Multi-Agent System Simulation Framework *

Bryan Horling, Victor Lesser, Régis Vincent
Department of Computer Science
University of Massachusetts, Amherst, MA, 01003
Email: bhorling,lesser,vincent@cs.umass.edu, Phone: +1 413 545 0675

Abstract

The Multi-Agent System Simulator (MASS) is a discrete, event-based multi-agent simulator, providing environment, messaging, execution and sensor services. The Intelligent Home domain environment is also covered, as an example of how this system may be used in practice.

Key words: Multi-Agent System, Agent Simulation.

AMS subject classifications: .

1 Motivations

The motivation for the Multi-Agent System Simulator (MASS) is based on two simple but conflicting objectives. The first concerns our ability to accurately measure the influence of different multi-agent coordination strategies in an unpredictable environment. It is similarly difficult to realistically model adaptive behavior in multi-agent systems within a static environment. These two seemingly contradictory goals lie at the heart of the design of MASS.

A significant advantage multi-agent systems (MAS) have over traditional designs is the fact that the system is distributed. The decentralized, partially autonomous and redundant, nature of such a system makes them less sensitive to certain classes of faults or attacks. This same decentralization, however, also makes it difficult to analyze these systems. How do we recognize the reactions, adaptations or failures a MAS exhibits when a fault occurs? How can these capabilities be tracked over time? How hard will it be to instantiate a domain to test such characteristics?

In this paper, we will address the issues we have encountered designing a suitable environmental space, and the agents to run in it, for evaluating the coordination and adaptive qualities of multi-agent systems. In the following sections, our evaluation requirements and the MASS simulation environment will be described. As an example, we present The Intelligent Home (IHome) domain testbed, which makes use of MASS, to describe how these concepts work in practice. We will conclude with a brief overview of the future directions of this project.

2 Evaluation Environment

Numerous problems arise when systematic analysis of different algorithms and techniques needs to be performed. If one works with a real-world MAS, is it possible to know for certain that the runtime environment is identical

*Effort sponsored by the Defense Advanced Research Projects Agency (DARPA) and Air Force Research Laboratory Air Force Materiel Command, USAF, under agreement number F30602-97-1-0249 and by the National Science Foundation under Grant number IIS-9812755 and number IRI-9523419. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. Disclaimer: The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Defense Advanced Research Projects Agency (DARPA), Air Force Research Laboratory, National Science Foundation, or the U.S. Government.

from one run to the next? Can one know that a failure occurs at exactly the same time in two different runs when comparing system behavior? Can it be guaranteed that inter-agent message traffic will not be delayed, corrupted, or non-deterministically interleaved by network events external to the scenario?

If one works within a simulated environment, how can it be known that the system being tested will react optimally a majority of the time? How many different scenarios can be attempted? Is the number is large enough to be representative?

Based on these observations, we have tried to design an environment that allows us to directly control the baseline simulated environment (e.g. be deterministic from one run to the next) while permitting the addition of "deterministically random" events that can affect the environment throughout the run. This enables the determinism required for accurate coordination strategy comparisons without sacrificing the capricious qualities needed to fully test adaptability in an environment.

Hanks et al. define in [3] several characteristics that multi-agent system simulators should have:

- **Exogenous events**, these allow exogenous or unplanned events to occur during simulation.
- **Real-world complexity** is needed to have a realistic simulation. If possible, the simulated world should react in accordance with measures made in the real world. Simulated network behavior, for instance, may be based on actual network performance measures.
- **Quality and cost of sensing and effecting** needs to be explicitly represented in the test-bed to accurately model imperfect sensors and activators. A good simulator should have a clear interface allowing agents to "sense" the world.
- **Measures of plan quality** are used by agents to determine if they are going to achieve their goal, but should not be of direct concern to the simulator.
- **Multiple agents** must be present to simulate inter-agent dependencies, interactions and communication. A simulator allowing multiple agents increases both its complexity and usefulness by adding the ability to model other scenarios, such as faulty communications or misunderstanding between agents, delay in message transfer.
- **A clean interface** is at the heart of every good simulator. We go further than this by claiming that the agents and simulator should run in separate processes. The communication between agents and simulator should not make any assumptions based on local configurations, such as shared memory or file systems.
- **A well defined model of time** is necessary for a deterministic simulator. Each occurring event can be contained by one or more points in time in the simulation, which may be unrelated to real-world time.
- **Supporting experimentation** should be performed to stress the agents on in multiples classes of scenarios. We will also add **deterministic experimentation** as another very important feature of a simulator. To accurately compare separate runs, we must be sure that the experimental parameters are those which produce different outcomes.

One other characteristic, somewhat uncommon in simulation environments, is the ability to have agents perform a mixture of both real and simulated activities. For instance, an agent could use the simulation environment to perform some of its actions, while actually performing others. Executable methods, sensor utilization, spatial constraints and even physical manifestations fall into this category of activities which an agent might actually perform or have simulated as needed. An environment offering this hybrid existence offers two important advantages: more realistic working conditions and results, and a clear path towards migrating work from the laboratory to the real world. We will revisit how this can be implemented in later sections.

3 Multi Agent Survivability Simulator

MASS is a more advanced incarnation of the TÆMS simulator created by Decker and Lesser in 1993. The new MASS simulator is completely-domain independent; all domain knowledge is obtained either from configuration files or data received from agents working in the environment. Agents running in the MASS environment use TÆMS [5], a domain-independent, hierarchical representation of an agent's goals and capabilities (see Figure 1), to represent their knowledge, but they are clearly decoupled from the simulator, which provides a more accurate model of how they would function under real conditions. This TÆMS model is also used to supply the internal domain knowledge of the simulation controller. These facts, coupled with a simple configuration mechanism and robust logging tools, make MASS a good platform for rapid prototyping and evaluation of multi-agent systems.

Figure 2 shows the overall design of MASS, and, at a high level, how it interacts with the agents connected to it. On initialization, MASS reads in its configuration, which defines the logging parameters, random seed, scripts (if any) and

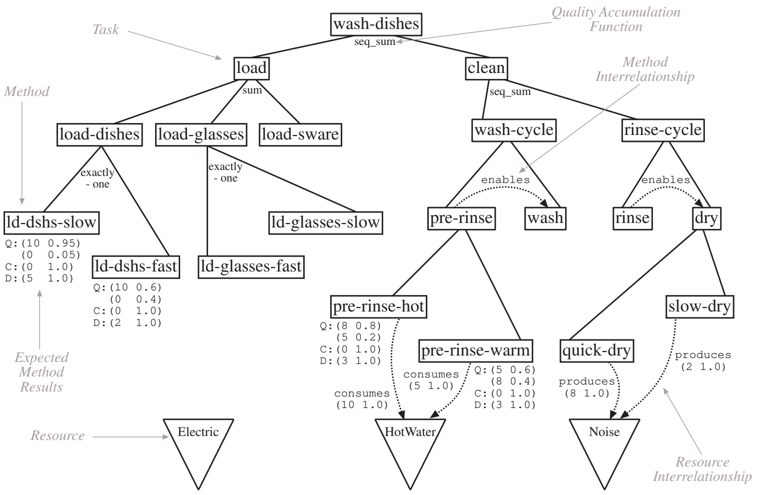

Figure 1: TÆMS task structure for the IHome Dishwasher agent

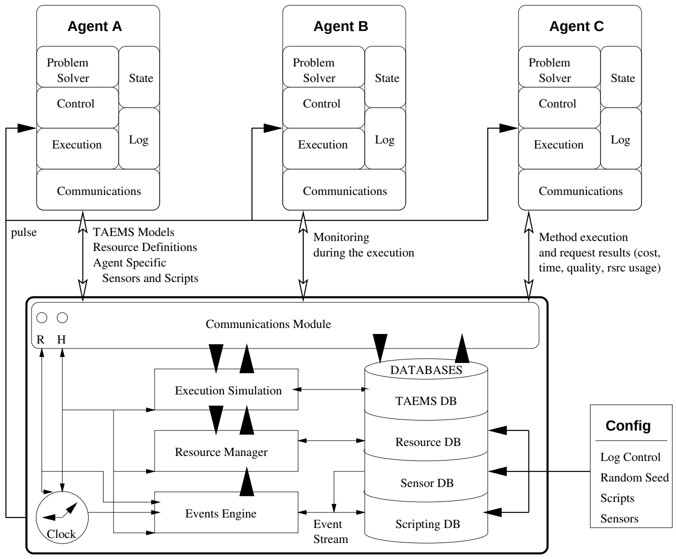

Figure 2: Architecture of the MASS and agent systems.

global sensor definitions. These are used to instantiate various sub-systems and databases. When connected, agents will send additional information to be incorporated into this configuration, which allows environmental characteristics specific to the agent to be bundled with that agent. Arguably the most important piece of data arising from the agents is their TÆMS task structures, which are assimilated into the TÆMS database shown in the figure. This database will be used by the execution subsystem during simulation to quantify both the characteristics of method execution and the effects interactions with resources and other actions have on that method. The resource manager is responsible for tracking the state of all resources in the environment, and the event engine manages the queue of events which represent tangible actions that are taking place. The last component shown here, the communications module, maintains a TCP stream connection with each agent, and is responsible for routing the different kinds of messages between each the agent and their correct destination within the controller.

The MASS controller has several tasks to perform while managing simulation. These include routing message traffic to the correct destination, providing hooks allowing agents to sense the virtual environment and managing the different resources utilized by the agents. Its primary role, however, is to simulate the execution of methods requested by the agents. Each agent has a partial view of the environment, typically describing its local view of a goal and possible solutions, which determines the expected values resulting from such an execution. This view of the world local to the agent is known as its *subjective* view. The simulator has its own view of the world (the "correct" one, which we call the *objective* view) which it uses to compute the results of the execution. Engineering differences between the subjective and objective views allow for a wide range of scenarios to be simulated.

We will give here a summary of TÆMS and how it is used to simulate an execution, the reader should refer to [5, 1] for more details. In TÆMS each method is described along three dimensions: cost, quality and duration, each of which is described with a discrete probability distribution. The quality represents any method value or characteristic that should be maximized, the cost is the dimension you want to minimize and duration provides a enabling mechanism for scheduling, coordination and deadlines. The simulator uses the distributions in its objective view when computing the values for a method execution. Note also that this probabilistic distribution offers the average case outcomes, the simulator will degrade or improve results as necessary if, for instance, required resources are not available, or other actions in the environment enable or facilitate the method's execution in some way.

The second responsibility consuming a large portion of the simulator's attention is to act as a message router for the agents. The agents send and receive their messages via the simulator, which allows the simulation designer to model adverse network conditions through unpredictable delays and transfer failures. This routing also plays an important role in the environment's general determinism, as it permits control over the order of message receipt from one run to the next. Section 3.1.2 will describe this mechanism in more detail.

3.1 Controllable Simulation

In our simulated experiments, our goal is to compare the behavior of different algorithms in the same environment under the same conditions. To correctly replicate running conditions at some later time, the simulation should have its own notion of time, its own notion of random and its own notion of events. Two simulation techniques exist which we have exploited to achieve this behavior: discrete time and events. Discrete time simulation segments the environmental time line into a number of slices. In this model, the simulator begins a time slice by sending a pulse to all of the actors involved, which allows them to run for a period of (real) CPU time. In our model, a pulse does not represent a predefined quantity CPU time; each agent decides independently when to stop running; this allows agent performance to remain independent of the hardware it runs on. The second type of simulation is event based, which means that the control is directed by events that force agents to react. The MASS simulator combines these by dividing time into a number of slices, during which events are used to internally represent actions and interact with the agents. In this model, agents then execute within discrete time slices, but are also notified of activity (method execution, message delivery, etc.) through event notification.

In the next section we will discuss discrete time simulation and the benefits that arise from using it. We will then describe the need for an event based simulation within a multi-agent environment.

3.1.1 Discrete time simulation

Because MASS utilizes a discrete notion of time, all agents running in the environment must be synchronized with the simulator's time. To enable this synchronization, the simulator begins each time slice by sending each agent a "pulse" message. This pulse tells the agent it can resume local execution, so in a sense the agent functions by transforming the pulse to some amount of real CPU time on its local processor. This local activity can take an arbitrary amount of real time, up to several minutes if the action involves complex planning, but with respect to the simulator, and in the perceptions of other agents, it will take only one pulse. This technique has several advantages:

1. A series of actions will always require the same number of pulses, and thus will always be performed in the same amount of simulation time. The number of pulses is completely independent of where the action takes place, so performance will be independent of processor speed, available memory, competing processes, etc...

2. Events and execution requests will always take place at the same time. Note that this technique does not guarantee the ordering of these events within the time slice, which will be discussed later in this section.

Using this technique, we are able to control and reproduce the simulation to the granularity of the time pulse. Within the span of a single pulse however, many events may occur, the ordering of which can affect simulation results. Messages exchanged by agents arrive at the simulator and are converted to events to facilitate control over how they are routed to their final destination. Just about everything coming from the agents, in fact, is converted to events; in the next section we will discuss how this is implemented and the advantages of using such a method.

3.1.2 Event based simulation

Events within our simulation environment are defined as actions which have a specific starting time and duration, and may be incrementally realized and inspected (with respect to our deterministic time line, of course). Note that this is different from the notion of event as it is traditionally known in the simulation community, and is separate from the notion of the "event streams" which are used internally to the agents in our environment.

All of the message traffic in the simulation environment is routed through the simulator, where it is instantiated as a message event. Similarly, execution results, resource modifiers or scripted actions are also represented as events within the simulation controller. We attempt to represent all activities as events both for consistency reasons and because of the ease with which such a representation can be monitored and controlled.

The most important classes of events in the simulator are the *execution* and *message* events. An *execution* event is created each time an agent uses the simulator to model a method's execution. As with all events, execution events will define the method's start time, usually immediately, and duration, which depends on the method's probabilistic distribution as specified in the objective TÆMS task structure (see section 1). The execution event will also calculate the other qualities associated with a method's execution, such as its cost, quality and resource usage. After being created, the execution event is inserted into the simulator's time based event queue, where it will be represented in each of the time slots during which it exists. At the point of insertion, the simulator has computed, but not assigned, the expected final quality, cost, duration and resource usage for the method's execution. These characteristics will be accrued (or reduced) incrementally as the action is performed, as long as no other events perturbate the system. Such perturbations can occur during the execution when forces outside of the method affect its outcome, such as a limiting resource or interaction with another execution method. For example, if during this method's execution, another executing method overloads a resource required by the first execution, the performance of the first will be degraded. The simulator models this interaction by creating a limiting event, which can change one or more of the performance vectors of the execution (cost, quality, duration) as needed. The exact representation of this change is also defined in the simulator's objective TÆMS structure.

Real activities may be also incorporated into the MASS environment by allowing agents to notify the controller when it has performed some activity. Using this mechanism, an agent may execute some method locally, generating actual results in the process. When completed, it then reports these results to the simulator, which updates its local view accordingly to maintain a consistent state. Agents may then be incrementally generated to meet real world requirements by adding real capabilities piecemeal, and using MASS to simulate the rest.

The other important class of event is the message event, which is used to model the network traffic which occurs between agents. Instead of communicating directly between themselves, when a message needs to be sent from one agent to another (or to the group), it is routed through the simulator. The event's lifetime in the simulation event queue represents the travel time the message would use if it were sent directly, so by controlling the duration of the event it is possible to model different network conditions. More interesting network behavior can be modeled by corrupting or dropping the contents of the message event. Like execution events, the message event may also may be influenced by other events in the system, so a large number of co-occurring message events might cause one another to be delayed or lost.

To prevent non-deterministic behavior and race conditions in our simulation environment, we utilize a kind of "controlled randomness" to order the realization of events within a given time pulse. When all of the agents have completed their pulse activity (e.g. they have successfully acknowledged the pulse message), the simulator can work with the accumulated events for that time slot. The simulator begins this process by generating a a unique number or hash key for each event in the time slot. It uses these keys to sort the events into an ordered list. It then deterministically shuffles this list before working through it, realizing each event in turn. This shuffling technique, coupled with control over the random function's initial seed, forces the events to be processed in the same order during subsequent runs without unfairly weighting a certain class of events (as would take place if we simply processed the sorted list). This makes our simulation completely deterministic, without sacrificing the unpredictable nature a real world environment would have.

3.2 Sensor Simulation

To provide the sensing capabilities, we have defined a mechanism which allows agents to read environmental information from the simulated world via the controller. Our goal was to make the interface generic, so agents could actively gather information on subjects ranging from the current time to the current noise level in a room to the location of another agent in the system. Within the simulator one or more sensor objects can be created, each of which can access a particular environmental characteristic with some amount of precision.

Multiple sensor objects may be defined to access the same environmental characteristic, but with different precision levels and agent access privileges. This allows us to model different sensing capabilities and damaged or faulty sensors, and explore scenarios involving redundant and/or conflicting sensor results.

As an example, suppose the dishwasher is able to sense the current noise level, but the simulator will modify the sensed level by up to 5% of its real value. Meanwhile, the vacuum agent will read the value with perfect precision. So if the current level is 76 Db, the dishwasher can receive a reading anywhere from 72.2 to 79.8, while vacuum will always know the correct value of 76.

4 Applications and Conclusions

The MASS simulation environment was built to permit rapid modeling and testing of the adaptive behavior of agents with regard to coordination, detection, diagnosis and repair mechanisms functioning in a mercurial environment. The primary purpose of the simulator is to allow successive tests using the same working conditions, which enables us to use the final results as a reasonable basis for the comparison of competing adaptive techniques. This section will briefly describe such an environment, IHome, which has been modeled with MASS.

The domain of our first environment is an Intelligent Home [6], in which several appliances are represented as agents, possessing self-knowledge, goals, environmental sensing capabilities and communicative behavior. The resources in IHome are very limited, so agents must decide if coordination is necessary to achieve their individual goals. The objective of this work is to design an environment where we are able to study different kinds of coordination techniques, and evaluate the adaptative capability of our agents to a changing or constrained world. This work is similar to the Adaptive House in Boulder, Colorado [7, 8] and [2, 4]. In IHome, MASS was used to provide the environment and simulate method execution for twelve agents, ranging from diswashers to mobile robots. Within MASS, we were able to implement and evaluate the performance of several coordination protocols, and is currently being used to test diagnosis based adaptation and multi-linked contract protocols.

We feel the main advantages of this framework are its domain independence, ease of use and flexible architecture. Our efforts to retain determinism without sacrificing unpredictability also make it well suited for algorithm generation and analysis.

References

[1] Keith Decker and Victor Lesser. Quantitative modeling of complex environments. Technical report, Computer Science Department, University of Massachusetts, 1993. Technical Report 93-21.

[2] Werner Dilger. A society of self organizing agent in the intelligent home. Technical report, Technical Report SS-98-02 Standford, California, Menlo Park, March 1998.

[3] Martha E. Pollack Hanks, Steven and Paul R. Cohen. Benchmarks, testbeds, controlled experimentation, and the design of agent architectures. *AI Magazine*, 14(4):pp. 17–42, 1993. Winter issue.

[4] B. Hurberman and Clearwater S. A multi agent system for controlling building environment. In *The First International Conference on Multi-Agent Systems (ICMAS-95)*, 1995.

[5] Multi-Agent Systems Lab. The taems white paper. http://mas.cs.umass.edu/research/taems/white/, 1999.

[6] Victor Lesser, Michael Atighetchi, Bryan Horling, Brett Benyo, Anita Raja, Regis Vincent, Thomas Wagner, Ping Xuan, and Shelley XQ. Zhang. A Multi-Agent System for Intelligent Environment Control. In *Proceedings of the Third International Conference on Autonomous Agents*, Seattle,WA, USA, May 1999. ACM Press.

[7] Michael C. Mozer. The adaptive house, boulder, colorado. Web page, 1998. http://www.cs.colorado.edu/ mozer/nnh/index.html.

[8] Michael C. Mozer. The neural network house: An environment that adapts to its inhabitants. In M. Coen, editor, *Proceedings of the American Association for Artificial Intelligence Spring Symposium on Intelligent Environments*, pages 110–114, Menlo, Park, CA, 1998. AAAI Press.