BRAHMS

A multi-agent programming language for simulating work practice

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

Research Institute for Advanced Computer Science[1]
Mail Stop 19-39
NASA Ames Research Center
Moffet Field, CA 94035
msierhuis@mail.arc.nasa.gov

Computational Sciences Division[2]
Mail Stop 269-1
NASA Ames Research Center
Moffet Field, CA 94035
bclancey@mail.arc.nasa.gov

Bell Atlantic Information Systems[3]
400 Westchester Avenue
White Plains, NY 10604
rvhoof@basit.com

ABSTRACT

A human work system is an example of a complex system of collaboration and coordination between intelligent agents, namely human beings. Our thesis is that if we can develop an agent-oriented language with which we can describe the collaboration between human agents, we will have created a multi-agent programming language that deals with agent interactions, such as the coordination, collaboration, and mobility of intelligent agents, including software agents.

Brahms is a multi-agent programming language for modeling and simulating human collaboration in a work system. The Brahms language has its roots in other agent-based languages, such as AGENT-0 (Torrance 1991), and PLACA (Thomas 1993). The Brahms language is based on the formal logic of computational multi-agent systems, as described by Wooldridge (Wooldridge 1992). However, as Wooldridge described, his theory was not intended as a model of human social systems. Brahms incorporates a theory of human social systems. Our theory focuses on meso human social systems —"as a mid-level theory that links a micro-level mechanisms to macro-level phenomena, in our case the physical and social to the cognitive" (Carley and Prietula 1994)— meaning that we try to describe a specific type of human system, namely that of a *human activity system* (Checkland and Scholes 1990). As such we needed to extend Wooldridge' formal logic with provisions for modeling human-actors (social agents); including their activities, collaboration, their environment, and the fact that they are situated in the real world, acting and observing, reacting to and interacting with other agents, objects, and artifacts.

In this paper, we will describe the Brahms language features using as an example a simulation of two human agents collaborating in an activity of correcting errors on work orders. The model shows the implicit coordination of the work activities of these two agents through the faxing of orders and a telephone conversation to resolve the error on the order.

Introduction to work practice modeling

Historical view

To explain why we became interested in creating an agent-based language to model and simulate human work practice, we need to go back in history. In an effort to design a new work system in one of the former Bell Telephone Operating Companies, the work system designers used an off-the-shelf workflow simulation tool to model the old and the new work process. The newly created design of this work process included a new designed coordination role. The work of a person playing this role did not include work activities that operated directly on the work product (i.e. the implementation of a high-speed data line) (Sachs 1995). It turned out that it was very difficult to include this type of coordination work into a workflow model. The result was that the workflow model did not explain the need for this new coordination role. The frustration of the design team was that they could not explain the importance of this new role using the model, while the rationale of their design was solely based on the introduction of this new coordination role.

Ironically, the company, because of the convincing power of the workflow simulation, adopted the design. However, in the implementation of the design the model gave little support and rationale for the design. In order to have a computer model that is convincing to the deciding powers, helpful for the understanding of the design, and helpful in the communication of that design, we started our effort in developing a modeling language and simulation environment that allows us to model the work activities and collaboration of people in a work process. Although useful, a workflow model typically omits collaboration, "off-task" behaviors, multi-tasking, interrupt and resume, informal interaction, and geography. In other words, workflow omits work practice (Clancey, Sachs et al. 1996).

Therefore, we developed the Brahms multi-agent language. Brahms allows us to model the work activities of each type of role, and each individual (or artifact) playing that role in an organization. The focus of a Brahms model is on the context of work, meaning, how does the work really happen. One of the essential requirements for Brahms is that we can model collaboration and coordination between people working on one task, as well as that people can work on more than one task at a time, and are interrupted and able to resume their activities where they left off.

Elements of work practice

To create a model that can show work practice, we need a way to describe it. Based on our research (Clancey, Sachs et al. 1996) we define a number of epistemological concepts:

- *Activities* are socially constructed engagements situated in the real world, which take time, effort and application of knowledge. Activities have a well-defined beginning and end, and do not have goals as used in the normal operation of expert systems. Instead, tasks and goals are conceptual constructs created and articulated within activities. Viewing work as activities allows us to understand why a person is working on a particular task at a particular time, why certain tools are being used or not, and why others are participating or not. This contextual perspective helps us explain the quality of a task-oriented performance. In this sense, activities are orthogonal to tasks and goals. While engaged in an activity people might articulate the task that they are working on, and the goal that they want to reach, but these are constructed within the activity, and not defined by the activity. An example of an activity is pursuing a research career. A goal within this activity might be to get a research paper accepted for the HICSS-32 conference. A task to reach that goal might be to gather all the relevant literature for the paper. The task and goal are created within the activity, but they are not defined by the activity (Clancey 1997). Conceptually we can view activities as the "what we are doing at each moment in time". Goals can be viewed as the "why we are doing what we are doing," while tasks can be viewed as the "how we are doing what we are doing."
- *Behavior* is defined by the socially constructed activities in which an individual is engaged in each day.
- C*ollaboration* is the collective activities of two or more individuals from a community of practice (Wenger 1997), communicating together within a work process in order to satisfy the individual goals.
- A *community* is a group of individuals, each with individual skills and knowledge, performing activities that collectively can be seen as a unity within the work process (Wenger 1997).
- *Communication* is the activity of directional transfer of beliefs, held by one individual called the sender, to one or more individuals called the receiver(s). After the transfer activity is complete, the receiver(s) will hold the same belief as the sender of the information.
- An *artifact* is a physical object in the world. When an artifact is being used in an activity, it becomes a tool for the activity.
- *Knowledge* is socially constructed, culturally defined, and fundamentally conceptual. Knowledge is not viewed as a body of descriptions in the heads of people, or in books. Knowledge is gathered in past and present activities, acquired by

observations from the environment and by interaction and communication with other individuals and artifacts. Knowledge is used to act in the present (Clancey 1997).

An example from the work practice in a telephone company

To show how in Brahms we can model collaborative activities between individuals, we describe a collaboration between a sales representative from the business sales office and the EMT trunk supervisor in a part of the Bell Atlantic company (formally NYNEX). After we have described the Brahms language, we then describe how we can model this collaboration in a Brahms model.

The collaboration between the sales representative and the trunk supervisor is situated around the communication of the network service request from the business sales office to the EMT group. The form of collaboration happens around the faxing of the network service request by the sales representative to the EMT group. The interesting aspect of this collaboration is that the sales representative and the trunk supervisor do not engage in a mutually agreed upon instantaneous form of communication. Instead, communication is asynchronous over distance. In case the trunk supervisor finds an error on the faxed service request form, he or she will call back the sales representative. How does the agent know whom to call back? This is actually an important factor in collaboration. How do we know with whom we are collaborating at any moment in time? In our example, many sales representatives fax service requests to the EMT group. Therefore, the obvious answer to this question is that the sales representative has written down his or her name and telephone number on the service request form, as the originator of the order. By reading this information from the fax the first line manager knows whom he or she has to call. The scenario goes somewhat like this:

> At nine o'clock in the morning sales representative Sierhuis, from the business sales office in Manhattan, faxes three service request forms from the day before. When Alan, the trunk supervisor of the EMT group comes into work, the first thing she does is to go to the fax machine and picks up all the faxes that have arrived that morning. She will go back to her office and will scan each service request fax to find any errors. If she finds an error on a service request, she will call back the negotiator whose name is written on the fax. In our scenario, one service request fax has an error on it. Alan will pick up the telephone and call sales representative Sierhuis. If sales representative Sierhuis (i.e. the negotiator) is at his desk when Alan calls him, he will answer the telephone, and together they will engage in a telephone communication activity to resolve the error. After the problem has been solved, they hang up the telephone and continue their other work activities.

Brahms: an agent-oriented language

In this section we describe the modeling formalism of Brahms. Brahms models are written in an agent-oriented language (AOL) that has a well-defined syntax and semantics. The Brahms language is a parsed language. A Brahms program is parsed by a LR(1) top-down parser, which generates an internal object representation for the run-time component. Using this language, a Brahms modeler can create Brahms models. The run-time component —the simulation engine— can execute a Brahms model; also referred to as a simulation run.

Brahms is a multi-agent programming language for modeling and simulating human collaboration in a work system. In Brahms, we use the notion of strong agency; i.e., the notion of agents that includes autonomy, social ability, reactivity, pro-activeness, mobility, and bounded rationality (Wooldridge and Jennings 1995). The reason for this is obvious, Brahms agents model human behavior. People are represented as agents, while artifacts (both physical and conceptual) are represented as objects. They generally have properties, such as geographical location, which may change over time depending on their interactions. The environment may also be modeled as having properties that change over time, which may depend on interactions between the modeled people and artifacts.

Agents

An *agent* in Brahms is the most central element in a Brahms model. An agent represents an individual person playing a particular role in an organization. The simulation engine schedules the constrained activities of agents. A Brahms model is always about the work activities of agents in a work process. Agents in Brahms are socially situated in the context of work, the organization, and its culture. However, the Brahms language is a multi-purpose AOL, which means that there is nothing inherent in the Brahms language that prevents the programmer from using agents for other purposes.

Actual individual's behavior

We can model the daily activities of actual individuals in an organization. For example, we can model the daily activities of agent "Trunk Supervisor Alan". The activities will then be defined local to the agent, and will be agent-specific (i.e. not inherited by other agents).

Generic individual behavior

Most Brahms models will not go into as much detail as to define the activities of an individual agent, but rather describe the behavior of abstracted groups of agents in entities called *groups*. In describing the activities of groups of agents, a specific agent will inherit the activities of the group. In this way, we can describe the daily activities of trunk supervisors in a more abstracted way (i.e., not specific to individuals).

Objects

In describing the work practice of individuals, we are interested in describing the context of their work. Doing so requires describing the environment in which people work, such as the artifacts they use and the interaction with these artifacts. Therefore, in Brahms, we describe artifacts and objects in the real world that are used in the practices of individuals. In a Brahms model, we represent artifacts, such as telephones, fax machines, and computer systems, as *objects*.

In a work process some of the information processing is done by artifacts, such as computer systems. To be able to describe the interaction between the people and such systems, for instance a database system, we need to describe the information processing capabilities of such systems. This led us to represent activities of artifacts as we represent activities of agents, although they are the conceptualization from the model-builder's point of view, not from the object's point of view, as is the case with agents.

Because in Brahms we are interested in describing the world with its animate and inanimate objects, we want the capability to make a distinction between human —intentional— agents and inanimate artifacts like a fax machine or a computer system.

Beliefs

"Beliefs" are verbal statements about states of affairs, and are not facts in themselves. Agents in Brahms have *beliefs* representing what they hold to be true in the world. Beliefs are represented as first-order propositions. For instance, suppose agent *A* believes that he is writing a conference paper about Brahms, and that it will be finished on time. *A* would then have the belief set:

{ BEL(Is-Writing (A, BrahmsPaper)), BEL(Will-Finish-On-Time(A, BrahmsPaper)) }

An agent will always start out with an *initial-belief* set that is defined at the agent’s either local-level or the group-level. Initial-beliefs are assigned in the initialization phase of a simulation. These initial-beliefs define the initial state for the agent. As the simulation time goes on agents will infer, detect and receive new beliefs, either based on their actions in the world, inferences, or communicated information.

Information

Objects can "store" information, such as the data stored in a database. In Brahms, we can store information in an object by giving the object "beliefs". In this way, we can model the data in a database system as beliefs of the object constituting the data. "Retrieving" the data from the database is done in the form of a communication activity with the object. For example, an agent retrieves data from a database in the activity "Doing payroll administration" by receiving the "beliefs" (data) from the database object, in a communication activity with this object. After this communication activity, the agent has new beliefs about the data; i.e., the data was transferred as new beliefs for the agent.

Facts

While deciding to perform an activity, human beings are situated in a physical environment, constrained by cultural norms. For example, we know not to stand up and talk out-loud when we are in a movie theater, watching a movie. We observe the state of our environment and use this information to decide our actions, based on cultural norms. In Brahms, we explicitly model the state of the environment as *facts* that can be observed. *Facts*, in Brahms, are factual states of the world. They represent a "birds-eye view" of the world of agents and objects. Facts are locally detectable by agents and objects, as specified by the workframes that model their behavior. When an agent detects a fact, it turns into the agent's belief. This way we can separate the state of the world from the state of an agent. Different agents can act differently to the state of the world. For example, we can model sound as a fact in the world, such as the "ringing" of a telephone. An agent "hearing" the telephone ring (i.e. detecting the fact) can decide to answer the telephone (as part of a broader activity), while other agents who can "hear" the sound, might not answer the telephone.

Activities

To understand activities we must first understand that human *action* is inherently social. The key is that "action" is meant in the broad sense of an "activity," and not in the narrow sense of "altering the state of the world." For example, the activity of being a trunk supervisor comprises many individual "tasks," such as "schedule work" or "test hypothesis," as found in expert systems and cognitive models.

However, instead of viewing "social activity" as something that people do together, such as "socializing at a party" or "the social chat before the meeting," we take a social scientist's view. Describing human activities as social means that the tools and materials we use, and how we conceive of what we are doing, are culturally constructed. Although an individual may be alone, as in reading a book, there is always some larger social activity in which he or she is engaged. For instance, the individual is reading the book in his hotel, as relaxation, while on a business trip. Engaging in the activity of "being on a business trip," there is an even larger social activity that is being engaged in, namely "working for the company," and so on. The point is that we are always engaged in a social activity, which is to say that our activity, as human beings, is always shaped, constrained, and given

meaning by our ongoing interactions within a business, family, and community. An activity is therefore not just something we do, but a *manner* of interacting. Viewing activities as a *form of engagement* emphasizes that the conception of activity constitutes a means of coordinating action, a manner of being engaged with other people and things in the environment.
The idea of activity has been appropriately characterized in cognitive science as intentional, a mode of being. The social perspective adds the emphasizes of time, rhythm, place, and a well-defined beginning and end.

Activities versus tasks and goals in expert systems

In AI and knowledge engineering (KE), reasoning behavior is described in terms of goal-satisfaction and tasks (Newell 1990; Schreiber, Wielinga et al. 1993). In expert systems, a task is performed by the description of the reasoning behavior of an expert, in terms of goals, sub-goals, and branch decision-points. Knowledge engineering has a narrow concept of knowledge, as being stored in the heads of the expert. Knowledge acquisition is seen as the transfer of this knowledge into a body of universal truths (i.e. the knowledge base). The expert is in the center, and his or her knowledge is transferable to anything or anyone who has to perform the same task.

Activities in Brahms

In Brahms, activities always take a certain amount of time. We define three types of activities:

(1) User-defined primitive activities

A user-defined *primitive activity* always takes an amount of time, and can represent any real-life activity we can think of. The amount of time a primitive activity takes may be specified by the model builder as a definite quantity or a random quantity within a range. However, because activities can be interrupted and never resumed, when an activity will finish cannot be predicted from its starting time. Primitive activities are atomic behaviors and are not decomposed. Whether something is modeled as a primitive activity is a decision made by the builder of any particular model. A primitive activity has a priority that is used for determining the priority of workframes.

(2) Built-in primitive activities

There are a number of built-in activities that are provided for within the Brahms language; *communication, broadcast, create-object,* and *move.* These activities are similar to user-defined primitive activities, but add certain pre-defined semantics. A communication activity transfers specified beliefs to other agents or objects. A broadcast activity broadcasts beliefs in the location the agent is currently in. All agents in that location will get the beliefs being broadcasted. A create-object activity creates a new object in the world. A move activity moves the agent or object to a new location. The predefined action of a built-in activity should be viewed as the result of engaging in the activity. For example, while the trunk supervisor is in the activity of "going to work," he or she is moving from his or her home to the office. At the end of the activity, the trunk supervisor has changed his or her location.

(3) Composite activities

Composite activities are always user-defined, and are decomposed activities into more detailed activities, either composite or primitive. How composite activities are defined is described in the next section on workframes.

Workframes

An agent cannot always apply all its available behaviors given the agent's cognitive state. Each activity is therefore associated with a conditional statement or constraint, representing a condition/activity pair, most of the time referred to as a production rule (Newell and Simon 1972). If the conditions of a rule are believed, then the associated activities are performed. In Brahms, such rules are called *workframes*.

A workframe defines an activity (or activities) that an agent or object may perform, as well as the preconditions to carrying out the activity. A workframe precondition tests beliefs held by the agent executing the workframe. A workframe detectable describes circumstances (in the form of facts about the world) an agent might observe while executing the workframe that could create a change of context for the agent. Consequences are used to represent cognitive state changes for the agent (i.e. new beliefs), and/or a new state in the world (i.e. a fact) due to the work done in an activity. So, we use consequences to model the effects of activities on both the individual and the world.

The relationship among these constructs is summarized in figure 1 below.

- GROUPS are composed of
  - AGENTS having
    - BELIEFS and doing
      - ACTIVITIES defined by
        - WORKFRAMES defined by
          - DETECTABLES, including IMPASSES
          - PRECONDITIONS
          - PRIMITIVE ACTIVITIES
          - COMPOSITE ACTIVIES
          - CONSEQUENCES

Figure 1. Taxonomy for groups, agents, beliefs and activities

A workframe is a larger unit than the simple precondition-activity-consequence design might suggest, because a workframe may model relationships involving location, object resources such as tools and documents, required information, other agents the agent is working with, and the state of previous or ongoing work. In this way, behavior may be modeled as continuous across time, and not merely reactive.

Activities in workframes

The actions of a workframe are one or more primitive activities, one or more composite activities or both. A composite activity includes one or more workframes, any of which may trigger other composite activities, each with its own workframes.

Composite activities

A composite activity expresses an activity that in turn may require several workframes to accomplish. Since activities are called within the action-part of a workframe, each is performed at a certain time within that workframe. Preference or relative priority of workframes can thus be modeled by grouping them into ordered composite activities. The workframes within a composite activity, however, can be performed in any order, depending on when their preconditions are satisfied. In this fashion, workframes can explicitly control executions of composite activities, whereas execution of workframes depends not on their order, but on the satisfiability of their preconditions and the priorities of their activities[1].

An example of activities and workframes

This section describes the workframes and the activities within the workframes for the sales representative and the trunk supervisor from the scenario in the example. Here, we describe them in a more or less natural language form[2]. Later we will give the actual Brahms syntax for this example.

workframes of a sales representative:

When it is time to go to work

Go To Work

this activity is necessary to have the agent go to work in the morning, i.e. the agent will leave home to go to the office. The agent knows what time he should go to work.

When the phone rings

[1] Composite activities can terminate in the following four ways. First, a composite activity terminates whenever the workframe in which it is declared terminates due to a detectable of type complete or abort; second, a composite activity terminates whenever a detectable of type complete or abort is detected within the composite activity; third, a composite activity terminates immediately whenever an end condition declared within the composite activity is activated; and fourth, a composite activity terminates when the model builder has defined it to be ended "when there is no more work available" and no more workframes in the composite activity are available or being worked on. During the execution of a composite activity, the engine continuously checks whether the agent has received a belief that matches any end-conditions.

[2] The words in *italics* are the preconditions of the workframe. **Bold** words show the activities.

- - **Pick Up Phone**
    this activity allows the agent to pick up the phone when it rings
- *When the phone conversation is done*
  - **Hang Up Phone**
    this activity allows the agent to hang up the phone when done with a phone call
- *When there are Service Request Forms to fax*
  - **Send Service Request Fax X To Emt Group**
    this composite activity allows the agent to fax the service request to the EMT group
    - **Faxing(Service Request Form X)**
      - **Go To Fax Machine**
        to send a fax the agent needs to go to the fax machine
      - **Send Fax(Service Request Form X)**
        this activity makes the agent actually fax a document
    - **Go Back To Office**
      after the fax has been send the agent needs to return to the office
- *When Agent X communicates an error on Service Request Form Y*
  - **Correct Errors On Service Request Form Y**
    this composite activity allows the agent to correct an error and communicate it to the trunk supervisor
    - **Read Service Request Form Y To Solve Error**
      this activity allows the agent read the form with the error on it
    - **Communicate Correction To Agent X**
      this activity allows the agent to communicate the correction to Agent X

<u>workframes of a trunk supervisor:</u>

- *When it is time to go to work*
  - **Go To Work**
    this activity is necessary to have the agent go to work in the morning, i.e. the agent will leave home to go to the office. The agent knows what time he should go to work.
- *When first get in to the office in the morning*
  - **Go To Fax Machine**
    the first activity is to check if there are service request faxes at the fax machine
- *When there is a fax at the fax machine*
  - **Pick Up Fax**
    this activity allows the agent to pick up a fax from the fax machine
- *When a Service Request Fax X is picked up at the fax machine*
  - **Process New Service Request Fax(Service Request Fax X)**
    for every new service request fax that comes in the agent has to process the service request, which means the agent checks the request for errors
    - **Scan Service Request Fax**
      in this activity the agent checks for errors
    - *When there is an error on Service Request Fax X*
      - **Correct Errors(Service Request Fax X)**
        this activity the agents calls the originator to correct the error
        - **Phone Call(Fax Originator Y)**
          this activity allows the agent to make a phone call
        - **Communicate Error Information**
          in this activity the agent communicates the error information from the service request fax to the Service Request Originator Y
        - **Correct Error On Service Request Fax**
          in this activity the agent corrects the error on the Service Request Fax
        - **Hang Up Phone**
          when the error is solved the agent hangs up the phone and stops the collaboration with Service Request Originator Y

Thoughtframes

Thoughtframes define deductions, mostly referred to as rules. Thoughtframes are similar to workframes, but are taken to be inferences an agent makes without executing any activities. Thoughtframes have the same preconditions as workframes. Thoughtframes have no activities, consume no time, and cannot be interrupted. Once the preconditions of a thoughtframe match, its consequences are automatically executed, similar to forward-chaining rules (Charniak and McDermott 1986).

Detecting change

A *detectable* is a mechanism by which, whenever a particular fact occurs in the world, an agent or object may notice it. Two things can occur in a detectable. First, the agent or object detects the fact and the fact becomes a belief of the agent or object. Second, the beliefs of the agent are matched with the condition used in the detectable, and if there is a match the action-part of the detectable is executed, which may abort or interrupt the workframe.

The action or then-part of a detectable defines the detectable type and is one of five keywords: *continue*, *abort*, *complete*, *impasse*, and *end-activity*. Continue is the default: the agent or object detects conditions, but the workframe proceeds unaffected. Abort causes the agent or object to stop executing the workframe. With complete, the agent or object only performs the remaining consequences of the workframe without doing the rest of the workframe's activities. With impasse, the continuation of the workframe is prevented until the condition is removed. End-activity is only meaningful when the detectable is in a composite activity: It does not detect facts, but causes the activity to be terminated immediately, based on matching the beliefs of the agent or object to the detectable condition. This allows an agent or object to abort working on a composite-activity.

Detectables can be used to model impasses. A common example of an impasse is inaccurate or missing information. With a detectable, an agent may notice passive observables, as when someone shouts, a fax machine beeps, or an agent is present vying for attention.

Multi-tasking agents

In a Brahms simulation, an agent may be doing multiple activities at any given time. At each clock-tick, the simulation engine determines which workframe should be selected, based on the priorities of available, current and interrupted workframes. Current work is selected in preference to interrupted or available work of equal priority, so that an agent tends to continue doing what it was doing. The selected workframe is then executed, leading the agent to act in the world and possibly begin a subactivity. When a workframe is interrupted, it is reexamined on subsequent clock-ticks to see whether it should be considered for selection. When an activity is interrupted, Brahms saves the line of activities and workframes so context can be reestablished after an interruption.

An important consequence and benefit of this activity-based programming paradigm is that all the workframes in a *workframe-activity hierarchy* are simultaneously competing and active. The ability of an agent to "work" (be active) on multiple activities is a major difference between Brahms and goal-oriented problem-solving systems, such as SOAR (Newell 1990) (Laird, Newell et al. 1987). The Brahms architecture is a *subsumption* architecture, which Brooks describes: "The fundamental decomposition of the intelligent system is not independent information processing units which must interface with each other via representations. Instead, the intelligent system is decomposed into independent and parallel activity producers […]." (Brooks 1991)

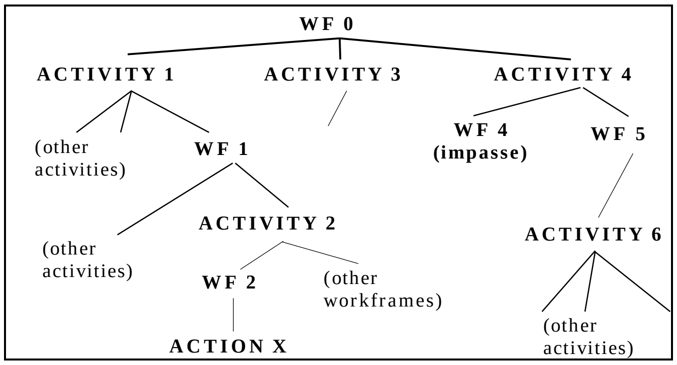


Figure 2. Multi-tasking in a workframe-activity hierarchy

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

With this activity-based paradigm, we can simulate the reactive situated behavior of humans. People are always working on many different activities, but our context forces us to be active in only one. However, at any moment we can change focus and start working on another competing activity, while interrupting others.

Geography

We live in a geographical space. Social interaction, collaboration, and agent activities are very much dependent on the geographical environment in which it takes place. Spatial relations provide a major constraint on work practice (Nowak and Latané 1994). Work is done in a 3-dimensional space, and it is therefore that we need to include a representation of the spatial relations between agents and artifacts in the environment. We need to be able to represent this physical world independent of the reasoning capability of agents about space. The geography design is described in more detail in (Steenvoorden 1995).

Geographical objects

Geographical primitives allow the model builder to model locations in the physical world and relationships between locations. The scope of the geography is limited to a city. Connectivity between cities is established and displayed through path relationships, rather than graphically. Geographical objects are those that can be placed in the city object, and generally fall in one or more of the following groups: areas, agents and artifacts. An area is a super-class for buildings. Areas can be connected through path relationships. The time it takes to travel —a move activity— along a path connection is specified in seconds on the path relationship. If there are no paths between two areas, —the simulation engine calculates the shortest path— it takes one engine clock-tick to travel along the path. In addition, agents and objects can be in only one area at a time, and can be located in an initial location at the start of the simulation.

A Brahms model of collaboration

To model the activities for the sales representative and the trunk supervisor, from our previous example, we model them at the group level. In the next three sections, we show how we define the Brahms activities and workframes at the group level. We give the definition of the groups in Brahms syntax. The activities section of a group first defines each activity. Once an activity is defined, it can be called in a workframe within the scope of the activity definition (i.e. the group or a composite activity in the group). However, first we describe the geographical places and the agent and artifact locations in the scenario. Second, we describe the activities of the sales representative and the trunk supervisor.

A model of the geography

Business Sales Office in Manhattan
artifact: Sales Rep Sierhuis' Telephone
artifact: Sales Rep Sierhuis' Fax Machine
agent: Sales Representative Sierhuis

EMT Trunk Supervisor Office in Manhattan
artifact: Alan's Telephone
artifact: Alan's Fax Machine
agent: Trunk Supervisor Alan

We model the locations in Brahms in terms of City and Building objects within a World object:

```
area Earth instanceof World { }
area NewYorkCity instanceof City partof Earth { }
area TheOldAdobeHut instanceof Building partof NewYorkCity { }
area BCSC_Office instanceof Building partof NewYorkCity { }
area EMT_TS_Office instanceof Building partof NewYorkCity { }
```

Next, we define the agents and artifacts and their initial location. This is done by asserting the initial building location as the location attribute of the agents and artifacts[3]:

```
agent Alan memberof EMT_TrunkSupervisor
{
    location: TheOldAdobeHut;
    initial_beliefs:
            (the telephoneStatus of AlanTelephone = free);
            (the phoneNumber of current = AlanTelephone);
            (AlanFaxMachine isTheFaxMachineOf current);
}
agent Sierhuis memberof BCSC_SalesRep
{
    location: TheOldAdobeHut;
    initial_beliefs:
            (the telephoneStatus of SierhuisTelephone = free);
            (the phoneNumber of current = SierhuisTelephone);
            (SierhuisFaxMachine isTheFaxMachineOf current);
}
object SierhuisTelephone instanceof Telephone {
    resource: true;
    location: BCSC_Office;
    icon: "phone";
    initial_facts:
            (the phoneNumber of Sierhuis = current);
}
```

```
object AlanTelephone instanceof Telephone {
            resource: true;
            location: EMT_TS_Office;
            icon: "phone";
            initial_facts:
                    (the phoneNumber of Alan = current);
}
object SierhuisFaxMachine instanceof FaxMachine {
            resource: true;
            location: BCSC_Office;
            icon: "faxmachine";
            initial_facts:
                    (current isTheFaxMachineOf Sierhuis);
}
object AlanFaxMachine instanceof FaxMachine {
            resource: true;
            location: EMT_TS_Office;
            icon: "faxmachine";
            initial_facts:
                    (current isTheFaxMachineOf Alan);
}
```

Definition of the EMT Group

In this group we define and implement activities that are common to all the agents in the EMT group. For example, every agent needs an activity "go to work". This activity is implemented in the EMTGroup. We are able to create a generic activity "go to work" because we are able to define an office location for every agent individually. Every agent will then go to his or her office when going to work.

```
group EMTGroup memberof BaseGroup
```

[3] "Current" is a predefined variable that is always bound to the agent or object that the active workframe belongs to.

```
{
    attributes:
            public Building officeLocation;  // the office building where the agent works
            public int timeToGoToWork;  // the time (in hours) the agent goes to work
    initial_beliefs:
            (the timeToGoToWork of current = 9);  // every member goes to work at 9:00 AM
    activities:
            move goToWork(Building myOfficeLocation)
            // this move activity makes the agent move to his office building. The duration is
            // calculated based on the path duration between the agent's current location and the
            // office location parameter
            {
              display: "go to work";
              priority: 0;
              random: false;
              min_duration: 0;
              max_duration: 0;
              location: myOfficeLocation;  // the office to move to
            }

    workframes:
            // When the agent knows it is time to go to work and the agent knows where his
            // office is, then the agent will go to his office.
            workframe GoToWork
            {
              repeat: false;
              variables:
                forone(Building) myOfficeLocation;
            when (
              knownval(the hour of Clock = the timeToGoToWork of current) and
              knownval(the officeLocation of current = myOfficeLocation))
              do
              {
                goToWork(myOfficeLocation);
              }/
            }//end workframe GoToWork
}//group EmtBaseGroup
```

Definition of the Sales Representative group

The sales representative in our example is the sales representative from the business sales office in Manhattan (the BCSC). We therefore define the sales representative group as the BCSC_SalesRep group. This group is a member of the EmtGroup, because it is part of the EMT example model, and therefore we want the sales representative to contain all the global activities of the model defined in the EmtGroup. Secondly, the sales representative needs to be able to use the telephone. Therefore, the group is a member of the PhoneUser group. The PhoneUser group defines how to use a telephone to make a call or pick up a ringing phone (not described). Finally, the sales representative also needs to be able to fax a document, therefore it is a member of the FaxUser group. This group defines how to send and receive a fax document (not described).[4]

```
group BCSC_SalesRep memberof EMTGroup, PhoneUser, FaxUser
{
    initial_beliefs:
            (the officeLocation of current = BCSC_Office);  // the sales rep's office
    activities:
            // This composite activity allows the agent to correct an error and communicate it to
```

[4] To safe space, certain elements of the model are left out. In these case you will see ellipses ("…").

```
            // the trunk supervisor.
            composite_activity correctErrorsOnSrFormActivity(…)
            {
               …
               activities:
                  // This communication activity communicates that the error is corrected to the
                  // EMT trunk supervisor.
                  communicate sendErrorIsCorrectedToEMT_TS(…)
                  {
                     …
                     with: TrunkSupervisor;
                     about: send(the error of SrFormFax = is_corrected);  // belief communicated
                  }
                  // In this activity the agent reads the form with the error on it.
                  primitive_activity readSrFormToSolveError(…)
                  {…}

               workframes:
                        // When the sales rep is communicating with the trunk supervisor and the
                        // the trunk supervisor cannot read some fields on the fax
                        workframe IllegibleFieldsError {
                           variables: forone(EMT_TrunkSupervisor) TrunkSupervisor;
                        when (
                           knownval(TrunkSupervisor isCommunicatingWith current) and
                           knownval(the error of SrFormFax = illegible_fields))
                           do
                           {
                              readSrFormToSolveError(…);
                              conclude((the error of SrFormFax = is_corrected), bc: 100, fc: 0);
                              sendErrorIsCorrectedToEMT_TS(…);
                           }
                        }// end workframe IllegibleFieldsError
            }// end composite_activity correctErrorsOnSrFormActivity

workframes:
            // When it is between 9:00 and 10:00 o'clock in the morning and the agent is in the
            // office, and there is a  service request form then send the form to the EMT group.
            workframe SendServiceRequestFaxToEmtGroup
            {
               variables:
                        foreach(ServiceRequestForm) SrForm;
                        unassigned forone(EnterpriseOrder) ConceptualSr;
            when (
               knownval(the hour of Clock = 9) and
               knownval(the agentLocation of current = the officeLocation of current) and
               knownval(the objectLocation of theSrForm = the officeLocation of current))
               do
               {
                        receiveInfoFromSrForm(…);
                        conclude((the needsToSendAFax of current = true), bc: 100, fc: 0);
                        conclude((TS_Alan needsToReceiveFax SrForm), bc: 100, fc: 0);
                        conclude((current hasToFax SrForm), bc: 100, fc: 0);
               }
            }// end workframe SendServiceRequestFaxToEmtGroup

            // When there is an error on the form, and the error is not yet corrected, correct the
            // error on the form.
            workframe CorrectErrorsOnSrForm
            {
               variables:
                        foreach(ServiceRequestFormFax) SrFormFax;
                        forone(EnterpriseOrder) ConceptualSr;
                        forone(ServiceRequestForm) SrForm;
            when (
               known(the error of SrFormFax = unknownval) and
               knownval(ConceptualSr isAConceptualObjectOf SrForm) and
               not(the error of SrFormFax = is_corrected))
               do
               {
                        correctErrorsOnSrFormActivity(…);
```

```
                    }
                  }//end workframe CorrectErrorsOnSrForm
      }//group BCSC_SalesRep
```

Definition of the Trunk Supervisor group

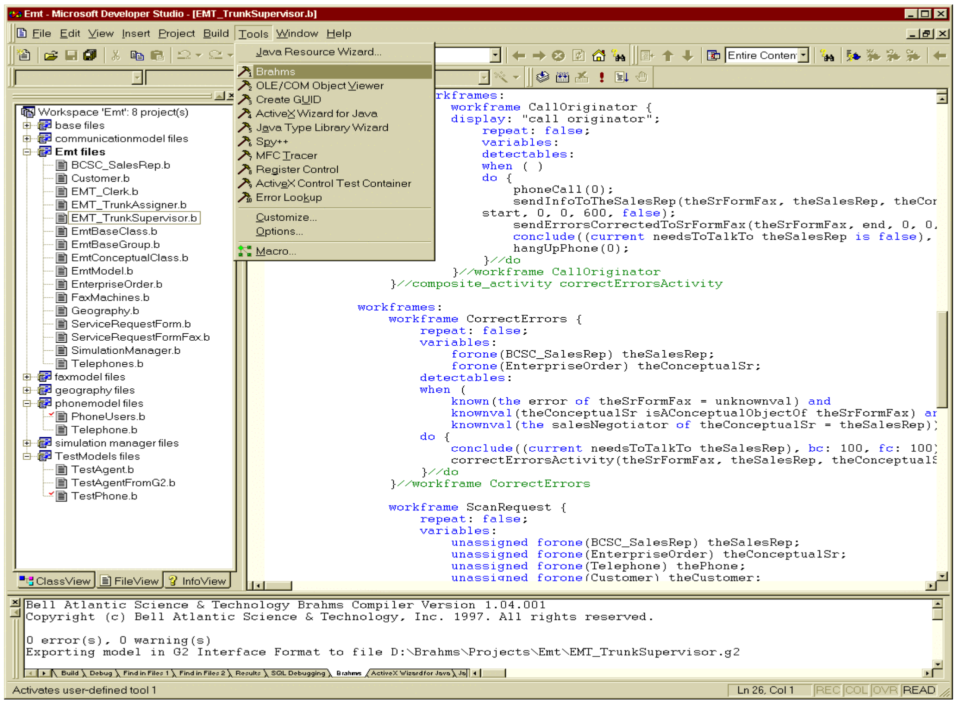
The trunk supervisor in our example is the trunk supervisor in the EMT group. We therefore define the trunk supervisor group as the EMT_TrunkSupervisor group. This group is also a member of the EmtGroup. Secondly, the trunk supervisor also needs to be able to use a telephone and a fax machine.

```
group EMT_TrunkSupervisor memberof EmtGroup, PhoneUser, FaxUser
{
    initial_beliefs:
            (the officeLocation of current = EMT_TS_Office);  // the trunk supervisor's office
    activities:
        composite_activity(…)
        // for every new service request fax that comes in the agent has to process the service
        // request, which means the agent checks the request for errors. This composite activity
        // allows the agent to do this.
        {
          …
          activities:
                communicate readServiceRequestFax(…)
                {
                  …
                  with: SrFormFax;
                  about:
                      receive(ConceptualSr isAConceptualObjectOf SrFormFax),
                      receive(the processingStatus of ConceptualSr = some_value),
                      receive(the error of SrFormFax = some_value),
                      receive(the salesNegotiator of ConceptualSr = SalesRep),
                      receive(the callingCustomer of ConceptualSr = Customer),
                      receive(the phoneNumber of theSalesRep = thePhone);
                }
                composite_activity correctErrorsActivity(…)
                {
                  …
                  activities:
                      communicate sendInfoToTheSalesRep(…)
                      {
                        …
                        with: SalesRep;
                        about:
                            send(ConceptualSr isAConceptualObjectOf SrFormFax),
                            send(the error of SrFormFax = unknownval);
                      }

                      communicate sendErrorsCorrectedToSrFormFax(…)
                      {
                        …
                        with: SrFormFax;
                        about:
                                send(the error of SrFormFax = errors_corrected);
                      }
                workframes:
                      workframe CallOriginator
                      {
                      when ( )
                        do
                        {
                            phoneCall(…);
                            sendInfoToTheSalesRep(…);
                            sendErrorsCorrectedToSrFormFax(…);
```

```
                                        conclude((current needsToTalkTo SalesRep is false), bc: 100, fc: 100);
                                        hangUpPhone(…);
                                      }
                                    }//end workframe CallOriginator
                              }//end composite_activity correctErrorsActivity

                   workframes:
                              workframe CorrectErrors
                              {
                              variables:
                                    forone(BCSC_SalesRep) SalesRep;
                                    forone(EnterpriseOrder) ConceptualSr;
                              when (
                                    known(the error of SrFormFax = unknownval) and
                                    knownval(ConceptualSr isAConceptualObjectOf SrFormFax) and
                                    knownval(the salesNegotiator of ConceptualSr = SalesRep))
                                 do
                                 {
                                    conclude((current needsToTalkTo SalesRep), bc: 100, fc: 100);
                                    correctErrorsActivity(…);
                                 }
                              }//end workframe CorrectErrors

                              workframe ScanRequest
                              {
                              variables:
                                    unassigned forone(BCSC_SalesRep) SalesRep;
                                    unassigned forone(EnterpriseOrder) ConceptualSr;
                                    unassigned forone(Telephone) Phone;
                                    unassigned forone(Customer) Customer;
                              when ( )
                                 do {
                                    readServiceRequestFax(…);
                                 }
                              }//end workframe ScanRequest
              }//end composite_activity processNewServiceRequestActivity

              workframes:
                    workframe ProcessNewServiceRequest
                    {
                    variables:
                          foreach(ServiceRequestFormFax) SrFormFax;
                    when (
                          knownval(SrFormFax isAFaxFor current) and
                          knownval(the processingStatus of SrFormFax = start_process_new_service_request) and
                          knownval(the agentLocation of current = the officeLocation of current))
                       do
                       {
                          processNewServiceRequestActivity(…);
                          conclude((the processingStatus of SrFormFax = done_process_new_service_request), fc: 0);
                      }
                    }//end workframe ProcessNewServiceRequest

                    workframe FirstActivityOfTheDay
                    {
                    variables:
                          forone(FaxMachine) FaxMachine;
                          forone(Building) faxMachineLocation;
                    when (
                          knownval(the hour of Clock <= 9) and
                          knownval(the agentLocation of current = the officeLocation of current) and
                          knownval(FaxMachine isTheFaxMachineOf current) and
                          knownval(the objectLocation of FaxMachine = faxMachineLocation))
                       do
                       {
                          goToTheFaxMachine(…);
                       }
                    }//end workframe FirstActivityOfTheDay
         }//end group EMT_TrunkSupervisor
```

The Development environment

The Brahms parser is written in the Java programming language. The parser is semi-integrated with Microsoft® Developer Studio™. Semi-integrated means that we have source-code control via Microsoft Visual SourceSafe™, source code coloring, file-level control via the Developer Studio FileView window, and access to the Brahms parser via the Developer Studio tools menu (see Figure 3). The parser errors show up in the error window. At this moment, there is no integration between parser generated errors and Developer Studio. This means that we cannot jump automatically to the source code line where the error occurs. However, the line numbers in the parser errors correspond to the line numbers in the source code files, and with a simple "goto-line" command we can find the error in the source code. At this moment, we also do not have any group-agent and class-object hierarchical representation within the Developers Studio ClassView window.



```
rkframes:
    workframe CallOriginator {
    display: "call originator";
        repeat: false;
        variables:
        detectables:
        when ( )
        do {
            phoneCall(0);
            sendInfoToTheSalesRep(theSrFormFax, theSalesRep, theCor
        start, 0, 0, 600, false);
            sendErrorsCorrectedToSrFormFax(theSrFormFax, end, 0, 0,
            conclude((current needsToTalkTo theSalesRep is false),
            hangUpPhone(0);
        }//do
    }//workframe CallOriginator
}//composite_activity correctErrorsActivity

workframes:
    workframe CorrectErrors {
        repeat: false;
        variables:
            forone(BCSC_SalesRep) theSalesRep;
            forone(EnterpriseOrder) theConceptualSr;
        detectables:
        when (
            known(the error of theSrFormFax = unknownval) and
            knownval(theConceptualSr isAConceptualObjectOf theSrFormFax) ar
            knownval(the salesNegotiator of theConceptualSr = theSalesRep)
        do {
            conclude((current needsToTalkTo theSalesRep), bc: 100, fc: 100)
            correctErrorsActivity(theSrFormFax, theSalesRep, theConceptualS
        }//do
    }//workframe CorrectErrors

    workframe ScanRequest {
        repeat: false;
        variables:
            unassigned forone(BCSC_SalesRep) theSalesRep;
            unassigned forone(EnterpriseOrder) theConceptualSr;
            unassigned forone(Telephone) thePhone;
            unassigned forone(Customer) theCustomer:
```

```
Bell Atlantic Science & Technology Brahms Compiler Version 1.04.001
Copyright (c) Bell Atlantic Science & Technology, Inc. 1997. All rights reserved.

0 error(s), 0 warning(s)
Exporting model in G2 Interface Format to file D:\Brahms\Projects\Emt\EMT_TrunkSupervisor.g2
```

Figure 3. Brahms development environment

Simulation

This section discusses the simulation of the model described in the previous sections. When the model builder has been able to parse the model without errors and/or warnings, the parser creates an export file in a G2® interface format. This file is generated specifically for the simulation engine, currently developed within the G2® environment[5].

[5] G2® is a registered trademark of Gensym Corporation. We are currently using G2 version 4.0

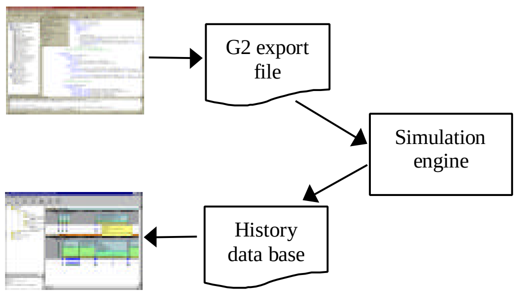


Figure 4. Simulation cycle

This export file is loaded into the simulation environment. The simulation engine runs the simulation and saves the history data while running the model. When the simulation run is stopped, the history data can be saved in a Microsoft® Access™ database. The history database can be viewed in the Brahms AgentViewer. Figure 5 shows the result of a simulation run of our example model in the Brahms AgentViewer.

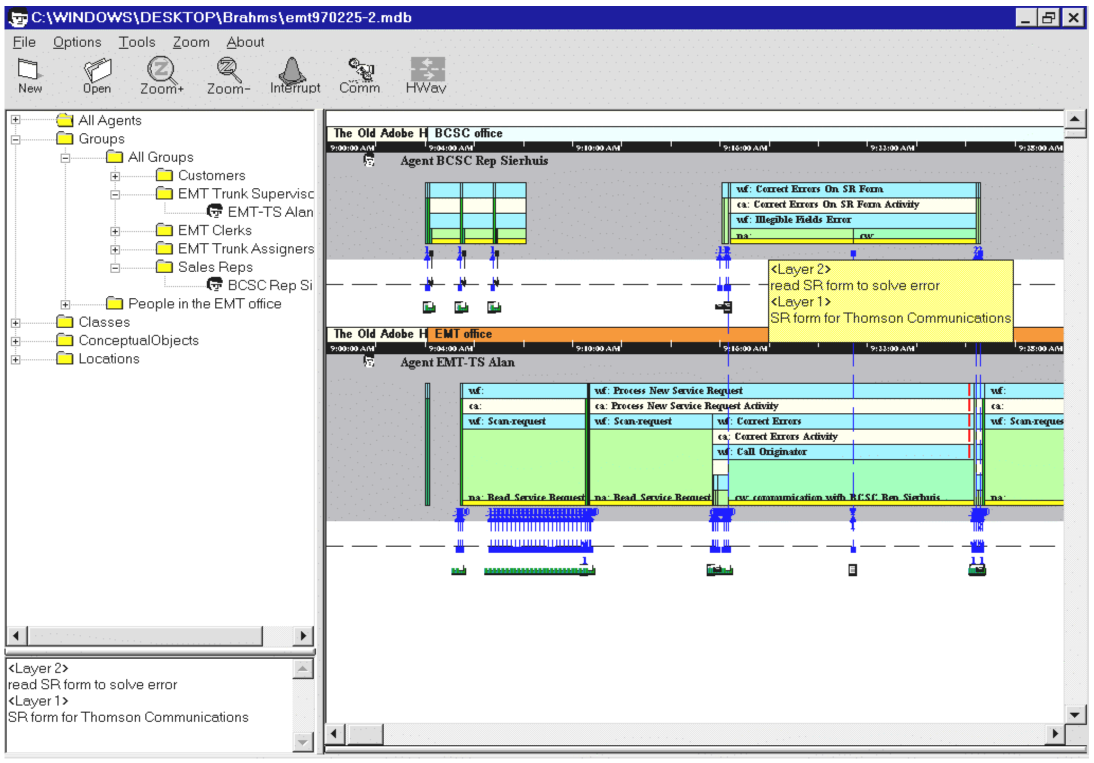

C:\WINDOWS\DESKTOP\Brahms\emt970225-2.mdb
File Options Tools Zoom About
New Open Zoom+ Zoom- Interrupt Comm HWav
All Agents
Groups
All Groups
Customers
EMT Trunk Supervisc
EMT-TS Alan
EMT Clerks
EMT Trunk Assigners
Sales Reps
BCSC Rep Si
People in the EMT office
Classes
ConceptualObjects
Locations
The Old Adobe H BCSC office
Agent BCSC Rep Sierhuis
wf: Correct Errors On SR Form
ca: Correct Errors On SR Form Activity
wf: Illegible Fields Error
The Old Adobe H EMT office
Agent EMT-TS Alan
wf: Process New Service Request
ca: Process New Service Request Activity
wf: Scan-request
wf: Correct Errors
ca: Correct Errors Activity
wf: Call Originator
pa: Read Service Request
cw: communication with BCSC Rep Sierhuis
<Layer 2>
read SR form to solve error
<Layer 1>
SR form for Thomson Communications

Figure 5. Agent view generated from history data base

The view displays a workframe/activity time line for the EMT Trunk Supervisor and the BCSC Sales Rep. Figure 5 shows that at 9:00 AM both agents are going to work, traveling from home (the Old Adobe Hut) to their offices. At work, the Sales Rep (top agent) starts faxing the service request orders. You can see the agent interacting with his fax machine (the fax machine is shown as an icon below the agent's activity area). A little after nine, you see the Trunk Supervisor (bottom agent) picking up the first fax from his fax machine, scanning the fax for errors. You can see a lot of interactions between the Trunk Supervisor agent and his fax machine. These interactions are actually the fax machine broadcasting a "belief" that a new fax has arrived. Because the Trunk Supervisor is already engaged in an activity, these broadcasts get ignored, until the first fax is scanned and apparently has no errors on it. At that moment the Trunk Supervisor agent reacts to the fax machine's broadcast, and goes back to the fax machine to pick up the newly arrived faxes (in the mean time, as you can see in the Sales Rep's part, two new faxes have been faxed). It shows the Trunk Supervisor scanning the second fax, and processing it. Around 9:15AM the Trunk Supervisor finds an error on the second fax, and calls the Sales Rep. The first part shows the Trunk supervisor calling the Sales Rep. This is where the two agents are collaborating over the phone, solving the error on the fax. The (blue) dotted lines between the two agents show the conversation (i.e. the transfer of beliefs) between the two agents[6]. When the agents have solved the error, you can see them hanging up their telephones. The Brahms AgentViewer allows us to investigate the model in detail by zooming in or out on the time line, and requesting beliefs an agents holds at a given point in time. The user can also request detailed information about the firing of a specific workframe, showing the variable bindings, what beliefs the agent holds that fire the workframe. If the workframe was stopped by the detection of a fact, we can jump to the place where the fact was created. We can also query the database for the existence of certain beliefs or facts for agents or objects. The object communication line shows the interaction (in belief transfers or create-object activities) between agents and objects. Many other useful features are not described in this paper.

Intelligent software agents

The Brahms technology can be used to build an intelligent software agent. An intelligent agent can function anywhere on the continuum between an assistant and a coach, generally for a particular person, a human user, but it may also serve a robot, or a computer-implemented process. At one end of the continuum, an assistant tries to help by doing work; at the other end, a coach provides advice or comment. The intelligent agent is built by extending conventional ITS (intelligent tutoring system) technology to use Brahms models for modeling the human user and as the source of expert knowledge (Clancey 1987). Among the advantages of using Brahms, models in this way are the following. A Brahms model models activities and not simply problem solving through reasoning. A Brahms agent is modeled as interacting in a world and exploiting social

[6] The small rectangle in the middle of Figure 5, to the right, should be ignored. It shows a tool-tip that pops up when the user moves the cursor over a layered region of the screen. In this case it shows the activity where the sales rep is reading the service request form to solve

knowledge in interacting with other Brahms agents distributed in a modeled geography. By integrating the world model and the Brahms model with each other and with real objects and sources of information, a Brahms agent modeling a user is able to receive facts the user would be interested in (as modeled by the agent's workframes and thoughtframes); and an intelligent agent observing these conditions could then provide the information to the user.

The usefulness of this may be seen by considering an intelligent agent tied to an orders database. The intelligent agent runs the Brahms simulation to develop and maintain a world model (of objects, their states and locations) and models of workers (beliefs, activities, and locations) who process orders. The intelligent agent advises an order coordinator by electronic mail when an urgent order has arrived. This is done by applying the order coordinator's workframes to the new order (detected on the real order database) to see what actions the coordinator would take if he saw the order now. Further, having a model of the coordinator, the intelligent agent knows where to try to reach him or who might know where he is. In this way, the intelligent agent can capitalize on the representation of social relationships.

An intelligent agent, acting as a coach, could consult a Brahms model of an experienced worker in a user's job. With a PDA (personal digital assistant) linked to the coach (intelligent agent), the user can receive coaching as he moves about performing his job. The user could use the PDA to get suggestions on sources of information, job priorities, procedures, people to talk to for advice, and so on. In another application, such a coaching intelligent agent could be embedded in a training simulation where the user plays the role he is learning.

An intelligent agent can be programmed as a Brahms model or it can be programmed in a different language. Programming an intelligent agent as a Brahms model allows the intelligent agent to be modeled in terms of its activities (what the intelligent agent does when). Thus, in designing an information technology such as a PDA, Brahms provides a tool to model a context of use in which the new technology is integrated in a model of practice that includes intelligent agents.

Conclusions

In this paper we described the Brahms agent-oriented language developed for modeling and simulating human behavior and work practices. With our focus on simulating human work behavior, we have developed an AOL that allows us to represent dependent and independent (i.e. emergent) coordination between (human) agents. We have shown, through a real-life example of two agents collaborating in solving errors on service requests, that the Brahms language can model agent collaboration, and communication through coordinated activities.

---

the error.

Currently, the Brahms environment is being used to model and simulate work practices of human organizations. However, in future research we plan to develop the Brahms language as a programming language for intelligent distributed software agents. By modeling a person's work activities we could implement intelligent personal digital agents (PDAs) that can collaborate with the user and with other distributed software agents. Such a programming language will allow us to develop human centered intelligent software agents.

Acknowledgements

Brahms was conceived and developed by Bill Clancey (formerly IRL), Dave Torok (Aston Brook, formerly NYNEX Science & Technology), Maarten Sierhuis (formerly NYNEX Science & Technology), Ron van Hoof (formerly NYNEX Science & Technology) in 1992-97. Our understanding of work systems design and social theories was developed through many long discussions with Patricia Sachs (IRL, formerly NYNEX Science & Technology), Gitti Jordan (Xerox-Parc, formerly IRL), David Moore (NYU, formerly NYNEX Science & Technology), Maddeline Ritter (formerly NYNEX Science & Technology), and Jim Euchner (Vice President Information Systems at Bell Atlantic, formerly NYNEX Science & Technology). Technical designs were contributed by Erik Vinkhuyzen (RIACS, formerly IRL), Alwin Bakkenes (Netscape, formerly intern at NYNEX Science & Technology), and Edgar Steenvoorden (formerly intern at NYNEX Science & Technology).

Brahms was funded by NYNEX Science & Technology, White Plains, NY (now part of Bell Atlantic Information Systems), in collaboration with the Institute for Research on Learning, Menlo Park, CA

The name “Brahms” stands for “Business Redesign Agent-based Holistic Modeling System.” Patent Pending.

References

Brooks, R., A. (1991). “Intelligence without representation.” Artificial Intelligence **47**: 139-159.

Carley, K. M. and M. J. Prietula (1994). ACTS Theory: Extending the Model of Bounded Rationality. Computational Organization Theory. K. M. Carley and M. J. Prietula. Hillsdale, NJ, Lawrence Erlbaum Associates.

Charniak, E. and D. McDermott (1986). Introduction to Artificial Intelligence. Reading, MA, Addison-Wesley Publishing Company.

Checkland, P. and J. Scholes (1990). Soft Systems Methodology in Action. Chicester, England., John Wiley & Sons Ltd.

Clancey, W. J. (1987). Knowledge-Based Tutoring. Cambridge, MA, The MIT Press.

Clancey, W. J. (1997). The Conceptual Nature of Knowledge, Situations, and Activity. Human and Machine Expertise in Context. P. Feltovich, R. Hoffman and K. Ford. Menlo Park, CA, The AAAI Press**:** 247-291.

Clancey, W. J., P. Sachs, et al. (1996). Brahms: Simulating practice for work systems design. Pacific Knowledge Acquisition Workshop, Sydney, Australia.

Laird, J. E., A. Newell, et al. (1987). “SOAR: An architecture for general intelligence.” Artificial Intelligence **33**: 1-64.

Newell, A. (1990). Unified theories of cognition. Cambridge, MA, Harvard University Press.

Newell, A. and H. A. Simon (1972). Human Problem Solving. Englewood Cliffs, NJ., Prentice-Hall.

Nowak, A. and B. Latané (1994). Social dilemmas exist in space. Social dilemmas and cooperation. N. Schultz, W. Albers and R. N. Mueller. Heidelberg, Springer-Verlag.

Sachs, P. (1995). “Transforming Work: Collaboration, Learning and Design.” Communications of the ACM **Vol. 38, No. 9**(September 1995): 36-44.

Schreiber, G., B. Wielinga, et al., Eds. (1993). KADS: A Principled Approach to Knowledge-Based System Development, Academic Press.

Steenvoorden, E. C. (1995). The geography of WorkFrame. Utrecht, The Netherlands, Centre of Excellence CIBIT.

Thomas, B. (1993). PLACA, An Agent-Oriented Language. Stanford, CA, Department of Computer Science, Stanford University.

Torrance, M. (1991). The AGENT0 Programming Manual. Stanford, CA, Department of Computer Science, Stanford University.

Wenger, E. (1997). Communities of Practice; Learning, meaning, and identity, Cambridge University Press.

Wooldridge, M. (1992). The logical Modellling of Computational Multi-Agent Systems. Manchester, UK, Department of Computation, UMIST.

Wooldridge, M. and N. R. Jennings (1995). “Intelligent Agents: Theory and Practice.” Knowledge Engineering Review.