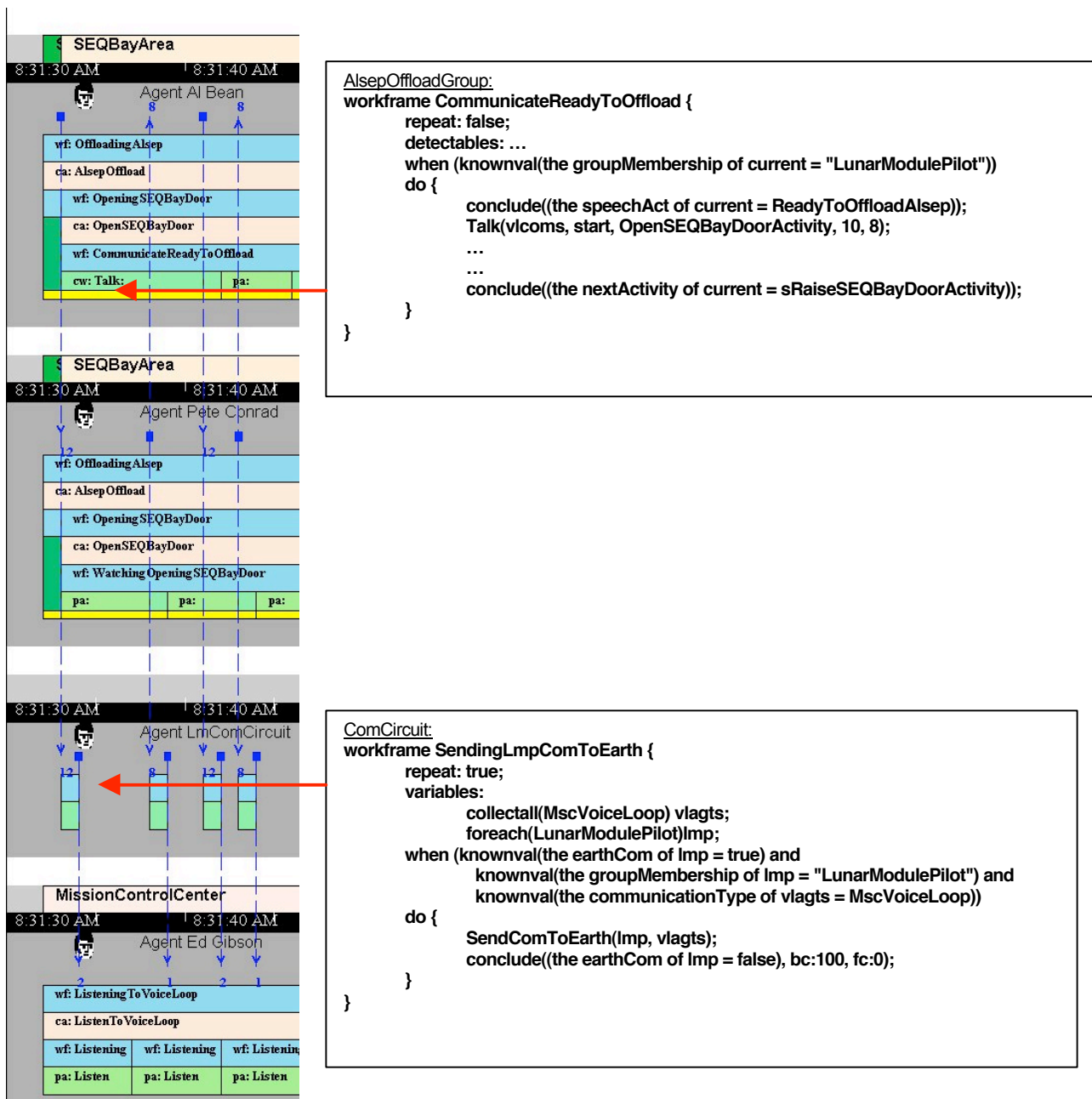


Figure 6-36. Voice-loop communication via LmComCircuit

6.7.3 The voice-loop library model

This voice-loop behavior is something that we want to re-use in other modeling efforts. I therefore developed this behavior as a library model that can be re-used over and over again. To do this we need to abstract the functionality of the voice-loop into separate functional groups. In this section, I describe the design of the voice-loop library model as it is shown working in Figure 6-36.

We can abstract the workings of the voice-loop system into two separate groups. First, there is a group of agents that can communicate over a voice-loop together. These agents are all members of the *VoiceLoopCommunicator* group. The *VoiceLoopCommunicator* group in turn is a member of the more abstract *Communicator* group. This group specifies those agents that can communicate in one way or another with each other, be it using a voice-loop, a telephone, e-mail, et cetera. There are three subgroups of the *VoiceLoopCommunicator* group, namely the *LmVoiceLoop*, the *CmVoiceLoop*, and the *MscVoiceLoop* group.

Then there is a group of agents that represent the communication circuits for each voice-loop. This is the *ComCircuit* group. This group has two member agents, namely one for the CM voice-loop, *CmComCircuit*, and one for the LM voice-loop, *LmComCircuit*. These two agents represent the communication circuits that create the delay of the communication between Earth and the Moon. The reason for modeling these as a group with agents, as opposed to a class with objects, is because we need these communication circuit agents to react to the communication transfers of *beliefs* from the “talking” agent. Although objects can receive and communicate beliefs, they cannot react to the beliefs they receive (objects only react to facts). All in all, it makes things easier from a modeling standpoint to model the communication circuits as agents, and since Brahms does not prescribe when to use agents versus objects this is a perfectly fine decision. The group hierarchy of the voice-loop model is presented in Figure 6-37.

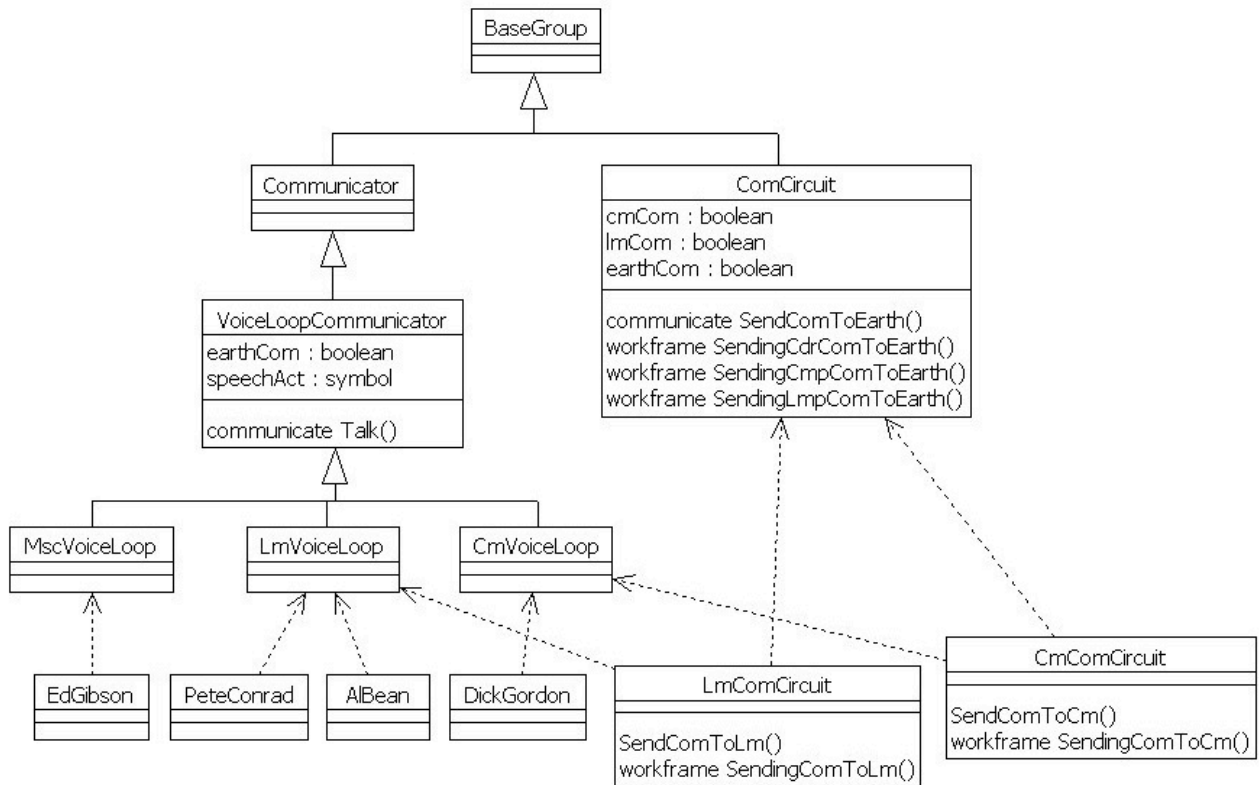


Figure 6-37. Voice-loop library model group hierarchy

6.8 OBJECT INTERACTION

We live in a world with objects. We look at them, touch them, and use them in our every day lives. When people work they use tools to accomplish what needs to be done. Interacting with objects in our environment is something so natural that we almost take it for granted when we consider how we do things. If we take a closer look at the work practice level, we need to include the way people interact with objects to describe what they do. On the moon the astronauts were together. However, they had artifacts with them, and objects that they needed to work on, and tools to use in their work. In this section, I describe how in Brahms we can model the interaction between objects and agents. I show the astronauts taking photographs and describe the model of the activities of the agent, and how the object it uses in these activities reacts and the way they both interact.

6.8.1 Lunar surface photography

Imagine taking a photograph. What do you do? What do you need? What does the camera do? Is it you or the camera that creates the photo? As I described before, all the tasks of the astronauts were planned and well trained. However, taking photographs was an acceptance to that rule. As it turns out, the Apollo

photographs were one of the most important scientific data returned to Earth. Some photographs were planned, but most were not, as is shown in the following example.

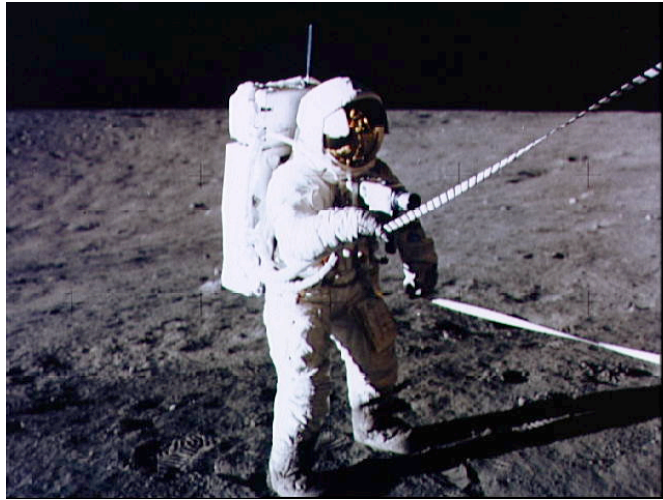


Figure 6-38. NASA picture AS12-47- 6913

Figure 6-38 shows a photograph that Al Bean took of CDR Pete Conrad, when he was lowering ALSEP Package-1 to the lunar surface. How did he do it? It is a subtle point, but it shows the collaboration between the two astronauts through the use of the photo camera.

116:32:48 Bean: Sure do. (Pause) Here it (probably the first package) comes.

116:32:53 Conrad: Coming right out.

116:32:54 Bean: And just about right. Riding right out on the boom, Houston. Sure looks pretty.

116:33:02 Gibson: (Making a mis-identification) Roger, Pete. We copy. (Long Pause)

116:33:36 Bean: (Wanting to take a picture) Look at me, Pete. (Pause) It's a good shot, babe. The LM and everything's reflecting in your visor. (Pause)

*[Al's photos AS12-47- [6913](#) (**) and [6914](#) (**) show Pete using a tape to guide the first of the ALSEP packages out of the SEQ Bay. Photo [47-6915](#) (**) was probably taken late in the ALSEP off-load.]*

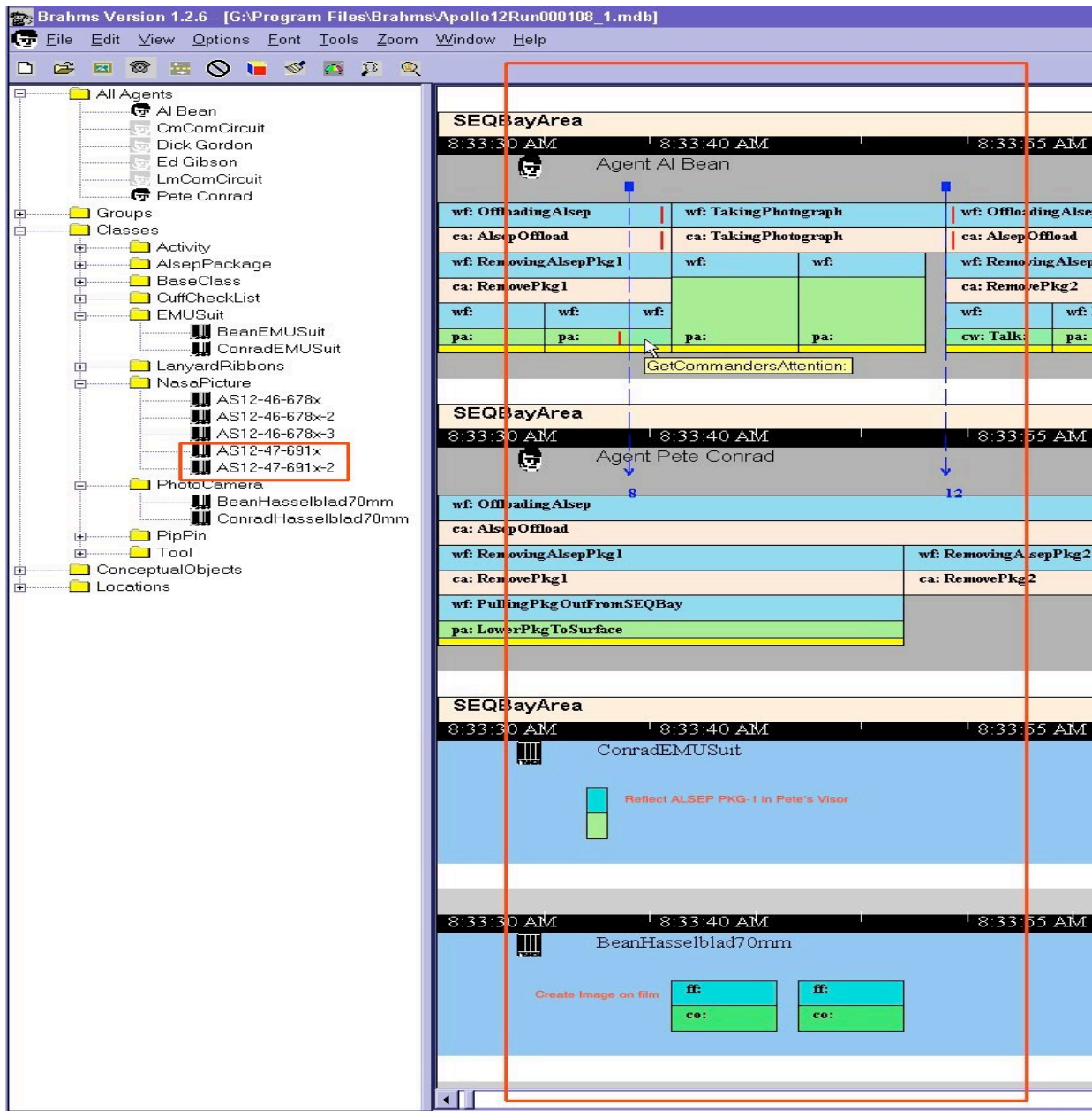


Figure 6-39. AI Bean taking two photographs of Pete Conrad

Figure 6-39 shows what happened during the simulation of the activity of taking this picture. First, it is important to realize how AI Bean decides to take a picture, and how this is modeled in Brahms. If we look at the utterance of AI Bean, we get some clues as to how this interaction happened. AI Bean says: “The LM and everything’s reflecting in your visor.” I interpret this as that the beautiful reflections in Pete Conrad’s visor of his EMU suit made him want to take a picture (see the beautiful reflection in Figure 6-38). You can see in Figure 6-39 that the ConradEMUSuit object creates the fact that there is a reflection from its visor. At that moment, agent PeteConrad performs the activity LoweringPkgToSurface. Agent AIBean detects the reflecting visor fact, while watching agent PeteConrad. This detection interrupts agent AIBean’s activity, and makes him perform the activity GetCommandersAttention. This activity represents the communication of AI Bean at time 116:33:36, where he says: “Look at me, Pete.” This communication is shown in Figure 6-39 by the first arrow. After this activity, agent AIBean starts the *TakingPhotograph* workframe shown in Figure 6-39 and described in Figure 6-40.

First of all, taking a photograph is something that is not related specifically to the ALSEP Offload activity. Therefore, the *TakingPhotograph* workframe is not defined in the *AlsepOffloadGroup* group. Instead, it is part of all possible activities for members of the *LunarSurfaceAstronaut* group, because every lunar surface astronaut can take photographs at any moment. It is therefore that the agent *AlBean* interrupts the *AlsepOffload* activity to start the *TakingPhotograph* activity.

Group LunarSurfaceAstronaut

Class PhotoCamera

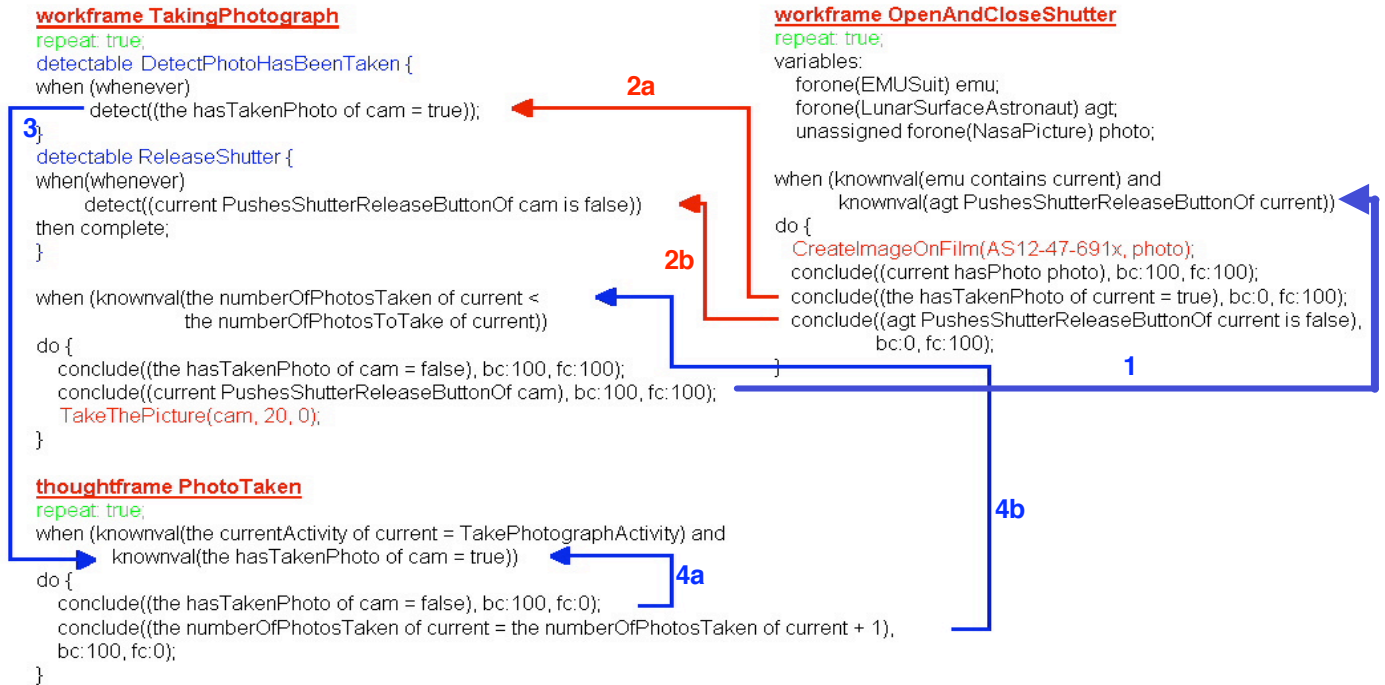


Figure 6-40. Taking a photograph

After he has taken the photographs, he continues with the interrupted *AlsepOffload* activity (see the small vertical four lines in Figure 6-39, at the beginning and end of the *OffloadingAlsep* workframe of agent *AlBean*). Figure 6-40 describes the interaction between the *agent* and a *PhotoCamera* object in order for the agent to take a photograph (this numbered list refers to the numbers in Figure 6-40):

1. After agent *AlBean* starts the workframe *TakingPhotograph*, due to the fact that it believes that the number of photos to take is smaller than the number he has taken (see the precondition), it is simulated that the agent pushes the shutter release button on the camera. This is represented by the creation of the belief and fact

(AlBean PushesShutterReleaseButtonOf BeanHasselblad70mm)

The creation of this fact triggers the *PhotoCamera* object *BeanHasselblad70mm* to perform the *OpenAndCloseShutter* workframe, due to the fact that its preconditions are now satisfied.

2. Next, the camera object performs the *CreateImageOnFilm* create-object activity. On the left side of Figure 6-39 this *dynamically-created* object is shown as a *NasaPicture* object (*AS12-47-691x*). This actually represents the photo in Figure 6-38. After this activity, the camera object creates three *facts*; first, it creates the fact that the photo object has been created. Secondly, it creates the facts that it has taken a photo and that the agent *AlBean* stopped pushing the shutter release button on the camera. These last two facts are detected by agent *AlBean*, who is still performing the *TakingPhotograph* activity (arrows 2a and 2b in Figure 6-40). Arrow 2b shows that the agent stops the *TakeThePicture* activity by performing a *complete* action in the *ReleaseShutter* detectable, simulating that the agent has pushed the shutter button and has taken the picture.

3. Arrow 3, at the same time, shows that the agent fires the thoughtframe *PhotoTaken*. This thoughtframe increases the agent's belief about the number of photos it has taken.
4. Last, but not least, arrows 4a and 4b, make sure that the agent takes the right number of photographs. In the example in Figure 6-39, the agent takes two photographs, one after the other.

This example shows a general model for taking pictures. The only thing the agent needs to start out with is its camera contained on his EMU suit. Later on in the ALSEP Offload activity, during the offload of the second package, the PeteConrad agent actually takes three photos of AlBean while he is lowering object *AlsepPkg2* to the ground, using the *ConradHasselblad70mm* PhotoCamera object (see Figure 6-41).

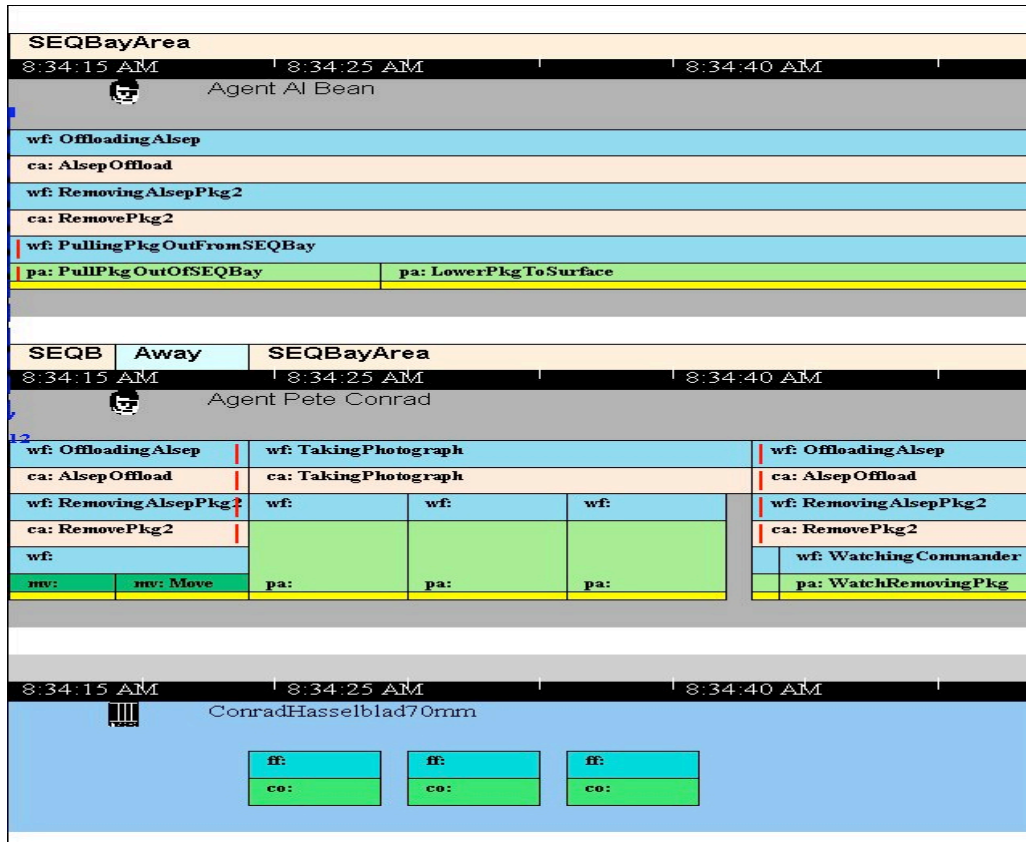


Figure 6-41. The PeteConrad agent taking photographs

Figure 6-42 shows the three actual photographs Pete Conrad took.

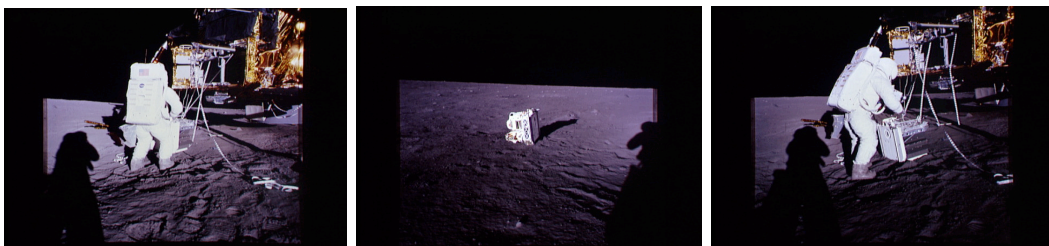


Figure 6-42. Photographs AS12-47-6783, 84, and 85 by Pete Conrad

A Brahms limitation

This example shows one of the limitations of a Brahms simulation. Although we can represent and simulate the taking of photographs, as well as the camera actually creating the *NasaPicture* objects, Brahms cannot show the *sightlines* of the camera, and that of the agents. The picture objects created do not include a

representation of what was captured on film (shown by the three photos in this example). The only way we could possibly represent this is to create beliefs that represent the scene being captured and “store” these beliefs in the NasaPicture object.

Not being able to model the line of sight of agents means that the model does not include whether the astronauts could actually see each other and/or the objects during their activities. Noticing other people and/or objects is often constrained by the line of sight. In Brahms, we can only model the detection of facts, based on the detectable being active and the existence of the fact in the world. However, in the real world the detection of certain facts depends on whether we can “observe the fact” through our field of vision, such as seeing someone in distress. Not being able to model the field of vision limits us in constraining the detection of facts based on the sightlines of the agent.

6.9 VERIFICATION AND VALIDATION

In this section, I describe the verification and validation (V & V) process I have followed to test the accuracy of the Apollo 12 ALSEP model. First, I will talk about V & V as a process and describe its elements, and some of its issues. Then, I will show in some detail the V & V steps I have followed and the results of this process in this experiment. Using this V & V process, I can say something about the accuracy of the model and my hypothesis about Brahms as a modeling and simulation language for *describing* a work practice.

To clarify the issues involved, I define the concepts *verification*, *validation*, and to be complete, *credibility* as follows:

- *Verification* is the process whereby the modeler asks if the model is performing as it was designed. In this step in the V & V process, the objective is to determine if the logic of the computer model correctly implements the assumptions made in the conceptual model.
- *Validation* is the process whereby the modeler asks how accurately the model is representing reality. That is, is it a good model of the intended work system?
- A *credible* model is one that the client accepts as being valid enough to use in making decisions. That is, is it a useful model for the task at hand? It should be noted that in this experiment we do not have a client that will make such a credibility judgment.

6.9.1 The purpose of verification and validation

An important part of modeling and simulation is the V & V of the model and the results of the simulation. Without a thorough V & V there is no ground in having any confidence in the model and the results of the simulation. Although it is important to realize that it is impossible to prove that a model is a general valid model (Robinson 1999). The reason for this is the fact that:

1. A model is only certified as valid with respect to its purpose. For instance, a model that has been created for the purpose of predicting the future state of a system might not be valid as a prescriptive model of the future system.
2. There are different interpretations of the real world possible. Depending on the worldview, or *Weltanschauung*, is a different interpretation of the real world and therefore, of the model and its validity (Checkland and Scholes 1990).
3. The data used to develop the model may be inaccurate. Even if that is not the case, it should be realized that the data used and the data generated by the simulation are but a small data sample. Therefore, they can only be seen as a probabilistic answer and not a definitive one.

The conclusion is that, although in theory a model is either valid or invalid, in practice it is not easy and often not possible to prove that a model is valid. Therefore, we have to think in terms of the confidence we can place in the model. The V & V of the model in this experiment is not one of demonstrating that the model is correct, but instead it is a process of *falsification*, i.e. demonstrating that the model is incorrect (Robinson

1999). In so doing, the purpose of V & V is to increase the confidence in the model, even though we might find inconsistencies and problems with the model according to the real-world data.

6.9.2 The verification and validation process

Many authors have described the process of a successful simulation (Law and Kelton 1991) (Kleindorf et al. 1998) (Banks et al. 1996) (Robinson 1994). All of them mention a series of processes that need to be followed. The high-level processes are shown in Figure 6-43, which is borrowed from (Robinson 1999). A simulation study first starts with understanding the *real world*, as well as the problem to be tackled. In this Brahms study, the real world is the Apollo 12 ALSEP Offload, with as the problem to be tackled, to test if we can describe the work practices of the lunar surface astronauts in a Brahms simulation. When the real world is sufficiently understood the modeling activity starts, and a *conceptual model* is described. For this study, I described the model as a qualitative model using a modeling approach called *World Modeling* (Sierhuis and Selvin 1996). After this, the model was coded into a computer model, in this case the Brahms language. When the model is complete, experiments are run to develop *solutions* to the real-world problem being handled. In this case, a greater *understanding* of the real world was obtained. In real-world projects it is hoped that the solutions found in the experiments can be implemented in the real world, or that the better understanding of the problem will lead to better decision making. In this experiment there has been no attempt to implement the model or change the real world based on the understanding, simply because this was not the purpose.

Even though there is a natural sequence in following these steps, it is obvious that the actual process is not strictly sequential, and that several iteration through the steps are necessary. This was also the case in this effort. First, there was no implementation phase based on the outcome of this study. Secondly, there were a number of cycles through the conceptual model, computer model and solution/understanding phase that were mostly driven by the validation and verification of the models with the real-world data. Even though this study did not end with an implementable solution in the real world, the process as depicted in Figure 6-43 still holds.

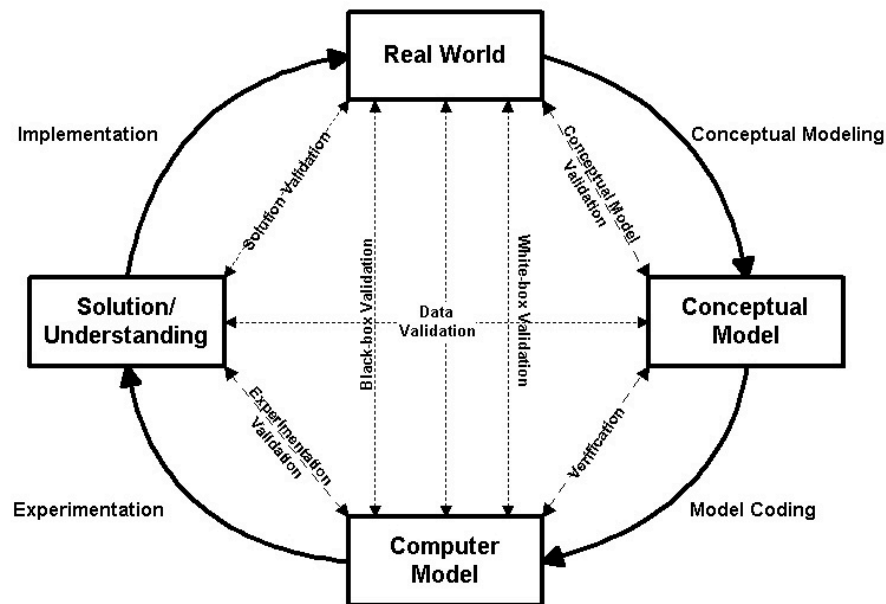


Figure 6-43. Simulation model verification and validation in the modeling process (borrowed from (Robinson 1999))

In the next sections, I will describe the activities of the three phases, conceptual model, computer model, and solution/understanding and the validation and verification methods used in each of these phases.

6.9.3 Data validation

As is shown in Figure 6-43, data validation is important at every step of the simulation process, because at each step in the process data is used. The data I used are all original NASA records of the actual Apollo missions. Table 6-4 lists all data sources that have been used in this case study. Since the Apollo missions are part of world history the facts and data are well known to the world and are therefore undisputed. By using the original lunar videos, as well as the transcripts of the original conversations of the astronauts, and the original photographs, mission reports and press releases, the validity of the data is very high. It can thus be said that, if the simulation data is validated against the original mission data, and it can be shown that the outcome is correct in relation to this data, the validity of the simulation model is high.

Table 6-4. Data sources used during experiment

Data Source	Data Type
Apollo Lunar Surface Journal	Transcriptions of actual astronaut voice loop recordings + mission photographs.
Apollo 14, 15 & 16 Video Tapes	Video Recordings of the actual Apollo missions from NASA.
Apollo 12 ,14, 15 & 16 Press Kits	Copies of the actual Apollo Press Kits from the Apollo missions, published by NASA.

6.9.4 Conceptual model validation

I started the modeling effort by creating a conceptual model of the Apollo 12 ALSEP Offload. The method used is called *Compendium*, and is described in (Sierhuis and Selvin 1996) (Selvin and Sierhuis 1999b) (Selvin and Sierhuis 1999a) (Selvin et al. 2001). The discussion of this method falls outside of this thesis. Figure 6-44 shows the *Raise SEQ Bay Door* activity described in the conceptual model. To model this activity, I used the voice loop transcription data from the Apollo LSJ (see Figure 6-45), as well as the Apollo video of the Apollo 14⁴⁸ mission. The voice loop data is modeled as the *communication* attribute in the model. By reading and listening to the communication, matching this to the mission plan, and validating this with the video, I was able to analyze who performed this activity (see Figure 6-45), and where in the voice loop transcription the astronaut was starting and ending this activity. The approach I used was to identify the activity duration based on communication sequences. The astronaut was performing the activity during the first utterance and the last utterance of a communication sequence. By using the timestamps in the Apollo LSJ, I calculated the total time of the activity (see Figure 6-47). I also represented where the agents were located while performing this activity, as well as what objects (artifacts) the astronaut was touching or using during this activity.

By analyzing the transcription of the voice loop data this way, I have represented and validated each activity of the agents. After this process was completed, the conceptual model had to be coded in the Brahms language.

⁴⁸ Due to a unfortunate problem with the camera, there is no video tape of the Apollo 12 ALSEP Offload.

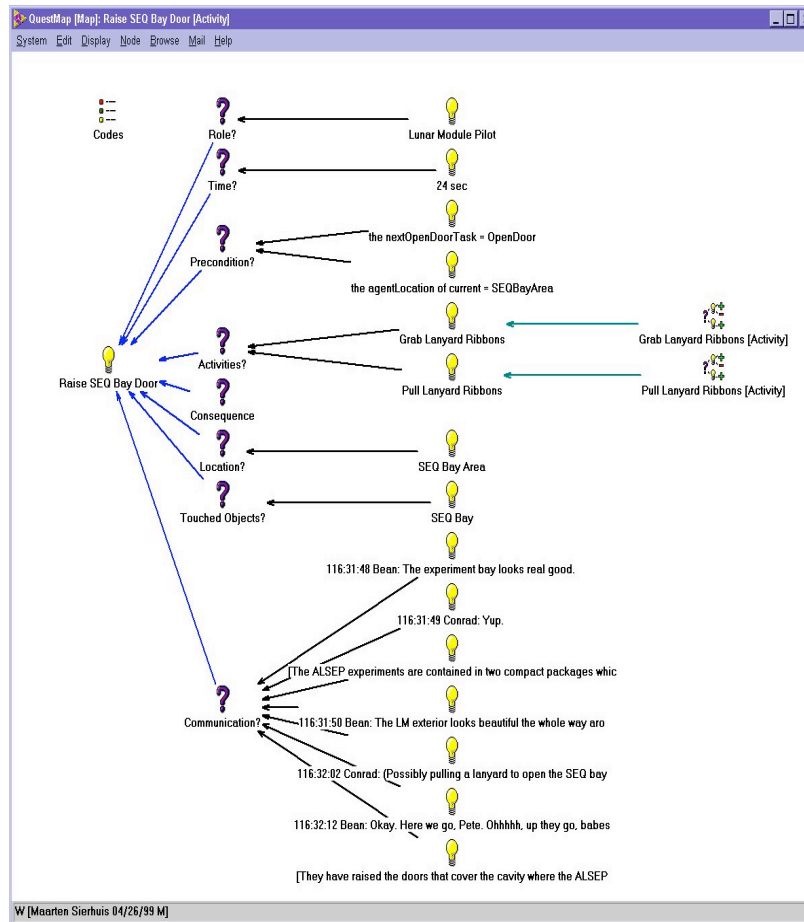


Figure 6-44. The conceptual model

The purpose of the conceptual model validation is to determine that the scope and level of detail of the proposed model is sufficient, and that all assumptions are correct (Robinson 1999). To describe this validation, let me take a step back and restate the problem I addressed in this study. The problem in this study was that of *showing that the Brahms modeling and simulation language is powerful enough to describe the work practice of the Apollo 12 lunar surface astronauts during the ALSEP Offload activity*. The level of model detail that is needed to test this hypothesis is given by the definition of what to include in a model of work practice (see chapter 3.2).

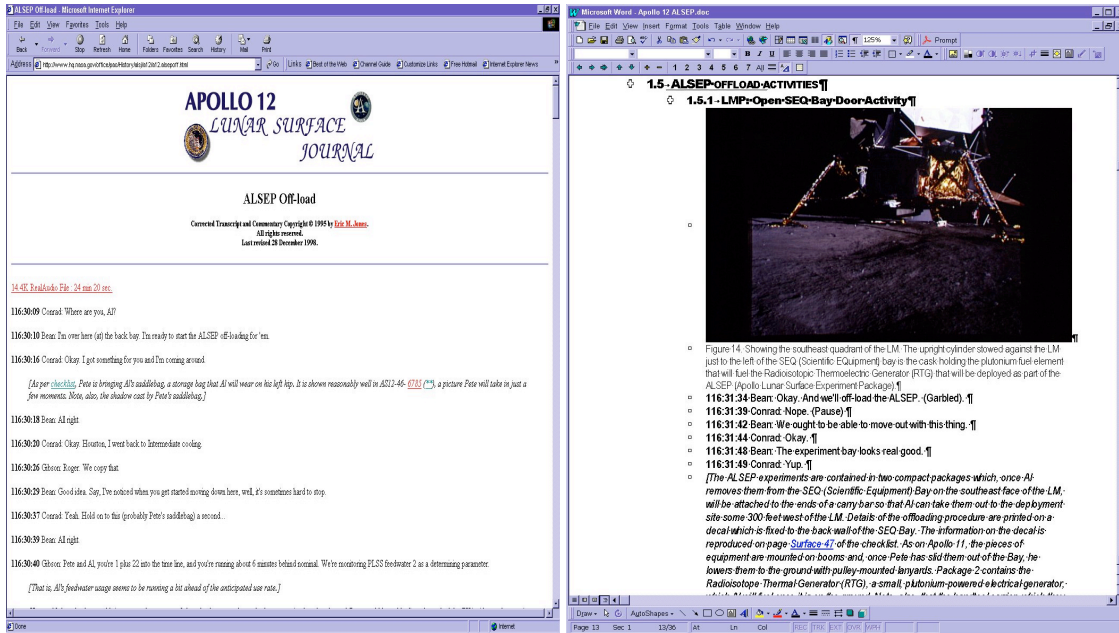


Figure 6-45. Voice loop transcription data from the Apollo LSI Figure 6-46. Voice loop transcription matched to activities

The image shows a Microsoft Word document with a table titled "1.1 ALSEP OFFLOAD ACTIVITY". The table has columns for Activity, Begin Time, End Time, Total Time, Δ from EVA Begin, Δ from ALSEP Offload Begin, Performer, Objects Worked On, Tools Used, and Location. A "Time Calculator" dialog box is overlaid on the table, showing start and delta times.

Activity	Begin Time	End Time	Total Time	Δ from EVA Begin	Δ from ALSEP Offload Begin	Performer	Objects Worked On	Tools Used	Location
1. Open SEQ Bay Door	116:31:34	116:32:22	00:00:48	1:23:06	0:00:00	LMP	SEQ Bay		LM (at the SEQ Bay)
2. Remove PKG-1	116:32:22	116:33:53	00:01:31	1:23:54	0:00:48	CDR	ALSEP PKG-1	Pulley-mounted Lanyard	LM (at the SEQ Bay)
3. Remove PKG-2	116:33:53	116:34:44	00:00:51	1:25:25	0:02:19	LMP	ALSEP PKG-2	Pulley-mounted Lanyard	LM (at the SEQ Bay)
4. Deploy Hand Tool Carrier	116:34:44	116:34:44	00:00:00	1:25:25	0:02:19	LMP	HTC		LM (at the SEQ Bay)
5. Unstow Cask Tools	116:34:44	116:34:44	00:00:00	1:25:25	0:02:19	LMP	Cask Tools		LM (at the SEQ Bay)
6. Stow Booms	116:34:44	116:34:44	00:00:00	1:25:25	0:02:19	LMP	ALSEP Booms		LM (at the SEQ Bay)
7. Unstow UHT	116:34:44	116:34:44	00:00:00	1:25:25	0:02:19	LMP	UHT		LM (at the SEQ Bay)
8. Close SEQ Bay Door	116:36:25	116:36:49	00:00:24	1:27:57	0:04:51	CDR	SEQ Bay		LM (at the SEQ Bay)
9. Remove Cable from CDR's Foot	116:36:49	116:37:30	00:00:41	1:28:21	0:05:15	LMP/CDR			LM (at the SEQ Bay)
10. Changing Antenna on CSM	116:37:38	116:38:45	00:01:07	1:29:10	0:06:04	CagCom/CM MP	High Antenna	Gain	CSM and LM (at the SEQ Bay)

Figure 6-47. Voice loop activity time analysis

If we take as a given the aspects of work practice from chapter 3.2, then we can validate that these aspects are indeed included in the model. Therefore, the validation method I used for the conceptual model was to analyze the important aspects of modeling work practice, as described in the theory, and to make sure that the conceptual model included all of them. Table 6-5 lists the aspects that were to be included in the model, as well as how these aspects are made operational in the coded model:

Table 6-5. Aspects of modeling work practice

Aspect of Work Practice	Model
Communities of Practice	This aspect is incorporated in the model by modeling the roles and functional groups of the agents as <i>groups</i> with behavior in the model. People who belong to certain CoP are represented as agents being members of the groups, inheriting the common behavior of the group members.
Activities	The behavior of all agents and artifacts is described in terms of primitive activities taking time, and composite activities decomposed into lower-level activities, and the lowest-level into primitive activities.
Collaboration	Collaboration is an emergent aspect of the model that is shown in the output data of the simulation. By describing the activities of agents, and the interaction and constraints that make each agent perform an activity based on activities of other agents, shows that agents are collaborating together.
Communication	The model includes all the speech acts from the real voice data. Activities are sometimes dependent on whether these speech acts are performed and received.
Real world artifacts	For each activity the artifacts that are used or touched in the activity are represented in the model. Relationships between activities and artifacts are represented.
Geography and Movement	For each activity, the location where the activity is performed is represented. Agents move from location to location, and performance of an activity is sometimes dependent on being in the location or noticing other agents and/or artifacts in a location.

6.9.5 Computer model verification

The next phase in the modeling process is the design and implementation of the Brahms model source code. In this phase, the modeler needs to translate the activities, groups, agents, classes and objects represented in the conceptual model into the Brahms language. To do this, the modeler needs to be proficient in the Brahms language, and specifically in the multiagent and activity programming concepts in Brahms. For first time Brahms modelers this is a painstaking process, and is similar to the compile-debug cycle in traditional programming languages, such as C++ or Java.

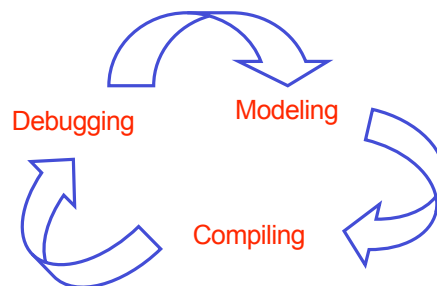


Figure 6-48. Brahms compile-debug cycle

Figure 6-48 shows the modeling cycle, which first continues until the complete model can be compiled without syntax errors by the Brahms compiler. However, verifying the model is more than getting the Brahms compiler to compile the model without syntax errors. Although this is of course a first and important step in the process, the most important step is to compare the “functioning” of the model with the conceptual model. The model validation and verification steps are driving the Brahms model development process, shown in Figure 6-49

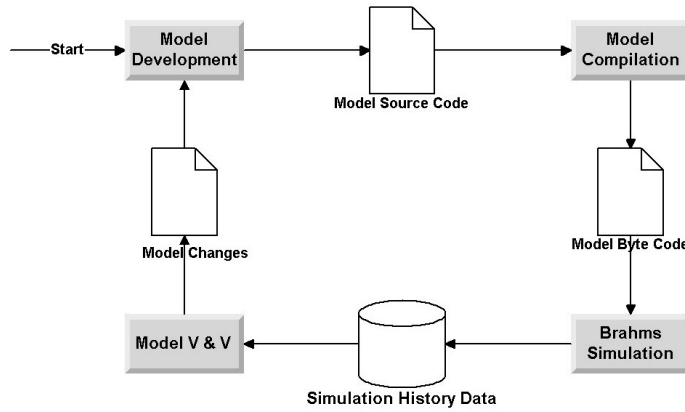


Figure 6-49. Brahms model development cycle

The functioning of the model is visually verified using the *AgentViewer* application. The AgentViewer is a separate Brahms application that uses the simulation history data to display a 2-dimensional graphical timeline view of the activities of agents and objects. The timeline figures in this and other subsequent chapters are all screenshots from selected agents and objects in the AgentViewer. Using the AgentViewer application the modeler can investigate the simulation run.

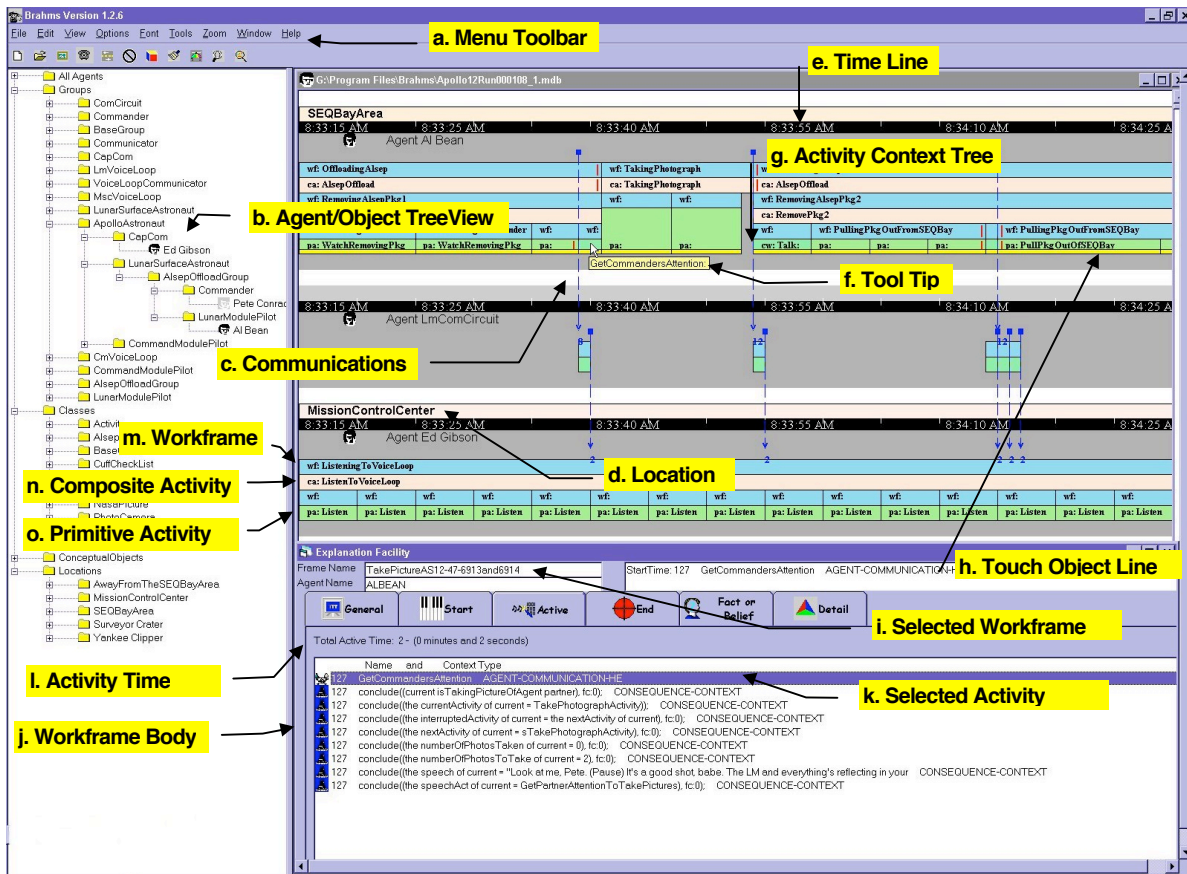


Figure 6-50. AgentViewer application

Figure 6-50 shows the AgentViewer. Using this application the end-user can select which agents and objects to view in the time-line view, and investigate the exact behavior of those agents and objects during the simulation (see a-l explanations in Figure 6-50):

- a. Using the menu-bar, the end-user can parse the simulation history data into a history database, and open a history database for viewing.

- b. When the database is opened all the agents and objects are loaded into the tree view. Using the tree view, the end-user can select which agents and/or objects (s)he wants to view in the time-line view.
- c. By selecting to view the agent/object communication, the (blue) arrows show all the communication activities, and the direction of the communication (sender and receivers). The communicated beliefs are also accessible by clicking on the square at the top of the sender side of the communication arrow.
- d. For each agent/object the "current" location is shown. When the agent/object moves to a new location, it is shown as a change in the location name and color.
- e. The time-line can show the time in different time-intervals, therewith zooming in and out.
- f. The tool-tip pops up when the mouse is moved over "hot spots". The hot spots are those areas where more information is available than can be shown on the screen. By moving the mouse over those areas the hidden information pops up in a tool-tip, such as the name of a workframe or activity.
- g. The Activity-Context Tree is the central piece of the agent/object time-line. It shows the workframe and activities hierarchy of the agent or object.
- h. The touch-object line is a (yellow) line that is shown when the agent/object is using certain objects in its activity. "Touch objects" are used to calculate the time those objects are used in activities.
- i. The explanation facility view is used to display more detailed information about the execution of workframes. By clicking on any workframe (light blue in color), an explanation facility window is opened for the workframe at hand.
- j. By selecting the "Active" tab in the explanation facility view, the executed statements in the workframe body are shown.
- k. You can select the statements in the workframe body to get more info.
- l. When you select a statement in the body of the workframe, the total time the activity was active is shown. Using the other tabs in this view, you can find out the exact time the workframe became available, as well as the exact time it became active and ended.
- m. Workframes are situated-action rules that execute activities. The top of a Activity-Context tree is always a workframe. You can recognize a workframe by the "wf:" symbol, followed by the name of the workframe. When the zoom-level is too high to contain the name of the workframe it is left out of the display. Using the tool-tip the user can find out the name.
- n. Composite Activities are executed by workframes, and contain lower-level workframes. You can recognize Composite Activities by the "ca:" symbol followed by the name of the activity. When the zoom-level is too high to contain the name of the activity it is left out of the display. Using the tool-tip the user can find out the name.
- o. Primitive Activities are executed by workframes, and are always at the bottom of the Activity-Context Hierarchy. You can recognize Primitive Activities by the following symbols, depending on the type of primitive activity: "pa:" (for a primitive activity), "mv:" (for a move activity), "cw" (for a communicate activity), "co:" (for a create object activity), followed by the name of the activity. When the zoom-level is too high to contain the name of the activity it is left out of the display. Using the tool-tip the user can find out the name.

Using this AgentViewer I have visually inspected the simultaneous behavior of the agents and objects, and compared the expected behavior from the conceptual model with the actual behavior during the simulation.

6.9.6 Experimentation validation

Comparing the model output to data from the real system is the most objective and scientific method of validation. Of course, this type of validation can only be performed if there is a real system, and real-world data that correspond to the simulation parameters. In this descriptive modeling experiment there was a real system back in the Apollo days. That system does not exist anymore, but what is most important is the fact that there is historical data available to validate our model.

I describe two types of quantitative data validation of the simulation output data of the Apollo 12 model, based on the historical data from the Apollo missions:

1. Validate the simulated activity times and duration with the activity times and duration derived from the timestamps in the Apollo 12 communication transcript from the Lunar Surface Journal (Jones 1997).
2. Validate the simulated voice loop communication with the voice loop recordings from the actual mission, which are transcribed in the same Apollo 12 communication transcript (Jones 1997).

6.9.6.1 White-box versus black-box validation

We consider two types of real-world data validation, *white-box* and *black-box* validation. The model verification described in section 6.9.5 is considered a *white-box* validation. Validating the simulated activity times with the timing of the activities based on the transcript of the voice loop communication is a *white-box* validation. The second validation, that of the actual voice loop data, is a *black-box validation*.

White-box validation is a *micro validation of the content* of the model. In a white-box validation we try to validate the model by investigating the model content in detail. The purpose of this type of validation is to ensure that the content of the model is true to the real world. The use of a graphical visualization and spreadsheet tools are very appropriate in this type of validation.

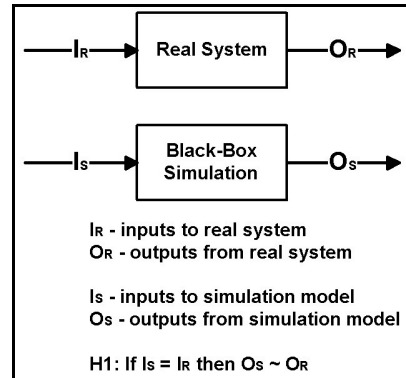


Figure 6-51. Black-box validation: comparison with the real system (from (Robinson 1994))

In a black-box validation we are not looking inside the model, but we are *validating the overall behavior of the model* with the output of prespecified real-world data. In this type of validation we need to validate that when we specify input data to the simulation model that is similar to that of the real system, the output data from the simulation should be relatively similar to that of the real system as well. This is a validation of the alternative hypothesis H1 (Figure 6-51).

6.9.6.2 Validate activity times

To validate the timing and duration parameters of the simulation model, we measure the activity times of the individual activities performed by each astronaut. Initially I had identified the activities of the astronauts based on the Apollo 12 communication transcripts (see Figure 6-45). Based on this and the fact that each communication utterance in the transcript is timestamped with the actual mission clock at MSC, I was able to calculate the ground-estimated time (GET) start and end times of the activity. From this the total activity time could be calculated (Figure 6-47). Here I am only showing the validation of the first three high-level

activities (Table 6-6). This is only in the interest of space, and I hope that with this example the reader is satisfied and can infer that the same holds for the other activities.

Table 6-6. Calculated activity times based on real-world data

Activity	Start GET	End GET	Total Time	? from ALSEP Offload Begin	Performer
Open SEQ Bay Door	116:31:34	116:32:22	00:00:48	0:00:00	LMP
Remove PKG-1	116:32:22	116:33:53	00:01:31	0:00:48	CDR
Remove PKG-2	116:33:53	116:34:44	00:00:51	0:02:19	LMP
		Total	0:03:10 (190 sec)		

An issue is the fact that the start and end times of the activities were chosen based on a thorough reading of the Lunar Surface Journal transcriptions, books and reports on the Apollo 12 mission, as well as the videos of the ALSEP Offload activities in subsequent missions. The choices I made are subjective to my own interpretation, as well as that of Erik Jones, the editor and creator of the Apollo Lunar Surface Journal (Jones 1997). It might well be possible that someone else would make a different interpretation of the timing based on the same data. Although this may be the case, it should not have much influence on the outcome of this study, since the goal of this validation is in context of the objective of the experiment. As mentioned before, the objective is to show that with Brahms we can describe the work practice of a real human activity system. We can still make a judgment on this, regardless of the fact that the subjectivity of the modeler is unavoidable in a modeling activity.

Table 6-7. Activity times for LMP Al Bean from simulation history database

DoneByID	DisplayText	Start ⁴⁹	Start SET ⁵⁰	End	End SET	TotalTime	Status
ALBEAN	OpenSEQBayDoor	1	8:31:34	49	8:32:22	48	COMPLETED
ALBEAN	RemovePkg1	49	8:32:22	53	8:32:26	4	INTERRUPTED
ALBEAN	ChangeEMUSuitCooling	53	8:32:26	58	8:32:31	5	COMPLETED
ALBEAN	RemovePkg1	58	8:32:31	124	8:33:37	66	INTERRUPTED
ALBEAN	TakingPhotograph	124	8:33:37	137	8:33:50	13	COMPLETED
ALBEAN	RemovePkg1	137	8:33:50	140	8:33:53	3	COMPLETED
ALBEAN	RemovePkg2	140	8:33:53	191	8:34:44	51	COMPLETED
					Total	190	

⁴⁹ The times in the Start, End, and TotalTime columns are in seconds.

⁵⁰ The times in the Start Simulation Elapsed Time (SET) and End SET are in the format h:mm:ss.

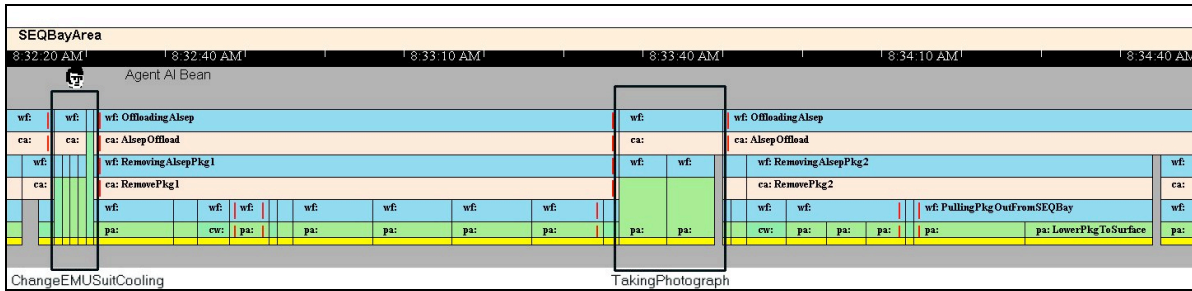


Figure 6-52. AI Bean's RemovePkg1 and RemovePkg2 activities

The activity times from Table 6-7 are the result of the emergent performance of lower-level activities of AI Bean, as can be seen in Figure 6-52. The timing of the composite activities from Table 6-7 are based on the cumulative times of the lower-level primitive activities performed as part of these composite activity.

Table 6-7 shows that AI Bean interrupts the RemovePkg activities twice to perform activities that are not necessarily part of the high-level AlsepOffload composite activity. The ChangeEMUSuitCooling activity is an activity that can be performed at the moment the astronaut feels too warm or too cold. Performing this activity is an interruption of the AlsepOffload activity and its underlying subactivities that are being performed at that moment. Consequently, the current RemovePkg1 activity will continue *after* the ChangeEMUSuitCooling activity is finished. You can see this represented in both Table 6-7 and Figure 6-52. Table 6-8 and Figure 6-53 show the activities for Pete Conrad. Pete Conrad does not perform the ChangeEMUSuitCooling activity (he is too busy offloading the package!), but he is interrupting his RemoveAlsepPkg2 activity taking three photographs while AI Bean is lowering the second ALSEP package.

Table 6-8. Activity times for CDR Pete Conrad from simulation history database

DoneByID	DisplayText	Start	Start SET	End	End SET	TotalTime	Status
PETECNRAD	OpenSEQBayDoor	1	8:31:34	50	8:32:23	49	COMPLETED
PETECNRAD	RemovePkg1	50	8:32:23	140	8:33:53	90	COMPLETED
PETECNRAD	RemovePkg2	140	8:33:53	170	8:34:23	30	INTERRUPTED
PETECNRAD	TakingPhotograph	170	8:34:23	189	8:34:42	19	COMPLETED
PETECNRAD	RemovePkg2	189	8:34:42	191	8:34:44	2	COMPLETED
				Total		190	

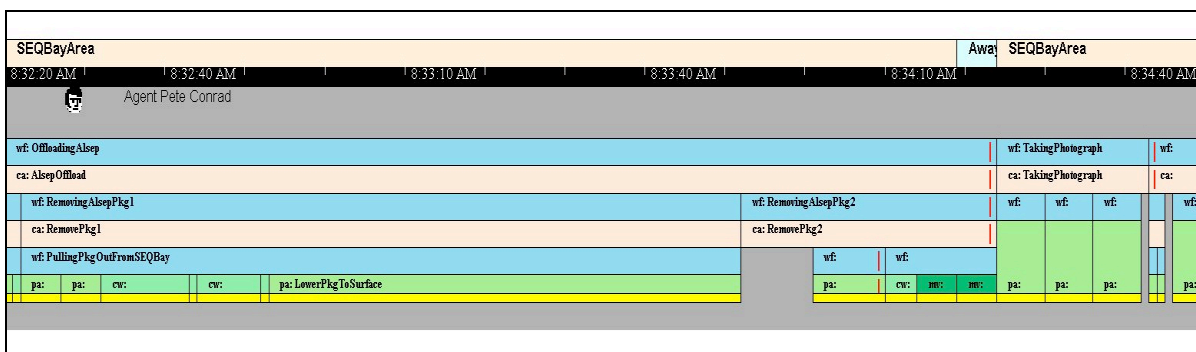


Figure 6-53. Pete Conrad's RemovePkg1 and RemovePkg2 activities

From Table 6-6, Table 6-7 and Table 6-8 it can be seen that the timing for both the AIBean and PeteConrad agents are similar as the timing data from the Apollo LSJ. With this verification the white-box validation of the model is completed, and we can state that the computer model content (i.e. the Brahms model) is a valid implementation of the conceptual model, which in turn is based on the Apollo 12 data.

6.9.6.3 Subtracting communication delay

The times, as shown in Table 6-6, present the following small but significant validation issue. The times based on the actual voice transcriptions are the mission times as they were measured by mission control. Since the activities are those of the lunar surface astronauts being performed on the moon, the actual times that the astronauts spoke the words transcribed in the Apollo LSJ documents would have had to be one and a quarter (1.25) second earlier. This is because there was a one and a quarter second delay between earth and moon communications (see section 6.7.1 about communication delay).

The start and end GET times are not the actual start and end times of the activities performed on the moon. Although the total activity time stays the same, to be correct, the activities of the astronauts need to start one and a quarter second earlier. To get to this point, I followed a two-step validation process:

1. Validation of the simulated activity times by making the times match up exactly with those measured on earth, shown in Table 6-6. The result of this validation was shown in Table 6-7 and Table 6-8.
2. After the simulation model is validated according to (1), we transpose the simulation times to include the earth/moon delay. Because only the start time is different, we can simply start the simulation clock earlier. This results in the activities of the astronauts starting at the actual start time, and thus in communication utterances arriving at mission control at the time measured by the GET clock. The result for agent AlBean is shown in Table 6-9

There is an issue with the capability of the simulation engine only being able to have an integer clock-grain-size. This means that we cannot simulate the one and a quarter second delay. The closest we can get is to have a clock-grain-size of one (1) second. Therefore, the delay I have been able to introduce in the simulation is one second. The numbers that have consequently been generated are still off by a quarter (0.25) of a second. However, this is a consistent error rate, and thus could easily be subtracted from the generated numbers.

Table 6-9. Activity times for LMP Al Bean including communication delay

DoneByID	DisplayText	Start	Start SET	End	End SET	TotalTime	Status
ALBEAN	OpenSEQBayDoor	1	8:31:33	49	8:32:21	48	COMPLETED
ALBEAN	RemovePkg1	49	8:32:21	53	8:32:25	4	INTERRUPTED
ALBEAN	ChangeEMUSuitCooling	53	8:32:25	58	8:32:30	5	COMPLETED
ALBEAN	RemovePkg1	58	8:32:30	124	8:33:36	66	INTERRUPTED
ALBEAN	TakingPhotograph	124	8:33:36	137	8:33:49	13	COMPLETED
ALBEAN	RemovePkg1	137	8:33:49	140	8:33:52	3	COMPLETED
ALBEAN	RemovePkg2	140	8:33:52	191	8:34:43	51	COMPLETED
					Total	190	

6.9.6.4 Validate output with real-world data

Next is the black-box validation. The purpose is to validate that the simulation can recreate the communication utterances by the astronauts exactly and at the same ground-elapsed time as the data from the Apollo LSJ. I show this validation of the model for the OpenSEQBayDoor activity as described in section 6.5.3. For ease of the reader, I repeat here the activity/communication table for the OpenSEQBayDoor activity.

Table 6-10. OpenSEQBayDoor activity with communication

LMP		CDR	
Communicate Ready To Offload		Watching Opening SEQ Bay Door	
activity	Communication	communication	activity
<i>Talk</i>	116:31:34 Bean: Okay. And we'll off-load the ALSEP. (Garbled).		<i>Watch Opening SEQ Bay Door</i>
		116:31:39 Conrad: Nope. (Pause)	<i>Talk</i>
<i>Talk</i>	116:31:42 Bean: We ought to be able to move out with this thing.		<i>Watch Opening SEQ Bay Door</i>
<i>Inspect SEQ Bay</i>		116:31:44 Conrad: Okay.	<i>Talk</i>
<i>Talk</i>	116:31:48 Bean: The experiment bay looks real good.		<i>Watch Opening SEQ Bay Door</i>
		116:31:49 Conrad: Yup.	<i>Talk</i>
Raising SEQ Bay Door			<i>Watch Opening SEQ Bay Door</i>
activity	Communication		<i>Watch Opening SEQ Bay Door</i>
<i>Grab Lanyard Ribbons</i>	116:31:50 Bean: The LM exterior looks beautiful the whole way around. Real good shape. Not a lot that doesn't look the way it did the day we launched it.		<i>Watch Opening SEQ Bay Door</i>
<i>Walk Back To Pull Ribbons Tight</i>			<i>Watch Opening SEQ Bay Door</i>
<i>Pull Lanyard Ribbons</i>		116:32:02 Conrad: (Possibly pulling a lanyard to open the SEQ bay doors) Light one. (Pause)	<i>Talk</i>
<i>Talk</i>	116:32:12 Bean: Okay. Here we go, Pete. Ohhhhh, up they go, babes. One ALSEP. (Pause)		<i>Watch Opening SEQ Bay Door</i>
<i>Pull Lanyard Ribbons</i>		116:32:22 Conrad: There it is.	<i>Talk</i>

Table 6-10 shows the subactivities of the OpenSEQBayDoor activity. The objective of this black-box validation is to have the simulation generate the exact communication utterance for each agent, at the exact time specified in. Of course, the same issue exists regarding the measurement of the time in GET and the communication delay to/from the moon. It should again be realized that the times in Table 6-10 are times measured by MSC, and are therefore the times that the CapCom agent heard the utterance over the voice loop, thus one and a quarter second later than the time the lunar surface astronauts uttered the words.

After having validated the activity times from the previous section, I changed the model to include the communication utterances specifically for this validation. To generate the exact utterance, the agent creates the utterance as a belief right before it communicates the belief in the Talk activity.

```

workframe CommunicateReadyToOffload {
  [detectable deleted]
  when (knownval(the groupMembership of current = "LunarModulePilot"))
  do {
    //communication from transcription
    conclude((the speech of current = "Okay. And we'll off-load the ALSEP. (Garbled).", bc:100, fc:0);
    conclude((the speechAct of current = ReadyToOffloadAIslep), bc:100, fc:0);
    Talk(vlcoms, start, OpenSEQBayDoorActivity, 10, 8);
    //end validation

    //communication from transcription
    conclude((the speech of current = "We ought to be able to move out with this thing."), bc:100, fc:0);
    Talk(vlcoms, start, OpenSEQBayDoorActivity, 0, 1);
    //end validation

    InspectSeqBay(0, 4);

    //communication from transcription
    conclude((the speech of current = "The experiment bay looks real good."), bc:100, fc:0);
    conclude((the speechAct of current = the exteriorAppearance of SEQBay), bc:100, fc:0);
    Talk(vlcoms, end, OpenSEQBayDoorActivity, 0, 1);
    //end validation

    conclude((the nextActivity of current = sRaiseSEQBayDoorActivity), bc: 100, fc: 0);
  }
}

```

Figure 6-54. Workframe with communication utterance from Apollo LSJ

Figure 6-54 shows the rewritten *CommunicateReadyToOffload* activity including the communication utterances. For each utterance there is a belief created for the *speech* attribute for the LMP agent (i.e. AIBean). This speech attribute signifies the actual speech-utterance of the agent.

```

communicate Talk(Communicator agt, symbol whn, Activity act, int pri, int maxd) {
  priority: pri;
  max_duration: maxd;
  resources: act;
  with: agt;
  about: send(the speech of current = value),
        send(the speechAct of current = value);
  when: whn;
}

```

Figure 6-55. Talk activity to validate communication

Next, the *Talk* communicate-activity actually communicates the *speech* belief to appropriate agents (Figure 6-55). Running the simulation again with this added communication, first and foremost, does not change the behavior of the agents. The end-result of the simulation is the same, as can be seen in Figure 6-56, but Table 6-11 shows that the simulation generates the actual voice loop communication transcription consistent with that in the Apollo LSJ.

The data in Table 6-11 is compiled from the history database. The data shows the communication of the *speech* attribute from the *LmComCircuit* agent. This is the agent that simulates the communication delay from/to the moon (see section 6.7, explaining the voice loop model). Therefore, the time this agent relays the communication should be equal to the GET from the Apollo LSJ. This is shown in the last two columns. The second to last column shows the simulated elapsed time (SET), which is the time from the simulation. The last column shows the ground-elapsed time (GET) as it is recorded at MSC, and is shown in the Apollo LSJ.



Figure 6-56. Voice loop communication for OpenSEQBayDoor activity

Table 6-11. Agent speech communication validation

Agent	FrameName	Attribute Name	Value (from Apollo 12 LSJ)	Start Time	Speech at SET	Speech at GET (from Apollo 12 LSJ)
LMCOMCIRCUIT	SendingAlBean ComToEarth	speech	"Okay. And we'll off-load the ALSEP. (Garbled)."	0:00:02	8:31:34	116:31:34
LMCOMCIRCUIT	SendingPeteConrad ComToEarth	speech	"Nope."	0:00:07	8:31:39	116:31:39
LMCOMCIRCUIT	SendingAlBean ComToEarth	speech	"We ought to be able to move out with this thing."	0:00:10	8:31:42	116:31:42
LMCOMCIRCUIT	SendingPeteConrad ComToEarth	speech	"Okay."	0:00:12	8:31:44	116:31:44
LMCOMCIRCUIT	SendingAlBean ComToEarth	speech	"The experiment bay looks real good."	0:00:16	8:31:48	116:31:48
LMCOMCIRCUIT	SendingPeteConrad ComToEarth	speech	Yup.	0:00:17	8:31:49	116:31:49
LMCOMCIRCUIT	SendingAlBean ComToEarth	speech	"The LM exterior looks beautiful the whole way around. Real good shape. Not a lot that doesn't look the way it did the day we launched it."	0:00:18	8:31:50	116:31:50
LMCOMCIRCUIT	SendingPeteConrad ComToEarth	speech	"Light one."	0:00:30	8:32:02	116:32:02
LMCOMCIRCUIT	SendingAlBeanCom ToEarth	speech	"Okay. Here we go, Pete. Ohhhhh, up they go, babes. One ALSEP."	0:00:40	8:32:12	116:32:12
LMCOMCIRCUIT	SendingPeteConrad ComToEarth	speech	"There it is."	0:00:50	8:32:22	116:32:22

The data from Table 6-11 shows that the simulation, indeed, generates the communication transcription from the Apollo LSJ, herewith validating the output of the simulation model. This concludes the validation of the model. In the next section some conclusions will be discussed.

6.10 CONCLUSION

In conclusion, I restate the research questions that needed to be answered, and show that indeed these questions are answered in this experiment. These questions are operationalized in the Apollo 12 ALSEP domain, and this operationalization is implemented in a Brahms model of the domain. The goal of this experiment was to investigate the use of the Brahms-language in order to *describe an existing work practice*. The challenge was to investigate if our theory of modeling work practice, implemented in the Brahms language, would be sufficient to describe the work practice in the chosen domain. The research questions were:

1. How can we represent the people, things, and places relevant to the domain?
2. How can we represent the actual behavior of the people, second by second, over time?
3. How can we show which of the tools and artifacts are used when, and by whom to perform certain activities?
4. How can we include the communication between co-located and distributed people, as well as the communication tools used, and the effects of these communication tools on the practice?

Table 6-12 shows how these questions were implemented in the Brahms model. The first column shows a more detailed instantiation of the research questions. The second column shows the operationalization based on the Apollo 12 mission. The third column shows how this is implemented in the Brahms model, thus answering the question in the first column.

Table 6-12. Answering the research questions

Research Question	Operationalization in Apollo 12 ALSEP Offload	Implementation in Brahms Model
How to represent people?	The astronauts Al Bean, Pete Conrad on the moon, CapCom Ed Gibson, and CMP Dick Gordon	Agents AlBean, PeteConrad, EdGibson, and DickGordon
How to represent Communities of Practice?	The different organizational roles of Commander, Lunar Module Pilot, Capsule Communicator, and Command Module Pilot. Also, the functional roles of "being an astronaut on the moon" and "offloading the ALSEP."	Hierarchy of different roles as groups of agents; ApolloAstronaut, CDR, LMP, CMP, CapCom, LunarSurfaceAstronaut, AsepOffloadGroup
How to represent artifacts?	The artifacts that are used and are important during the lunar surface activity of the two astronauts on the Moon; the LM, SEQ Bay, ALSEP packages, Lanyard Ribbons, Booms, Photo cameras, Space Suits, etc.	Class hierarchy representing types of objects, and objects being instances of classes to represent specific artifacts in the world; LM, SEQBay, AsepPkg1, AsepPkg2, Pkg1LanyardRibbons, Pkg2LanyardRibbons, etc.
How to represent places?	The areas where the astronauts are located, Mission Control, the	Type of areas as area definitions. Representing the Apollo 12

	Command Module, and the areas on the moon where the astronauts are working to offload the ALSEP, such as the area in front of the SEQ Bay, etc.	Geography model as the World area, containing the areas Moon, PlanetEarth, and LunarOrbit. Next, the separate areas part of the Moon. Mission Control is part of PlanetEarth, and the CommandModule area is part of the LunarOrbit area.
How to represent location of people and artifacts?	The lunar surface astronauts are located on the Moon, the CapCom is located in Mission Control, and the CMP is located in the Command Module.	Using the <i>initial_location</i> attribute in agents and objects. Each agent and object is given an initial location at the beginning of the simulation. From that moment on agents and objects have locations, which means they are located in an area and can move to other areas when needed.
How to represent actual behavior over time?	During the mission the astronauts are always performing activities. While the CDR and LMP are offloading the ALSEP packages the CapCom is listening on the voiceloop, etc.	The agent's real-life activities are represented as different types of Brahms activities that take time. Composite activities decomposed into primitive activities, communicate activities, and move activities. Behavior of objects, such as the astronaut's space suit and photo camera is also represented as activities. Next, the activities are executed as part of workframes, constrained by the agent's beliefs acquired or changed over time.
How to represent the use of tools and artifacts?	The lunar surface astronauts use tools to perform activities, such as the use of the lanyard ribbons to lower the ALSEP packages from the SEQ Bay.	The use of tools and artifacts in activities is represented using the <i>resources</i> attribute. Also, the generation and detection of facts represent the interaction or use of an artifact in an activity by an agent. The generation of facts is a representation of the actual physical interaction with the artifact being used in the activity. For example, in the <i>taking a photograph</i> activity the agent is using the PhotoCamera object.
How to represent communication?	The communication between the lunar surface astronauts on the moon, the CapCom in Mission Control, and the communication between CapCom and CMP.	Communication is represented as an activity. During this activity beliefs are communicated to/from agents. All the communication between the astronauts is represented as timed activities communicating speechacts, i.e. the speechact is represented as the value of the attribute

		<i>speechact</i> communicated as a belief from one agent to another.
How to represent communication tools?	The Apollo astronauts were on a communication voiceloop circuit with each other and CapCom at Mission Control. The communication time delay between Earth and the Moon was 1.25 seconds.	Voiceloop communication is represented as a communicate-activity, communicating with agents of the group VoiceLoopCommunicator. To represent the time delay in the communication a LmComCircuit agent represents the voiceloop circuit through which the communication is sent to/from Earth to the Moon. Both the agents on Earth and on the Moon communicate through this LmComCircuit agent.

In this Apollo 12 ALSEP Offload experiment I was able to represent the intricate detail of the human activities and collaboration using the Brahms language. The fact that the model generates all the communication between the astronauts, including the timing of the communication, shows that the Brahms Language is powerful enough to model and simulate the work practice of the astronauts on the Moon and on Earth. Of course the level of collaboration is shown in terms of the activities each agent is performing, as well as the location of the agents, the artifacts the agent is using at that moment, and the use of artifacts in the activity. It has been shown that the research questions posted are answered satisfactorily. Therefore we can say that the hypothesis is proven, and that with Brahms we are able to *describe an existing work practice*.

This concludes the first of three experiments to show that Brahms is a sufficient language for modeling and simulating work practice. In the next experiment I will show that with Brahms we can predict the future activity behavior of agents, based on a model of the work practice.