The Intelligent Home Testbed *

Victor Lesser, Michael Atighetchi, Brett Benyo, Bryan Horling,
Anita Raja, Régis Vincent,
Thomas Wagner, Ping Xuan and Shelley XQ. Zhang

Computer Science Department
University of Massachusetts at Amherst
Amherst, MA 01003
lesser@cs.umass.edu

Abstract

Intelligent environments are an interesting development and research application problem for multi-agent systems. The functional and spatial distribution of tasks naturally lends itself to a multi-agent model and the existence of shared resources creates interactions over which the agents must coordinate. In the UMASS Intelligent Home project we have designed and implemented a set of distributed autonomous home control agents and deployed them in a simulated home environment. Our focus is primarily on resource coordination, though this project has multiple goals and areas of exploration ranging from the intellectual evaluation of the application as a general MAS testbed to the practical evaluation of our agent building and simulation tools.

1 Introduction

The intelligent home project (IHome) at the UMASS multi-agent systems lab is an exploration in the application of multi-agent systems technology to the problem of managing an intelligent environment. We have implemented a sophisticated simulated home environment, populated it with distributed intelligent home-control agents (including simulated robots) that control appliances and negotiate over shared resources, and begun experimentation with different coordination protocols and agent adaptability / responsiveness to changing environmental conditions.

Our work is akin to the Adaptive House [14] and [5, 8] in that the objective is for the environment to automate some of the tasks currently performed by humans – possibly with improvements in efficiency or quality of service. However, our focus is on resource coordination and temporally sequencing agent activities over shared resources. A broad spectrum of research falls into the general category of intelligent environments. For example, one class of work deals with collecting and integrating information about the activities that occur within the environment [1] while another class focuses on identifying and tracking humans as they move about the environment [2, 15].

The UMASS simulated IHome environment is controlled by intelligent agents that are associated with particular appliances; a snapshot of a sample run is shown in Figure 1. The IHome population set includes agents like an intelligent WaterHeater, CoffeeMaker, Heater, A/C, DishWasher, etc., and a robot for fetching items and moving physical goods from one location to another. The home agents reason about their assigned tasks and select candidate actions based on the occupant's preferences and the availability of resources. For example, if hot water is scarce, the DishWasher agent may elect to run a cold cycle, trading-off solution quality for resource consumption – the agent may also elect to wait until hot water becomes available. Agents coordinate over shared resources like noise, electricity, temperature, and hot water. Resources, resource interactions, task interactions, and the performance characteristics of primitive actions are all represented and quantified in the TÆMS [4] task modeling framework. This enables agents to reason about the trade-offs of different possible courses of action and to adapt behaviorally to the changing environment.

The research has several goals, among them are:

1. Examine the intelligent home domain as a general application testbed for research in multi-agent systems.
2. Apply the TÆMS [4] domain-independent task modeling framework to a new domain and evaluate its use in the rapid development of a new multi-agent application.
3. Test and refine our multi-agent simulation environment [17] that controls method execution and communication characteristics for a set of distributed agents. The environment employs a complex time mapping scheme and a process controller to resolve timing issues between the distributed agents and to ensure reproducibility.
4. Test and refine our java-based generic agent construction framework [6] that facilitates agent construction through an event-driven component architecture. The framework also enables agents to be decoupled from the simulator and executed in their application domain with a simple change in internal components, i.e., with a change to the makefile.

Space precludes discussing all of these points. We will focus on the challenges offered by the application domain, discuss the TÆMS modeling framework from a high-level view, describe the application and some of the agents, touch on the simulator we have used for the IHome project, and present experimental results. For information about the agent development tools or more information about the project, including screen snapshots, readers should consult the group web pages [16].

* 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.

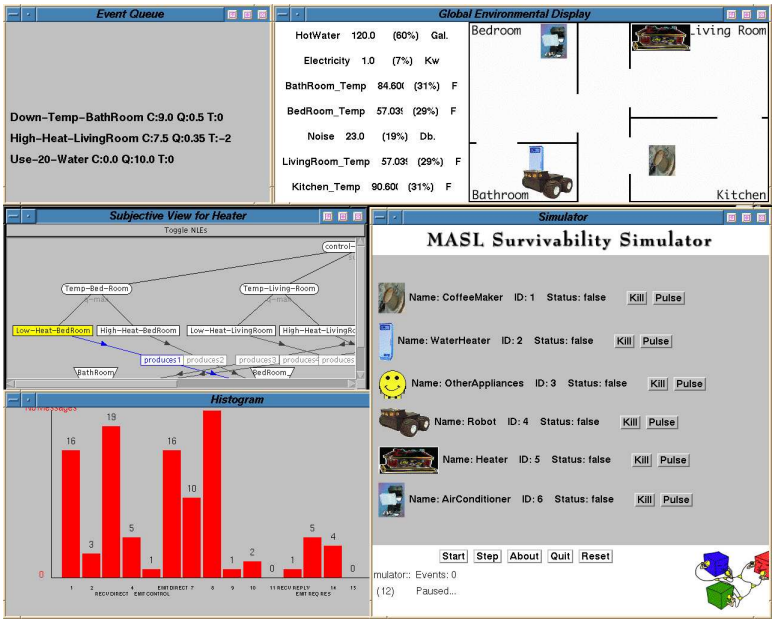

Figure 1: IHome Agents in Action

2 TÆMS Task Structures in the Intelligent Home

The simulator and the agents in our intelligent home model problem solving activities using the TÆMS domain independent task modeling framework [4, 11]. TÆMS models planned actions, candidate activities, and alternative solution paths from a quantified perspective; all primitive actions are described statistically via discrete probability distributions in terms of quality, cost, and duration. A fourth dimension, uncertainty, is implicit in the probability distributions. Thus, TÆMS-based reasoners (e.g., [18]) can evaluate the quality, cost, and duration (and uncertainties in each of these) characteristics of each possible course of action and select the course of action that best[1] meets the current constraints and environmental conditions. For example, in a time constrained situation, an agent may sacrifice solution quality and possibly consume more resources to produce a result within the specified deadline.

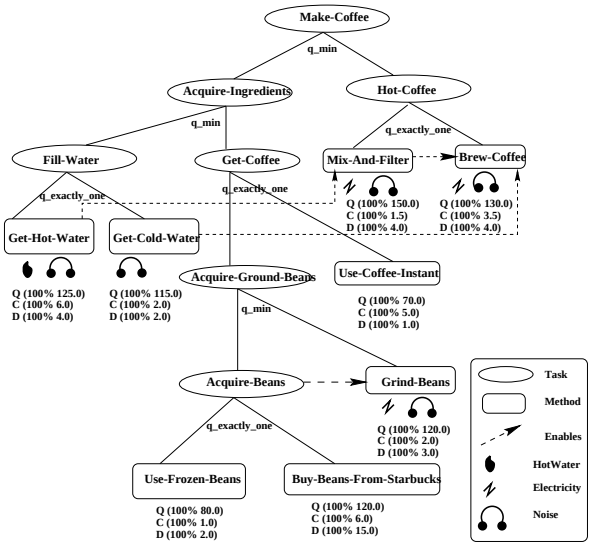

Figure 2: TÆMS Task Structure for Making Coffee

Figure 2 shows a TÆMS task structure used by the CoffeeMaker agent to represent its process for making coffee. The simulator has a different *objective* view of the agent's *subjective* task structure and the views may be radically different – this enables experimentation with situations in which the agent's model is inaccurate. While TÆMS is a modeling framework, agents often use it internally to reason about the structure of their computation. In this case, it is akin to a process plan or other meta representations. It is a structure that describes how the primitive actions relate to accomplishing the overall objective and often the primitive actions in TÆMS correspond directly to underlying code. TÆMS task structures are models in the sense that they may be abstracted from some of the execution details, not in the sense that they are completely isolated from execution and can only be used in a simulated environment. In the IHome project, programmers describe the agents problem solving options in TÆMS, usually via a TÆMS graph-grammar-generator, and then build the tools, or use existing ones, for reasoning with the task structures. In this usage, the programmers take the place of a generative planner or problem solver that would normally produce the task structures (as in [12]) from its own internal representations. This enables programmers to rapidly create agents for applications where an off-the-shelf planner/problem solver is not available.

The task structure shown in Figure 2 describes alternative ways

[1] Due to the combinatorics of the TÆMS scheduling problem, "best" does not necessarily denote optimal.

to obtain water, obtain coffee, and brew the coffee. Consider the *Acquire-Ground-Beans* task; it has two subtasks, one of which is another decomposable task, and another (*Grind-Beans*) which is a primitive action and is described in terms of quality, cost, and duration. The small icons under the action denote resource usage – resources and the resource interactions are absent from this figure to improve readability. The arc leading from *Acquire-Beans* to *Grind-Beans* is an *enables* non-local-effect (nle) and it denotes a hard constraint that *Acquire-Beans* must have quality in order for *Grind-Beans* to execute, i.e., the agent must have beans before it can grind them. The *q_min* quality-accumulation-function (qaf) associated with *Acquire-Ground-Beans* denotes that its quality is computed by *min(Acquire-Ground-Beans,Grind-Beans)*, modeling the notion that poor beans or poor grinding produces poor ground beans. *Acquire-Beans* has two primitive action subtasks. Note that using frozen beans produces a lower quality result than buying beans from Starbucks, but that it also costs less and is considerably faster. If the CoffeeMaker is in a hurry, or has limited financial resources, it may thus choose to use frozen beans. However, if the agent is extremely time constrained, it will probably perform *Get-Coffee* by using instant coffee rather than obtaining ground beans of either form.

This task structure illustrates the notion of quantified choice in TÆMS and its facilitation of trade-off behaviors at run-time. However, it is not a good illustration of the use of uncertainty in TÆMS as the methods all have simple distributions and no represented probability of failure. If execution failure should occur the agent will reschedule accordingly. However, the lack of any representation of failure may keep the agent from working to reduce the probability of failure by choosing more conservative options. This can be important in cases when tight deadlines exist.

The quantifications of items in TÆMS is not regarded as a perfect science. Task structure programmers or problem solver generators *estimate* the performance characteristics of primitive actions. These estimates can be refined over time through learning [9] and reasoners typically replan and reschedule when unexpected events occur. Quantification in TÆMS is not limited to the characterization of primitive actions. Interactions between tasks, actions, and resources are also described statistically. For example, agents describe their resource consumption behaviors in terms of a *consumes* non-local-effect and the effects of the resource on the task are described via a *limits* non-local-effect. The limits nle describes the negative effects of lacking sufficient resources to perform a task in terms of power-effects on quality, cost, and duration. These effects can model a range of behaviors, from an increase in duration in the case of a network resource to a complete reduction of expected quality to zero in the case of a hard resource like a locked file. For a non-consumable resource, e.g., network bandwidth, where the resource is diminished during the usage and then returned to its initial state.

3 The Intelligent Home

The intelligent home is a model of a small home constructed and executed using the generic multi-agent simulation environment [17]. The home consists of four rooms: a bedroom, a living room, a bathroom, and a kitchen, all joined by a common hallway. Though the home is more of an apartment, size is actually not necessary in this application to obtain interesting results; the interesting issues arise when agent controllers interact and a smaller space requires fewer agents to generate interesting interactions.

Expanding the size of environment may create an issue of scalability with respect to resource coordination protocols unless the expansion is achieved through composition of (primarily) independent sub-environments. If the intelligent environment were a large manufacturing factory, for instance, where hundreds of agents shared common resources like electricity and water, the simple peer-to-peer agent organization used in this project will probably lead to combinatorics and high coordination overhead. In situations such as these, the agents should be organized into work-groups or according to other partitioning schemes to reduce the scope of interaction.

In our model agents are associated with major appliances. We decided on the model of associating agents with appliances because we believe it is likely that in the future intelligent appliances will be packaged with their own intelligent control software. Different appliances will probably have different types of agent controllers and the agents will probably be heterogeneous, interfacing through a common protocol. This leads to either a peer-to-peer organization or a group-style organization where agents are perhaps clustered according to function (e.g., washer and dryer), spatial location, or resource usage. We choose the peer-to-peer approach for this initial implementation because it allows us to use the same simple protocol sets for all agents and it does not limit or reduce agent interaction. In the future, we plan to experiment with different organizational structurings.

Thus, agents are associated with major appliances and they interact directly to coordinate over shared resources. Currently, we model and coordinate over electricity, hot water, noise or sound levels, and room temperature in each of the modeled rooms. Agents coordinate using a resource coordination protocol discussed in [13].

In terms of modeling issues, we made some simplifying assumptions. Based on work ongoing in the community, we assumed the existence of supporting technology for: identification and tracking of individuals moving about the environment, obtaining client preference profiles that include things like deadlines on particular activities (e.g., dishes should be done by the time the client gets home from work), and assimilating different occupant preferences for parameters like room temperature. Since we currently employ only one fetching robot, we also did not address spatial constraint issues like two robots attempting to use the same door simultaneously (it is not clear that we will model robots at that fine a level of granularity in the future either).

The agents that populate the intelligent home are heterogeneous, each having its own internal problem solver that reasons using TÆMS task structures. Some of the agents make use of generic agent control tools like the Design-to-Criteria scheduler [18], but there is no requirement to do so as we are interested in examining the bottom-up production of agents for this application. All the agents were constructed using the generic Java Agent Framework [6], however the framework's role is to "glue" together disparate components and it does not impose any restrictions on the types of agents that can be constructed or how the agents approach particular problems. Interagent communication is done via KQML [10] routed through the simulation environment as discussed previously. The population of the intelligent home includes a mobile robot and appliance agents like the Dryer, TV, DishWasher, WaterHeater, VacuumCleaner, Heater, A/C, CoffeeMaker, and the OtherAppliances agent. The OtherAppliances agent is a place holder for other appliances not currently modeled by agents. It makes resource requests and otherwise stresses and exercises the system in much the same way as an additional x agents would. Space precludes discussing each agent in detail, though the agents are generally characterized according to the tasks they perform, the alternative ways to perform them, the resources they consume, and the agents with which they interact. For example:

A/C Agent Summary: Responsible for climate regulation. Has cooling ability, limited heating ability by routing air flow through home, and the ability to control humidity by routing air through the compressor. The agent's control flow is shown in Figure 3.

Task Performance Options Different fan and compressor levels resulting in different cooling rates with different noise characteristics.

Shared Resources **Noise:** interacts with the DishWasher, Dryer, VacuumCleaner, CoffeeMaker, and TV agents. **Electricity:** interacts with the DishWasher, Dryer, VacuumCleaner, TV, and CoffeeMaker agents. **Temperature:** interacts with the Heater agent.

Task Interactions Task sharing with the Heater agent to control room temperature.

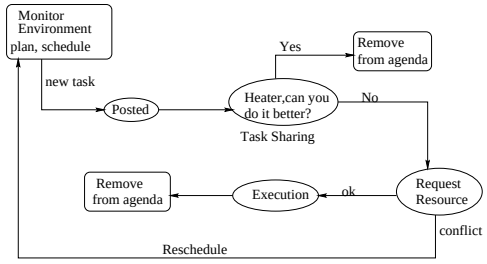

Figure 3: AirConditioner Agent's Control Flow

One of the agents in the home is actually a generic agent. It uses the Design-to-Criteria scheduler so that its behaviors are completely defined and described in TÆMS and a set of goal criteria for the scheduler. The generic agent can, in essence, become any of the other agents simply by changing its descriptive task structures and the scheduling criteria. The generic agent will not always perform identically to the agent it emulates because the agent may make different trade-off decisions than those made by the scheduler. In the future, we will compare the performance of the generic agent to the specialized agents to determine the differences and the relative strengths of each.

4 Simulation Environment

The intelligent Home project was built to test agent's resources coordination but also to test our agent framework. The framework, we build is composed of separate entities the Multi Agent Survivability Simulator (MASS) and the Java Agent Framework (JAF). MASS was used to simulate the intelligent house, to simulate agent's method execution and resources usage. JAF was used to build all the agents and is a very nice framework that let people concentrate on their agent's behaviour rather than the technical aspects of an agent (like how to deal with TCP/IP messages, parse them, etc..). In fact JAF offers for free all the technical aspects of an agent like comunication, control, state storage, TÆMS generation.

This section will explain how MASS was designed for evaluating multi agent systems.

4.1 MAS Simulator

MASS was designed to be a tool to evaluate agents and agent's coordination. When you tried to evaluate agent and multi agent systems, a lot of problems occurs. 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 truely adaptive behavior in multi-agent systems within a static environment. These two seemingly contradictory goals lie at the heart of the design of the Mutli-Agent Survivability Simulator.

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 anaylze these systems. How do we recognize the reaction of a MAS when a fault occurs? How does one measure its adaptative capabilities?

If you evaluate a real-world MAS, is it possible to know for certain that the runtime environment is identical 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 garunteed that inter-agent message traffic will not be delayed or corrupted by network events external to the scenario?

If you evaluate a MAS system in 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 have been 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 environement (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 in an environment to fully test adaptability.

MASS[13] is the next generation of the TÆMS simulator created by Decker and Lesser in 1993. Agents running in the MASS environment still use TÆMS a hierarchical representation of an agent's goals and capabilities (see section 2), to represent their knowledge.

The primary role of the MASS controller 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 [3] 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 best case outcomes, the simulator will degrade results as necessary (lower quality, more cost or longer duration) if required resources are not available, or if execution is affected by interactions with other methods.

The operating environment is represented in MASS by two mechanisms, a visual representation and a list of resources. The visual representation is composed by a 2D map (the floor map of a house, for example). Each agent connected to the simulator can the post its visual representation (usually an icon) and location on the map. Semblances of movement can be obtained by simply informing the simulator of a new location. The resource listing shows the state of all the resources used by agents' methods during their "simulated" execution. The simulator is able to define resources a priori, but for convenience they are usually instantiated dynamically at runtime as the objective TÆMS view specifies a need for them.

The simulator's second purpose is to act as a message router for the agents. The agents send and receive their messages through the simulator, which allows us 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 reciept from

one run to the next.

4.1.1 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 time line into a number of slices. In this model, the simulator begins a time slice by sending a pulse to all of the running components, which allows them to run for a period of (real) CPU time. In our model, a pulse does not have a predefined CPU time; each agent decides independantly when to stop running, which allows agent performance to remain independant 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 using a discrete notion of time but along with event based control. In this model, agents execute within descrete time slices, but are also notified of activity (method excution, 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.

4.1.2 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 simultaor, 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, etc...

2. Events and execution requests will always arrive at the same time. Note that this technique does not garuntee 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.

4.1.3 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 as one may do in the real world (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 consistancy 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, typically immediately, and duration, which depends on the method's probabilistic distribution as specified in the objective TÆMS task structure (see section 2). 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 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.

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-occuring 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 sucessfully 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 determinis-

tic, without sacrificing the unpredictable nature a real world environment would have.

5 Sample Runs

The first results presented here only reflect one type of coordination and is not yet a comparaison of different negociation protocols. 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. In this section, we will examine performance on an agent by agent basis, and compose an aggregate observation, using a working definition of optimal agent performance: the optimal performance of any agent is the performance achieved when it is run alone in the environment with ample resources with which to perform its tasks. Performance in this case denotes the quality the agent achieves and the constraints it meets, e.g., preference constraints or deadline constraints.

In the three experiments presented in this section, the IHome is populated by with seven agents, including the DishWasher, Robot, WaterHeater, CoffeeMaker, Heater, AirConditioner, and the OtherAppliances agent (that simulates the presence of multiple other agents in the environment). The communications patterns in each experiment are monitored, as is resource consumption and the behaviors of the agents. Communications statistics, such as the number of messages produced, provide a measure of the efficacy of coordination. The environment is held constant in each of the runs (in terms of communication bandwidth, execution performance of actions, etc.) while the availability of resources is varied.

In all three experiments, the preferred temperature setting is 76 degrees in all rooms and temperature change in the house is effected by the temperature-related agents, but also according to a curve that describes the heat exchange between the inside of the house and the outside environment and between the rooms of the house. The temperature related agents (AirConditioner/Heater) are reactive in nature, they respond to situations in which the temperature is not at its preferred point. In these experiments, the initial temperature is set at some point other than the preferred temperature and it is the task of the temperature control agents to bring it back into line. Like the temperature control agents, the WaterHeater agent works to keep the hot water level between a defined minimum and maximum capacities, and the tank is assigned an initial quantity of hot water. Using the MASS simulator, we are sure that all thoses experiences are done in the same controllable run and there is no unexplicit randomness. Therefore it's possible to compare all thoses runs together.

The objective in the experiments is for the agents to carry out their assigned tasks, e.g., make coffee or wash the dishes in the alloted time. For reactive agents, like the A/C agent, the objective is to satisfy the expressed preference constraint, e.g., keep the temperature at 76 degrees, keep the water tank above the defined minimum, and so forth.

In the first experiment, the resources are configured as follows: 15Kw of electricity is available, 140 gallons of hot water initially reside in the water tank and the tank maximum is 200 gallons, the maximum allowable noise level at any time is 120 Db, and the initial temperatures in the different rooms are as follows: bedroom 50F, bathroom 90F, kitchen 90F, living room 50F.

The results for the first run are shown in Figure 4. In this experiment the agents that require multiple resources to carry out their tasks, and who have longer sequential chains of actions that must take place, like the CoffeeMaker and the DishWasher, perform poorly when compared to their independent performance. Because these agents require multiple resources at the same moment their performance requirements are higher and in this situation, where resources are constrained, they are generally unable to obtain the necessary resources. In both of the experiments the deadline for task completion is fairly tight in order to make the coordination problem non-trivial.

In contrast to the long-planning agents the more reactive agents (WaterHeater, Heater, AirConditioner) fair better. The only difference between their individual runs and the group run is that in the latter case they take longer to achieve the desired results. The Heater and the AirConditioner take until time 77 to reach their temperature goal of 76F, in contrast to the 41 clicks required in the individual case.

The behavior of the CoffeeMaker and DishWasher agents indicate a problem with our simple protocol. Though we have priority measures, higher priorities are not assigned to agents that are currently executing their plans. Thus tasks like making coffee are always superseded and interrupted by other higher priority tasks. Additionally, the priorities of tasks are not elevated as they are interrupted, thus they do not become less interruptible over time (a feature often found in priority based scheduling algorithms) or as they get closer to their deadlines. The problem also stems from a flawed implementation of personal preference – agent priorities in these experiments do not always reflect the client's personal preferences and thus the notion of a global utility function (even a local view of one) is somewhat muddied. This issue is currently being addressed.

The second experiment is identical to the first, with the exception that the coffee making tasks are assigned the highest overall priority in the system, enabling the CoffeeMaker to obtain the desired resources to carry out its tasks. However, its resource consumption pushed back temperature regulation tasks resulting in the A/C and Heater agents taking until time 90 (rather than 77) to reach their target temperatures.

In the third experiment, Figure 5, the resources are configured similarly except that the maximum capacity of the water tank is reduced to 60 gallons and it is empty at the start of the experiment. This decrease forces all agents using hot water to negotiate over the resource. In this case, the DishWasher is able to perform only one task out of its four assigned tasks. The 84 messages sent by the agent is testimony to its attempts to obtain the resources so that it could perform its other tasks (it was refused and canceled by the WaterHeater).

Interestingly given the tighter hot water constraints, the WaterHeater agent also performed fewer tasks than it did in the previous experiments. This is because the DishWasher was unable to execute, and the maximum capacity of the tank was reduced, thus the demand for water from a volume perspective also decreased. It is also interesting to note that the WaterHeater sent a large number of nullification or cancellation messages to all of the consumer agents because it was unable to fulfill all the requests it received.

In this run the AirConditioner and Heater agents also failed to reach their target temperatures. This is the result of the DishWasher's thrashing behavior. It would request and reserve electricity and thus interfere with the temperature control agents. When the DishWasher was unable to obtain the desired amount of hot water, it would release the electricity reservation but the thrashing behavior confused the (slow to respond to released resources) temperature control agents, resulting in diminished performance on their part.

In addition to the three coordination experiments, we also performed an experiment in which the appliances are not intelligent (normal appliances) and do not coordinate over resources. In this case, appliances are given a set of tasks to perform and they simply attempt to carry out the tasks. Resources are configured as with the first coordination experiment, i.e., 15Kw of electricity is available, 140 gallons of hot water initially reside in the water tank and the tank maximum is 200 gallons, and the maximum allowable noise level at any time is 120 Db. With no coordination, 9 minutes after the start of the simulation the electricity resource is overwhelmed

Agent	# of Tasks		Final Quality		Resources Mes.		Conflict				Tasks Dropped	
	Alone	IHome	Alone	IHome	Alone	IHome	Alone	IHome			Alone	IHome
								E	R	N		
Dishwasher	4	**2**	135	**76**	10	10	0	21	1	5	0	**2**
Robot	5	5	10	10	0	0	0	0	0	0	0	0
WaterHeater		83		10				26	3	0	0	0
OtherAppliances	37	33	10	10	42	36	0	8	15	2	0	4
CoffeeMaker	4	**0**	80	**0**	3	3	0	1	11	24	0	**4**
	4	4	125	125	3	3	0				0	0
	4	4	125	125	3	3	0				0	0
Heater	8(41)	7 (77)	40	40	8	7	4	8	3	0	0	1
AirConditioner	8 (41)	12 (77)	35	20	16	24	4	12	4	0	0	0

Figure 4: Experiment 1: *Alone* indicates the performance when the agent executed alone in the environment with sufficient resources. The *IHome* column indicates performance when the agents are executed in a group and resources are shared. *E/R/N* indicates conflicts emitted, received, or nullified.

Agent	# of Tasks		Final Quality		Resources Mes.		Conflict				Tasks Dropped	
	Alone	IHome	Alone	IHome	Alone	IHome	Alone	IHome			Alone	IHome
								E	R	N		
DishWasher	4	**1**	135	**40**	10	**84**	0	16	2	11	0	3
Robot	5	5	10	10	0	0	0	0	0	0	0	0
WaterHeater		**50**		10				**43**	0	**31**	0	0
OtherAppliances	37	28	10	10	42	37	0	1	13	9	0	**9**
CoffeeMaker	4	0	80	0	3	3	0	3	4	21	0	4
	4	4	125	125	3	3	0				0	0
	4	4	125	125	3	3	0				0	0
Heater	8(41)	9 (*)	40	40	8	9	4	6	2	0	0	0
AirConditioner	8 (41)	10 (*)	35	20	8	**22**	4	1	6	0	0	0

Figure 5: Experiment 3: *Alone* indicates the performance when the agent executed alone in the environment with sufficient resources. The *IHome* column indicates performance when the agents are executed in a group and resources are shared. *E/R/N* indicates conflicts emitted, received, or nullified.

by the DishWasher's pre-rinse cycle and the CoffeeMaker's brewing in conjunction with the other appliances being active. The total demand was 19Kw. The severity of this event is dependent on one's model of what should happen in the event of an overload, e.g., circuit breaker cutting out and all actions coming to a halt until a human resets the breaker. If we ignore the electricity overload and continue, the appliances later exceed the noise threshold of 120Db by 10db. Other examples abound. Obviously, this experiment is based on the assumption that all tasks would be carried out at the same time without intelligent controllers, when in fact, a human would be handling the sequencing. It is intended only to illustrate the role of coordination in this context.

6 Conclusions and Future Work

We have designed and implemented a simulated intelligent home environment and populated it with intelligent appliance agents. The agents interact and coordinate using the simple protocol over shared resources, contract over task-allocation interactions, and use a different coordination protocol for task overlap conditions. While we are pleased with this work, there is much room for improvement and expansion.

IHome environment scenarios and situations requiring more complex negotiation between the agents. Temporal chains of multi-resource tasks is one example of this – particularly if member tasks are assigned to different agents. This leads to an interrelated multi-agent task and resource coordination problem. The introduction of multiple robots in the environment will motivate this area of exploration. We will also explore survivability, adaptability, and responsiveness issues in this context. We feel that diagnosis is a key part of adaptability, which is needed to make MAS more robust in changing or adversarial environments. Our goal is to use diagnosis [7] and adaptability to allow the agent to dynamically work towards the appropriate tradeoff of robustness versus efficiency. Ongoing research is looking at diagnosis of coordination activities, and how diagnostic and evidential information can be modeled in a domain independent manner.

As mentioned, other areas of improvement include refining our evaluation metrics so that we can more easily evaluate experimental data and fully incorporating personal preference profiles into the agents' priority mechanisms.

But we have alsa tested our MAS Simulator and its associated framework to built the Ihome project very quickly. The MAS Simulator will allow us to really compare all the protocols we are going to implement.

In short, the intelligent home is proving to be an interesting environment for experimentation with MAS technologies. The multiple different types of resource and task interactions present in this application domain provide a rich landscape for work in coordination and local agent control.

References

[1] Jason A. Brotherton and Gregory D. Abowd. Rooms take note: Room takes notes! In *Proceedings of the 1998 AAAI Spring Symposium on Intelligent Environments*, pages 23–31, 1998.

[2] T. Darell, G. Gordon, J. Woodfill, and M. Harville. Tracking people with integrated stereo, color, and face detection. In *Proceedings of the 1998 AAAI Spring Symposium on Intelligent Environments*, pages 44–49, 1998.

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

[4] Keith S. Decker. Task environment centered simulation. In M. Prietula, K. Carley, and L. Gasser, editors, *Simulating Organizations:*

Computational Models of Institutions and Groups. AAAI Press/MIT Press, 1996. Forthcoming.

[5] 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.

[6] Bryan Horling. A Reusable Component Architecture for Agent Construction. UMASS Department of Computer Science Technical Report TR-1998-45, October 1998.

[7] Bryan Horling, Victor Lesser, Regis Vincent, Ana Bazz an, and Ping Xuan. Diagnosis as an Integral Part of Multi-Agent Adaptability. Computer Science Technical Report TR-99-03, University of Massachusetts at Amherst, January 1999.

[8] Bernado Huberman and Scott H. Clearwater. A multi-agent system for controlling building environments. In *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS95)*, pages 171–176, 1995.

[9] David Jensen, Michael Atighetchi, Rgis Vincent, and Vicotor Lesser. Learning quantitative knowledge for multiagent coordination. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence*, Orlando, FL, july 1999. AAAI.

[10] Yannis Labrou and Tim Finin. A Proposal for a new KQML Specification. Computer Science Technical Report TRCS-97-03, University of Maryland Baltimore County, February 1997.

[11] Victor Lesser, Keith Decker, Norman Carver, Alan Garvey, Daniel Neiman, Nagendra Prasad, and Thomas Wagner. Evolution of the GPGP Domain-Independent Coordination Framework. Computer Science Technical Report TR-98-05, University of Massachusetts at Amherst, January 1998.

[12] Victor Lesser, Bryan Horling, Frank Klassner, Anita Raja, Thomas Wagner, and Shelley XQ. Zhang. BIG: A resource-bounded information gathering agent. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, July 1998. To appear. See also UMass CS Technical Reports 98-03 and 97-34.

[13] Victor Lesser, Atighetchi Michael, Brett Benyo, Bryan Horling, Anita Raja, Regis Vincent, Thomas Wagner, Ping Xuan, and Shelly XQ Zhang. The umass intelligent home project. In *Proceeding of the Third Conference on Autonomous Agent*, 1999. http://mas.cs.umass.edu/research/ihome/.

[14] Michael C. Mozer. The neural network house: An environment that adapts to its inhabitants. In *Proceedings of the 1998 AAAI Spring Symposium on Intelligent Environments*, pages 110–114, 1998.

[15] Rainer Stiefelhagen, Jie Yang, and Alex Waibel. Towards tracking interaction between people. In *Proceedings of the 1998 AAAI Spring Symposium on Intelligent Environments*, pages 123–127, 1998.

[16] The UMASS Multi-Agent Systems Laboratory. http://mas.cs.umass.edu.

[17] Regis Vincent, Bryan Horling, Thomas Wagner, and Victor Lesser. Survivability simulator for multi-agent adaptive coordination. In *Proceedings of the First International Conference on Web-Based Modeling and Simulation*, 1998. To appear. Also available as UMASS CS TR-1997-60.

[18] Thomas Wagner, Alan Garvey, and Victor Lesser. Criteria-Directed Heuristic Task Scheduling. *International Journal of Approximate Reasoning, Special Issue on Scheduling*, 1998. To appear. Also available as UMASS CS TR-97-59.