A GENERIC DISTRIBUTED SIMULATION SYSTEM FOR INTELLIGENT AGENT DESIGN AND EVALUATION

John Anderson
Department of Computer Science
University of Manitoba
Winnipeg, Manitoba, Canada R3T 2N2
andersj@cs.umanitoba.ca

Keywords: Generic Simulation, Distributed Simulation, Intelligent Agents, Multiagent Systems.

ABSTRACT

Using a simulator to design and evaluate intelligent agents in realistic environments places enormous demands on a simulation tool: everything from supporting multiple agents and their interactions, to providing detailed control over trials in an environment, to accurate perception within computational bounds. While the computationally intensive nature of this process is the most obvious reason to consider distributed simulation, we have also found that distributed simulation provides solutions to timing and perceptual problems that are particularly difficult in single-system simulation. This paper describes ongoing work on *DGensim*, a distributed version of the Gensim single-system simulator, and the significant advantages that distribution brings to the simulation process in this case. We also discuss the difficulties of preserving the generic aspects of a simulator in a distributed setting.

1. INTRODUCTION

While Artificial Intelligence is contributing important elements to applied simulation research, simulation has also has an extremely influential role to play in pure and applied AI research. Even those who insist that intelligent agent research requires the physical embodiment of agents usually employ simulation as an integral component of the design of those agents (e.g. Balch, 1998a). Simulation also provides important elements of control for the evaluation of intelligent systems (Hanks et al., 1993), as well as solutions to problems in the real world that are beyond the capabilities of current AI technology, allowing research one area to proceed despite the immaturity of research in related areas (Anderson, 1995). Simulation also allows us to model natural intelligent systems, thereby deriving a better understanding of the techniques employed in those systems and ultimately the means to exploit those techniques in new applications (e.g. Picault and Collinot, 1998)

Using a simulator to design and evaluate intelligent agents in realistic environments places enormous demands on a simulation tool: everything from supporting multiple agents and their interactions, to providing detailed control over trials in an environment, to accurate perception within computational bounds (Anderson, 1995). Surrounding these specific issues however, is the more pervasive problem of wide applicability: in order to perform ongoing research, where agent designs and the environments in which they are examined change as development pursues, we require a tool that will easily support such changes. Similarly, there are many applications outside of the development of intelligent agents themselves in which complex environments populated by such agents are useful: natural resource management (Deadman and Gimblett, 1994), biology (Kester, 1996), economics (Deadman, 1999), and sociology (Halpern, 1999) to name a few. Ideally, a simulator generic enough to support diverse agents and environments should also be applicable to these areas and others.

With this in mind, we developed Gensim, a generic timeshared simulator for multi-agent systems (Anderson and Evans, 1995). We have employed this system ourselves in intelligent agent design and verification (Anderson, 1995; Anderson and Evans, 1996) and have shown its potential in areas outside of this environment (Anderson, 1997; Anderson and Evans, 1994). This system is generic in that agents and environments can be easily defined and interchanged in a modular fashion. The system also provides pragmatic support for agent sensing, control over agent timing, and facilities for constructing domains and agents. However, Gensim has several significant limitations in its original form, and others for which we have developed compromises that are still philosophically awkward.

The major motivation for considering distributed simulation, in most cases, is the increased computing power it makes available (Hamilton et al., 1997). In fact, some take the view that the very purpose of distributed simulation is to reduce the simulation time when compared to sequential simulation (Pham et al., 1998). The advantage of additional computational power is certainly attractive in an application such as multiagent modeling. When modeling intelligent agents a great proportion of system resources is required to support the decision-making processes of these agents – especially so in heavily deliberative agents and agents that themselves model their world. While other simulators deal with this by assuming a simple reactive (and thus low resource consumption) agent model (e.g. Balch, 1998b), or by replacing agents with process-based simulation wherever possible (Hamilton et al., 1997), Gensim allows the implementation of as many internally sophisticated agents as desired. This forces the ongoing simulation to proceed more slowly on a single machine, though internal simulation time is of course unaffected. Much of the work performed in our laboratory is multiagent simulation work where the focus lies on agent collaboration or competition, and the evaluation of individual agent designs under experimental conditions is especially important. In addition to increased computing power, the greater experimental control and lower timesharing overhead afforded by placing the processes associated with a single agent on a single machine are also beneficial in such situations.

Despite the obvious advantage of increased computing power, our major motivation in moving to a distributed simulation model was in fact to increase the fidelity of Gensim simulations, as will be described in the next Section. While it is at least potentially acceptable in many cases to wait for a simulation involving computationally intensive agents to complete, distributing a simulation allows for better solutions to several awkward problems, including timing across agents and more accurate perceptual interface, thus leading to improved simulations.

These advantages led us to the development of *DGensim*, a system under ongoing development in our Autonomous Agents Lab. This system will ultimately form the major non-robotic platform for agent development and experimentation in our laboratory, and because of this it is our intent to keep the generic nature of the system as close to the original Gensim system as possible. This should also allow its potential transfer to the other application areas mentioned above. The remainder of this paper introduces the Gensim simulation approach, describes the methodology employed in creating DGensim, details the advantages that distribution brings to the Gensim approach, and discusses the difficulties of providing generic elements in a distributed simulation system.

2. GENSIM AND DGENSIM

As mentioned in the previous Section, Gensim (Anderson and Evans, 1995) is a generic simulation system for intelligent agent designs. It supports multiple agents consisting of multiple timeshared processes, manages an object-oriented environment, treats the relationship between agents and their environment consistently, and views agents and environments as plug-in modules that are easily substituted.

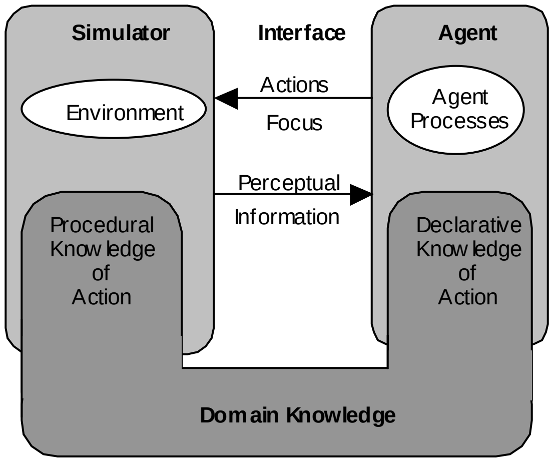

Figure 1. Conceptual View of Gensim (Anderson and Evans, 1995).

Figure 1 illustrates a conceptual view of the Gensim system. In Gensim, a LISP-based simulation process manages an object-oriented view of the environment, including the physical embodiment of the associated agents themselves. Collections of agent processes (also LISP-based) making up the decision-making components of agents are timeshared, with equal timeslices usually given to each agent. Agent processes are run, and during their timeslices may or may not commit to particular actions, which are communicated to the simulator. The simulator manifests these changes in a time-based manner (i.e. manifesting the changes made during the agent timeslices just completed), using an event queue to manage future change. During agent time-slices, agents also make requests for perceptual information within their current context, and after the simulator manifests the changes initiated during the current cycle, this perceptual information is provided based on limitations of bandwidth and recorded agent perceptual abilities. The system also contains specialized representations and mechanisms for keeping the event queue to a limited complexity; balancing accurate and pragmatic perception; allowing the translation of references to objects in agents' own world models in actions communicated to the simulator; allowing flexible autonomy for agents; and relatively easily defining agents and environments in a modular fashion. Details of these and other significant aspects of the system may be found in (Anderson and Evans, 1995).

Given that the major computational overhead in many multiagent simulations is the management of intelligent agent decision-making processes, the obvious choice for distribution is the movement of such internals to separate hardware systems. While it is desirable for experimental purposes to have a single agent per machine in order that machine load does not affect the performance of one agent over others, from the standpoint of keeping the simulator more broadly useful, it was decided to support as many agents as desired per component system.

The basic model employed in DGensim is for the most part the same as the original Gensim system of Figure 1, with some important differences that will be dealt with shortly. The physical organization and execution layer of DGensim however, is quite different. As shown in Figure 2, DGensim revolves around a set of *n* node machines, *n*-1 of which are dedicated to executing agent internals (the agent processes of Figure 1, which ultimately take perceptions, perform intelligent processing, and make commitments to action). The remaining node runs the *environment manager*, which performs most of the functions of the simulator process of Figure 1, with some very important exceptions that make significant philosophical and practical improvements over the original Gensim system. In order to deal with the management of multiple agents per agent node, an *agent manager* is employed. This process is initially run on each machine that is to participate in the simulation by supporting agent internals, and connects to the environment manager via a prespecified Linux port. When the environment manager is started on the environment node, it makes contact with remote nodes and transfers agent code to them. This transfer is intended in future to allow the environment node to choose the most appropriate machine for a particular agent. The agent manager keeps track of the communication ports and process details for all agents on that particular machine. Beyond accepting agent code, the agent manager is also sent limited environmental information, which allows it to play another important role that will be described shortly.

The most immediate result of the distribution of agent internals is a much more natural, realistic flow of agent decision-making over time due to a more realistic execution of the underlying agent processes. This is due in part to improvements in implementation platform. The choice of platform for DGensim was Allegro Common Lisp (ACL) under Linux, which among other things allows OS-level threads as opposed to the application-level timesharing of the original Gensim system. One of the original intentions for further development with Gensim (Anderson, 1997) was to port the system to Java, due to the language's strong supports for multithreading and networking. The difficulty with this however, was providing the same facilities for rapid agent and environment construction and the same level of support for AI components within agents afforded by Lisp. However, given the advances in multiprocessing and network sockets in ACL, the ACL/Linux platform brings all the major advantages of Java, while allowing code compilation, and the ease of definition discussed above.

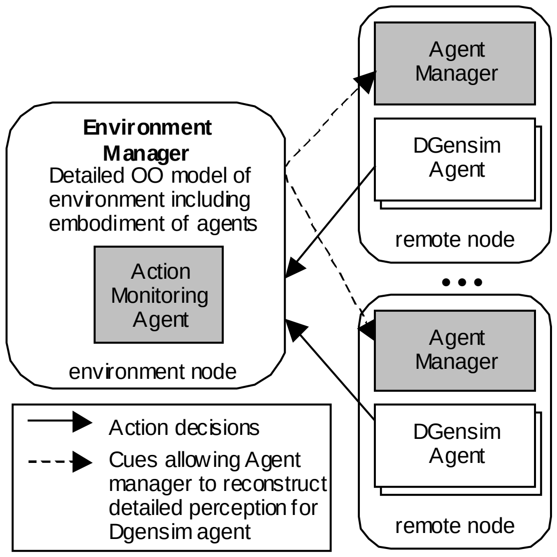

Figure 2. Overview of DGensim Organization.

2.1 Actions and Timing

There are also significant improvements in timing accuracy brought about by distribution *per se*. Because change initiated by agents is processed on an agent-by-agent basis cyclically in Gensim, some agents get a chance to have the change from their actions manifested ahead of others in the same cycle consistently. This leads to a predictable outcome of single-cycle interactions by the ordering of agents unless care is taken when defining those actions and their possible interactions. This and related timing problems are diminished in Gensim through the encouragement of small time cycle lengths, limiting the effect of this on the accuracy of a simulation, but cannot be completely eliminated. Rather than allowing an agent to take as many fixed time steps as required to come to a decision upon action, and then processing that decision along with those of other agents during that same cycle, agents in DGensim send their timestamped decisions asynchronously to an action monitoring agent (see Figure 2). This relatively simple agent (running on the same system maintaining the object-oriented environment) organizes incoming decisions and assists in correcting for limited network delays using the timestamps on incoming actions to order the queue of pending events.

While agent processes in DGensim make asynchronous decisions for action, the environment around those agent processes flows at a constant rate through time – thus defining the flow of time for the agents involved. The original organization of Gensim was intended to approach this, but was limited in this capacity simply because it was based on the concept of a single agent decision-making process being executed at a time. As in Gensim, the environment manager in DGensim is a time-driven simulation, since it is managing time for the external agent processes for which it exists. It cannot simply take large temporal leaps ahead to an upcoming future event and process it if there are no other pending events, since agents may make action commitments before these future events are meant to occur. Others have pointed out that complex time-based simulations can often become unwieldy (Hamilton et al., 1997), but in this case a time-based simulation is necessary. Time needs to flow for the agent-decision making processes in a constant manner, just as it does in the real world, since the simulation itself exists solely for the purposes of the agents that inhabit it.

Like Gensim, DGensim contains components that assist in making the time-depended simulation more efficient. For example, the event queue uses *event generators*, which insert the next of a long chain of events into the event queue when the current portion of an event sequence is processed. This allows us to represent ongoing sequences of events while only having any one part of that in the queue at a time, and also directly minimizes the amount of data that must be examined when the event queue is altered.

This timing model is both more accurate and far simpler to employ within DGensim agents than the original. As it is however, it is susceptible to problems with long network delays: in a wide-area network scenario, it is entirely possible to have an action commitment from an agent delayed enough that others that might have interfered with it would have been processed, and even related sensory information already delivered to other agents. It is a fairly simple matter to deal with small delays, based on the fact that the environment is updated in discrete time steps. Every event intended to occur during a particular unit time is processed in sequence, and as long as an action is received by the action monitoring agent within this time boundary, the action monitoring agent will rearrange the event queue such that everything is updated as if the actions arrived at the appropriate point in time. That is, there is a window of safety around with an event can be delayed, and a simulation may be designed to increase that window.

In the event that an action arrives after its time unit has been processed by the environment manager, compromises must be made. In this case, other agents may have already been given perceptions that might have been different had the action been received in a timely fashion (and if the delay is long, even made further action commitments on that basis). Given that the simulator is supposed to be an ongoing interactive real world for agents to inhabit, rollback is generally not an option. Rollback would also be difficult for large numbers of agents, and more importantly the mechanics of being able to roll back agent state change the nature of the agents themselves. The latter would lead to different results in experimentation than would be seen otherwise. Other options are made available however, in order to keep the simulator as general as possible. First, an action may be simply invalidated, as if it simply had not occurred, and the agent(s) involved informed of this. From an agent standpoint, this is not particularly realistic, but it may be useful in some simulations. Another available alternative is to process the event as if it had happened during the time unit in which it arrived as opposed to the one in which it was generated. Envision one agent throwing a ball, and a second (the late agent) attempting to catch it. In this situation, the end result would be as if the late agent moved to catch the ball too late to receive it. This is somewhat more suitable, but still inadequate for most agent evaluation experiments because the actions of the late agent itself were optimal – it was the nature of the simulation that caused the delay. Still, it is more philosophically pleasing than stopping a third agent that did catch the ball after the late agent missed, informing it that it in fact does not have the ball after all, and magically transferring the ball back to the late agent. We also allow both of these more viable alternatives on an action-by-action level, allowing the particular context to be considered as well.

Finally we allow a preventative measure that slows the simulation down significantly but deals with these problems as completely as can be expected. In DGensim, it is also possible to force each agent to transmit its actions regularly, including a *no-op* action if the agent is not performing any useful activity. This allows the environment manager to process one timeslice after a complete set of these actions is received. This is overwhelming in the case of a large number of agents however, or a particularly small time unit setting for the environment manager.

In the above scenario, we have been dealing with the late arrival of an agent action. The opposite scenario – the possibility that an agent may deliberate, commit to an action, then do the same before it gets perception from the first back because it is running faster internally than the ongoing simulation – is less troublesome. Here what is happening in terms of the model is that an agent is choosing to commit to actions without the aid of additional perception (the same situation arises above if network problems lead perceptions to be late). It is possible to implement the time it takes to actually perform the action as part of the action itself, which would cause a properly implemented agent to delay the commitment until it was effectively finished performing the first action, and thus take care of situations where agent processes were running artificially faster than the environment or other agents.

Note that none of these approaches to dealing with delay (except the case above where an agent can effectively be made to act in synch with a slower environment) is particularly appealing in the case of experimentally evaluating agents: network delays significantly affect results. Even in the situation where the environment manager can expect a completely regular response from each agent, it may not be worth continuing experimentally if it does not get one (i.e. an agent is severed from the simulation temporarily). However, this is a particularly exacting application of distributed simulation, and these approaches are available to be used where that may be more suitable. In our own work, because of the nature of the problems we are investigating, we naturally use small dedicated networks, and would consider an experiment invalid if delays in action reception such as those considered above occurred. On this scale however, such delays are rare and thus an insignificant downside in comparison to the timing benefits alone. However, this problem does affect elements of generality to which we are aiming. Issues related to this will be explored in Section 3.

2.2 Perception

Perception and its relationship to the agent and environment is another significant problem in Gensim that was dealt with there through compromise, but which can be significantly improved through a distributed implementation. In Gensim (and DGensim as well), perception is implemented pragmatically at the object level (Anderson and Evans, 1995), in order to remove the burden of pixel-level perception for agents – one of the major reasons simulators are employed in intelligent agent development (Hanks et al., 1993; Anderson, 1995). In both Gensim and DGensim, agents specify their interest using concepts familiar to both the agent and the simulator (e.g. scan for blue objects; where is the other agent) or may simply gaze in a direction of interest. In the original Gensim system, the simulator responds to this request with object-attribute-value specifications for a limited range (based on a model of the agent's perceptual abilities) of what can be perceived (given the agents' perceptual and focus biases). However, this is an artificial organization of perception, in that the simulator is doing more than just removing the overly-detailed burden of low-level vision – it is actually doing all the agent's perception for it aside from the highest level of integrating those perceptions into the state of the agent's decision-making components. The environment simulation process is deciding which objects the agent can see, and which it is limited from seeing. While some perceptual limitation is due directly to the environment and does seem to fall within these bounds (e.g. objects block one another, fog can obscure objects), others are due to the agent (what type of objects it is biased toward focusing perceptual attention upon, for example). Placing the perceptual component completely within either the agent or environment is philosophically inaccurate and technically problematic. Beyond philosophical issues, there are practical issues involved here as well: this sensory preparation is an extremely computationally-intensive element, and it is necessary to strike a balance in this aspect as well.

The solution to this problem in DGensim is to move the management of perception to a point *between* the agent decision-making components and the simulated environment. In DGensim, the bulk of perception is managed by the agent managers running on each agent machine. In response to an agent's perceptual request, the agent manager receives a description of objects that could be perceived based on the physical aspects of the environment – it is up to the agent manager to filter these according to the particular biases of the agent itself. This places perception in a much more natural position in a simulation organization, and removes a significant amount of unnecessary work from the environment manager. However, it also results in the potential for an inordinate amount of information transfer, and potential network overload for a large number of agents.

In order to restrict the amount of information that must be physically sent across the network by the environment manager, agent managers contain simplified environmental information: stereotyped views of objects in the environment that are exported to agent managers by the environment manager each time a new environment is used. When the agent manager accepts a particular agent, a registration is created indicating the mapping between agent and manager, a part of which states the frequency with which sensory information should be sent to the particular DGensim agent (via its manager). This effectively states the speed at which the agent can perceive objects. The environment manager maintains the agent's current orientation and maximum sphere of attention (physical attributes of the agent's body, which it is managing as part of the environment), and relays very basic object information to the appropriate agent manager. This information is essentially which object stereotypes to invoke and the particular changes beyond those stereotypes. The agent manager then reconstructs detailed attribute perceptions (the information originally provided by the simulator in Gensim) based on the information received and its local knowledge. This approach pragmatically balances reasonable perception with network bandwidth, and also allows us to deal with the perception-related problems described earlier.

This rebalancing of the function of perceptual information service is also in part a consequence of distribution: given that agents are now separated from the environment processes by machine boundaries, it makes sense to move those elements of perception that do not belong in the environment there as well. This also assists significantly in improving efficiency: In Gensim, the bulk of the simulation time was spent preparing perceptual information and translating it into the object references or descriptions an individual agent would comprehend for each unique agent (DGensim employs the same deictic object description facilities provided for Gensim; the interested reader is referred to (Anderson and Evans, 1995) for more details on this). Moving a significant portion of the work previously done by the environment simulator to the distributed components allows the simulation to run more efficiently and keep up with distributed agent internals. This in turn helps to minimize the problems associated with getting perception information back to agents in a timely fashion as discussed above.

In this section we have outlined a few of the major benefits brought about by a distributed redesign of the original Gensim system. In each case however, solutions improving upon aspects of the original Gensim system has impacted to some degree on the generic nature of the system. Since generality is a major focus in both Gensim and DGensim, the impacts on generality of dealing with the problems discussed above, as well other aspects of moving to a distributed simulation, need to be examined in more detail.

3. BREADTH, GENERALITY, AND DGENSIM

When employing a tool for examining intelligent agents within a simulated environment (or to take a different perspective, constructing a simulation model that happens to employ intelligent agents), breadth and generality in such a tool is extremely important. While it is certainly possible to construct completely specialized environments from scratch for each new project (and this occurs in AI and other areas with an unfortunate preponderance), we would prefer a tool that is flexible enough to support a wide range of agents and environments. Clearly this avoids unnecessary reimplementation, but control and verifiability aspects, for example, are involved as well. A tool allows a researcher to ensure that implementations under comparison have identical constraints placed upon them, and helps to ensure scientific validity in the results obtained. More importantly however, a flexible tool is required because of the complexity of the agents and environments themselves. Complex software systems can neither be completely understood nor completely designed in advance. This will sometimes necessitate changes in design approach in mid-stream, and if our original tool does not support such changes, the tool must be abandoned and some (possibly extensive) reimplementation performed. In the hypothetical worst case, we have to completely re-build the entire environment and agents *de novo* with each small design change. The desire to impose control and avoid reimplementation was our primary motivation for building the original Gensim system. In Gensim, we attempt to make as few assumptions about the nature of agents, the nature of environments, and the nature of their interface as is possible. Gensim does not assume any particular mode of operation for an agent, and makes no assumptions beyond a representation for space in an environment. Some assumptions must be made in order to have a coherent interface between agents and a simulated environment (e.g. common name for concepts or an ability to translate these, so agents can refer to objects in the environment; the common representation for space in order to specify objects and activities relative to the position of the agent). A common model of perception and action is also necessary, so that agents may communicate their committed actions appropriately and receive perceptual information from the simulated environment. These however, are relatively insignificant compared to the architectural assumptions of most agent-based simulators (Anderson, 1995; Anderson and Evans, 1995).

The generality and breadth capabilities of a simulation tool can be viewed as a spectrum along numerous dimensions, including the degree of programming effort required (typically the stronger the reliance on pre-programmed components, the smaller the breadth of situation the tool can fit) ; the breadth of agents that can be supported (in Gensim, for example, agents must fit the above criteria – this includes many agents but certainly not all possible agents); and the breadth of environments that can be represented. In particular, supporting a wide variety of agent types or architectures makes considerable demands on a simulator: in many simplistic agents, for example, using the perceptual facilities available in Gensim is overkill, while in others the perceptual model would be too unrealistic (e.g. agents where low-level vision is an important part of the agent architecture). Similar criticism can be made of any other agent decision. In this situation and others concerning generality, Gensim and DGensim attempt to hit a middle ground. Supporting every possible agent type, for example, is simply not viable: there would be so many potential facilities for perception alone and so much custom programming required that the tool would become largely unusable. Instead, Gensim and DGensim implement a range of facilities that attempt to hit what most agents in an experimental situation would require, allowing the use of custom programmed components where these will not suit. Similar provisos exist for other aspects of these systems as well.

The practical objective of DGensim is unchanged from that of the original Gensim system: to provide a generic platform for intelligent agent research. However, like the issues of breadth discussed above, there are issues of breadth and generality peculiar to the element of distribution itself that must be considered, and similar mid-level approaches taken. The difficulty with distributed simulators in this regard is that they have a far wider range of motivation than non-distributed simulators, and far more significant architectural consequences of those disparate motivations. In distributed simulations, primary motivation ranges from providing significantly detailed graphical environments suited to human participants, to focusing on the interactions of computational agents. Environments range from the corners of the internet to local area networks, and reliability, security, control, and level of fidelity and graphic detail (e.g. Jinxiong and Sartor, 1994; Reese, 1994) vary considerably, as do the number of expected agents (for example, DIS (IEEE, 1993) vs. DDD (Song and Kleinman, 1994)). More pervasive issues such as the integration of different simulation tools (e.g. the DIS (IEEE, 1993) and HLA (DOD, 1998) standards) also surround this spectrum. In short, the range of variability in even the same application area is orders of magnitude larger in a distributed simulation, because the element of distribution increases the potential applicability of the simulation and the impact on design significantly.

Given that our own primary motivation is the examination of intelligent agents and their interactions, we currently assume a local network, where security and reliability are not strong issues (though these are not completely absent either). Even with these restrictions, perfect generality is impossible without weakening a tool to the point of uselessness (as would perfect generality in any of the issues mentioned earlier), and so tradeoffs are made as they were in the original Gensim system to balance utility with generality. One significant aspect of this has been the design of the system itself (Figure 2). It is logical in an organizational sense to keep the environment on a single machine and distribute the agent internals, since the latter are loosely coupled and the former is highly interactive, making the synchronization of elements of the former difficult and computationally expensive (Hamilton et al., 1997). It is also logical in a practical sense from our own perspective, in that sophisticated agent internals are the most computationally costly elements of our own simulations. Within this organization we can still support a broad range of agents and environments, but the choice of distributing agent internals and leaving the environment on a single machine limits true generality in several ways. Most obviously, the physics of the environment cannot be computationally overwhelming, since they are on a single machine.

As mentioned, this has not at all been a problem in our own work, but may be such in others, and the ability to distribute the environment itself is in our plans for the future. Also as mentioned earlier, we have attempted to hit a middle ground and lessen the load on the environment manager by moving much of the work associated with perception to remote nodes as well. More significantly, the network assumptions made in our own work (by the nature of the problems themselves) forbid much in the way of trouble with network failures or lag, important issues in wide-area simulation. We also intend to attempt to make this approach more applicable to wider areas in future as well, through the addition of additional mechanisms for dealing with network delay and failure.

One of the ways in with DGensim is more restrictive than the original Gensim is in its ability to deal with agents with variable autonomy. While there is no more that is required to be shared between a DGensim agent and its environment than there was in the original Gensim system (while perception processing has been moved, the same mechanics and shared concept requirements of perception persist), Gensim agents could take advantage of the fact that all agents were stored on the same physical machine and share agent components. This allowed not only agents with lower autonomy through the sharing of knowledge and components, but served to very quickly define large numbers of similar agents and also made the simulation more efficient in some cases. In a distributed scenario, supporting this could be overwhelming in terms of information exchange if agents with shared components were distributed, and so a restriction is made forcing the use of this feature to agents residing on the same machine. This is not impossible to handle, but introduces issues of consistency and control where users must be careful of machine load balancing to ensure the accuracy of agent performance in simulation.

4. DISCUSSION AND FUTURE WORK

As mentioned earlier in this paper, DGensim is in ongoing development using LISP on a network of Linux-based machines. We have currently made what we believe are reasonable assumptions to provide a useful tool supporting a broad range of agents and environments efficiently and effectively. While we have designed the system in particular for experimenting with intelligent agents in spatially-explicit multiagent settings from the point of view of the design and performance of those agents, we believe this system to be applicable to other areas where multiagent simulation is of use as well.

DGensim is also in ongoing development in a larger sense. What is described in this paper is the first phase of a much larger distributed simulation system which will hopefully be just as powerful while dealing with some of the issues affecting the implementation of significantly large numbers of agents along with a wider range of distribution. The next phase will involve the distribution of the environment across multiple machines as well as agent internals. As stated earlier, the tightly interactive nature of environmental components makes synchronization difficult. However there are strong elements of locality of reference in agent interaction in virtually any physical environment, pointing toward geographical division as a logical choice. With reference to generality, there have also been numerous applications of applied multiagent simulations with a strong spatial reasoning aspect, especially in environmental or ecosystem management settings (e.g. Deadman and Gimblett, 1994). In such cases, changing the internal representation of the environment to be compatible with a common GIS format, and allowing agent perception to effectively become a GIS query would form both an effective means of handling the physical aspects of perception (i.e. before they are given to the agent manager as detailed in Section 2), and providing a detailed spatially-oriented representation as well. This would be especially useful in domains where fairly strong geographic boundaries could be defined (i.e. where the likelihood of agent sensory range overlapping a boundary and thus having to be serviced from two different machines would be minimal). We also intend to add the ability for agents to migrate from machine to machine to rebalance loads and (in the case of a distributed environment as well) make the overall simulation more efficient and dynamic. Given that agent code migrates from the initial controlling (environment) machine to the distributed agent machines at the outset of a simulation, this in particular would not be a difficult extension.

Finally, we are interested in improving this system to provide more effective control and more precise simulations in heterogeneous environments, where different numbers of agents and their effect on machine execution speed must be dealt with, as well as varying network loads and network types. We are currently proposing funding for a large scale heterogeneous distributed system employing several different network facilities, machine architectures, and operating systems, in order to begin this work.

This paper has described an effective tool for the examination and experimentation of intelligent agents in simulated worlds, as well as the construction of simulations employing intelligent agents. We have endeavoured to illustrate that despite the common view of distributed simulation as a necessary evil for large-scale simulations of difficult problems, with its overhead paid back in increased performance, the distributed approach in fact brings about many practical advantages in this case. In particular, simpler timing (and at a lower level, the implementation of timing), and the provision of a pragmatic agent perception component benefit greatly specifically from being implemented in a distributed setting. While this is by no means a simulator that can be used to implement every possible distributed simulation, we believe it is applicable not only to the initial problem to which it was designed, but a wide range of applications outside of this. Future work will extend this range while keeping the pragmatic choices outlined here in mind.

BIBLIOGRAPHY

Anderson, J., “Supporting Intelligent Agents in Individual-Based Ecosystem Models”, *Proceedings of the Eleventh Annual Conference on Geographic*

Information Systems, Vancouver, BC, February, 1997, pp. 3-6.

Anderson, J., *Constraint-Directed Improvisation for Everyday Activities*, Ph.D. dissertation, Department of Computer Science, University of Manitoba, March, 1995. 397pp.

Anderson, J., and M. Evans, “Real-Time Satisficing Agents for Complex Domains”, *Proceedings of the Ninth Florida AI Symposium*, Key West, FL, May, 1996, pp. 96-100.

Anderson, J., and M. Evans, "A Generic Simulation System for Intelligent Agent Designs", *Applied Artificial Intelligence*, Volume 9, Number 5, October, 1995, pp. 527-562.

Anderson, J., and M. Evans, “Intelligent Agent Modelling for Natural Resource Management”, *International Journal of Mathematical and Computer Modelling*, Volume 20, Number 8, October, 1994, pp. 109-119.

Balch, T., (1998a) *Behavioral Diversity in Learning Robot Teams*, Ph.D. dissertation, Department of Computer Science, Georgia Institute of Technology, December, 1998. 204 pp.

Balch, T., (1998b) *Javabots Software*, http://www.cc.gatech.edu/~tucker/JavaBots

Deadman, P., “Modelling Individual Behaviour and Group Performance in an Intelligent Agent-Based Simulation of the Tragedy of the Commons”, *Journal of Environmental Management* 56, 1999, pp. 159-172.

Deadman, P., and R. H. Gimblett, “A Role for Goal-Oriented Autonomous Agents in Modeling People-Environment Interactions in Forest Recreation”, *Mathematical and Computer Modelling* 20:8, 1994, pp. 121-131.

U.S. Department of Defence (DOD), *High Level Architecture Interface Specification, Version 1.3*, April, 1998 (http://www.dmso.mil/hla/).

Halpern, B., “Simulation in Sociology”, *American Behavioural Scientist* 42:10, August, 1999, pp. 1488-1508.

Hamilton, J. A., D. A. Nash, and U. W. Pooch, *Distributed Simulation* (Boca Raton, Fl.: CRC Press), 1997. 390 pp.

Hanks, S., M. E. Pollack, and P. R. Cohen, “Benchmarks, Test Beds, Controlled Experimentation, and the Design of Agent Architectures”, *AI Magazine* 14(4), Winter, 1993, pp. 17-42.

Institute of Electrical and Electronics Engineers (IEEE), *International Standard, ANSI/IEEE Std 1278-1993, Standard for Information Technology, Protocols for Distributed Interactive Simulation*, March, 1993.

Jinxiong, C., and M. Sartor, “Fluids in a Distributed Interactive Simulation”, *Proceedings of the 5th Annual Conference on AI, Simulation, and Planning in High Autonomy Systems*, Gainesville, Fl, Dec. 1994.

Kester, K., Individual-based Simulation Modelling: Approaches and Application in Insect Behaviour and Ecology. *Symposia at Annual Meeting of the Entomological Society of America*, Louisville, 1996. HTTP://www.inhs.uiuc.edu/cbd/ESA-Annual/ESA_WWW_design.html.

Pham, C., H. Brunst, and S. Fdida, “How Can We Study Large and Complex Systems”, *Proceedings of the 13th Annual Simulation Symposium*, Boston, April 1998, pp. 126-134.

Picault, S., and A. Collinot, “Designing Social Cognition Models for Multi-Agent Systems through Simulating Primate Societies”, *Proceedings of the Third International Conference on Multi-Agent Systems*, Paris, July, 1998.

Reece, D., “Extending DIS for Individual Combatants”, *Proceedings of the 5th Annual Conference on AI, Simulation, and Planning in High Autonomy Systems*, Gainesville, Fl, Dec. 1994.

Song, A., and D. Kleinman, “A Distributed Simulation System for Team Decision Making”, *Proceedings of the 5th Annual Conference on AI, Simulation, and Planning in High Autonomy Systems*, Gainesville, Fl, Dec. 1994.

BIOGRAPHICAL INFORMATION

Dr. J. Anderson is an Assistant Professor in the Department Computer Science, University of Manitoba. His research involves aspects of autonomous agency and multiagent systems, including intelligent agent architectures, simulation environments for developing and experimenting with agents, agent collaboration, and mobile robotics.