Agent Breadth in a Tool for Distributed Multi-Agent System Development

John Anderson[1]
Department of Computer Science
University of Manitoba
Winnipeg, Manitoba, Canada
R3T 2N2

Physical multi-agent systems display many interesting phenomena, an understanding of which holds the potential for valuable advances in many fields. Studying the emergent phenomena displayed by the interactions of physically mobile agents in flocks and herds have the potential to revolutionize everything from the study of animal behaviour to the design of road systems to encourage particular traffic patterns. These same phenomena influence the design and development of mobile software agents used for applications as diverse as mail sorting and e-commerce. The study of these systems, however, demands sophisticated tools for the development and support of these agents in software environments. This is inherent in the nature of mobile or immobile agents that exist purely in software domains (e-commerce agents, for example). However, software environments for development and examination of physical mobile agents are also a necessity, especially for projects of any scale. Most research using physical agents for experimentation examines only a very small number of agents because of the cost and support required, and so even those who insist that intelligent agent research requires the physical embodiment of agents employ simulation as an integral component of the design of those agents (e.g. [Balch, 1998a]). Moreover, such software environments for examining and developing intelligent mobile and immobile agents have also been shown to contribute significantly to the overall robustness of large scale systems, allowing developer to analyze potential problems and explore the consequences of changes [Amin, 2000]. Software tools for the support of multi-agent systems also provide an important element of control for the purposes of experimentation [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].

The demands of supporting mobile intelligent agent development and experimentation place enormous expectations on a software tool for these purposes: everything from supporting sophisticated individual action and group interactions, to providing detailed control over trials in an environment, to accurate perception within computational bounds, to the efficient management of the objects collectively representing the agents' environment [Anderson, 1995]. Surrounding these specific issues however, is the more pervasive problem of wide applicability or *breadth*: in order to perform ongoing research, where agent designs and the environments in which they are examined change as development pursues (a very natural expectation in the complex and poorly-understood domains to which intelligent systems technology is most applicable), we require a tool that will easily support such changes. A development tool supporting this diversity also has the potential for significant benefit in many other application areas where complex environments populated by agents are useful: natural resource

[1] This work is supported through the National Science and Engineering Research Council and the Canada Foundation for Innovation.

management [Deadman and Gimblett, 1994], biology [Kester, 1996], and economics [Deadman, 1999], to name a few.

We believe that the future success of agent-based systems will rely heavily on development tools that directly support a broad range of agent types during development in a distributed setting. The direct support of a broad range of agents will allow a single development tool to be used as the nature of an agent changes during development, as well as for the tool to be applied to a very broad range of application tasks. Development in a distributed setting will allow the computational support of a large number of agents as well as a more accurate modelling of the interaction between agents and their environment, and between individuals and groups of agents, than is possible in a single-system setting. We are addressing these issues in *DGensim*, a distributed tool for multi-agent system development and experimentation, under ongoing development in the Autonomous Agents Laboratory at the University of Manitoba. DGensim is based on Gensim, a generic single-system simulator for multi-agent systems [Anderson and Evans, 1995; Anderson 2000]. We have already employed Gensim ourselves in multi-agent system design, experimentation, and evaluation [Anderson, 1995; Anderson and Evans, 1996] and have shown its potential in application areas outside of this pure multi-agent systems research [Anderson, 1997; Anderson and Evans, 1994]. Gensim is generic, in that agents and environments can be easily defined and interchanged in a modular fashion, and supports the modular design of agent components. The system also provides pragmatic support for agent sensing, control over agent timing, and facilities for constructing domains and agents.

DGensim is being developed to specifically deal with improving the issues of agent breadth and distribution mentioned above. Dealing with breadth is a contentious issue in a development tool, because there is an obvious balance between the power of a tool and the breadth of use of that tool. Programming languages are almost universally broad, for example, but require a correspondingly significant effort to mould the tool to any single precise use. Specialized shells, on the other hand, deal with a much narrower range of potential applications (and are therefore much less applicable) but require a greatly-reduced effort to employ in the situations for which they are designed. The difficulty with the latter, as is common across all of AI, is that a greater understanding of the potential domain may require changes that a chosen tool cannot support, necessitating significant reimplementation. In our work, we attempt to take a middle ground by providing built in elements and settings that allow a range of breath to be supported (agents with very simple vs. complex perceptual abilities; agents with or without world models or mobility), and as many optional facilities as possible (e.g. internal object-oriented knowledge representation tools, which may or may not be employed by agents within the system, assorted perception methodologies). The system is also as open as possible in terms of documenting the protocols and standards used, and designed in as modular a manner as possible, in order to allow the user to make extensions to the system in a modular manner [Anderson, 2000]. We have found the latter of all things to be the most useful in our work, and conversely, the lack of openness in other tools to be one of the most frustrating elements of employing them in multi-agent systems development.

In order to deal with agent breadth, we integrate into the system a shell for a broad agent known as an *improvising agent*. Improvising agents were originally designed to deal with modelling breadth in human problem solving [Anderson, 1995; Anderson and Evans, 1996, 1994]. The breadth displayed in such activities is not dissimilar at a component level to the breadth

associated with less-than-human level intelligent agents in a broad range of applications, allowing this agent architecture to be highly applicable in generic multi-agent settings. Improvisation is a common technique in human problem solving, involving real-time decision making using a common routine or plan, together with the abstract knowledge that can be used to supplement the plan or replace that plan when it has no useful recommendations. Improvisation has been previously shown [Anderson, 1995; Anderson and Evans, 1996, 1994] to be a flexible means for real-time decision-making under resource bounds in complex domains. Agents can rely strongly on their routines when knowledge or decision-making time is scarce, and can spend arbitrary amounts of time constructing and reasoning about available options to the extent to which these resources are available. We implement improvisation using constraint-directed knowledge structures, which allow an agent to reduce the problem of dealing with a broad range of possible actions to allowing its action to be constrained by a number of constraints that can be evaluated in real time. Changing the constraints associated with individual actions, tasks, or higher-level means of evaluating and comparing tasks allows for rapid and radical change in the way an agent operates, which in turn allows agents to be modified rapidly in response to changing tasks. Moreover, the means by which an agent recalls and evaluates constraints is itself controlled by a constraint-directed process, once again allowing a few small changes to drastically change the agent's behaviour and abilities and directly supporting a broad range of agents. We are working to include within DGensim a cleaner and simpler framework for defining such agents over and above the interface already available in Gensim. The original interface will still support the ability to define and work with agents of any kind (that is, to include one's own code rather than developing a suitable improvising agent), but the supplied framework will also provide the ability to quickly construct new agents on a spectrum of purely reactive to purely deliberative, and the ability to easily alter those agents as development progresses. This framework, like the rest of DGensim, is Lisp based, directly supporting the mobility of agents: because code and data are interchangeable, the migration of an agent from one system to another in a distributed setting is a simple process.

Beyond the breadth of agents described above, DGensim's second goal is the support of breadth in a distributed setting. Distributed development is a crucial element in a tool for the construction and support of multi-agent systems. The most obvious reason for this is that when developing and experimenting with intelligent agents a great proportion of system resources is required to support the decision-making processes of these agents, and distribution is a well-understood means of bringing about these additional resources. Indeed, the some take the view that the very purpose of a distributed development/simulation environment is purely to increase the overall speed when compared to that of a single system [Pham et al., 1998]. The advantage of additional computational power is certainly attractive in an application such as multi-agent systems. While other (single-system) tools deal with the resource problem 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], this does little toward supporting a broad range of agents.

Despite the obvious advantage of increased computing power, the major reason distribution is crucial in supporting multi-agent systems, and much of the focus of the current work on DGensim, is that a distributed setting directly contributes to the ease of agent support and to a greater fidelity of modelling. While it is at least potentially acceptable in many cases to wait for a multi-agent application involving computationally intensive agents to complete on a single

system, distributing an application allows for better solutions to several problems that we have found to be very awkward in working with multi-agent development tools in a single system setting.

One of the most significant of these is timing issues between agents. For example, because agents are timeshared cyclically in Gensim, some get a chance to perform their actions ahead of others in the same cycle, leading to 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 dealt with in the original Gensim system as well as in other timeshared simulators through the encouragement of small time cycle lengths, limiting effects on accuracy in a simulation. The distribution of agents removes artificial timesharing and gives a more grounded basis for the interaction between agents. Agent processes in DGensim make asynchronous decisions for action, while the environment around those agent processes flows at a constant rate through time – thus defining a natural flow of time for the agents involved.

Perception is another of these significant problems. In Gensim, perception is implemented pragmatically at the object level [Anderson and Evans, 1995]: agents specify their interest ("scan for blue objects") or direction of interest, and the simulator responds with object-attribute-value specifications for a limited range (based on a model of the agent's perceptual abilities) of what can be perceived. However, this is an artificial view of perception, in that the simulator is doing more than just removing the burden of low-level vision from the agent – it's actually doing all the agent's perception for it aside from the highest level of integrating those perceptions into the agent's state decision-making components. While some perceptual limitation is due directly to the environment (e.g. objects obscure one another, or fog can obscure objects), others are due to the physical abilities of agents (how far one can see in dim light, for example). Placing the perceptual component completely in either the agent or environment is philosophically inaccurate and technically problematic. Like agent decision-making, this sensory preparation is also computationally intensive and has practical limits of acceptability. The distribution of computational processes in DGensim allows perception to be handled by physically distinct distributed processes that do not take away from the computational resources allocated to agent decision making. One of the most important elements of DGensim is a scheme for caching environmental information, allowing the distribution of perceptual processes while still limiting the amount of information transfer that must occur to support agent perception.

Space limits the extent of technical discussion that can be made in a position paper, but a high-level illustration of the general organization employed in DGensim appears in Figure 1. Agent internals are distributed across a network of individual Linux systems (where each system can run a number of agents as individual processes if desired). Rather than the awkward lock-step timing of the original Gensim system (where individual agents could take any number of time steps to reach decisions for activity, but required similar internal time step lengths and specialized facilities to preserve state information), agents in DGensim send their timestamped decisions asynchronously to an action-monitoring agent. 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. This timing model is both more accurate and far simpler to employ within DGensim agents than the original. Perception is provided in the same pragmatic object-level fashion as the original

Gensim, but is managed by perception agents running on each agent machine in order to restrict the amount of information that must be physically sent across the network by the environment management system. Perception agents contain simplified environmental information, which can be thought of as stereotyped views of objects in the environment. Perception agents register with the environment manager and state the frequency sensory information should be sent to the particular DGensim agent, effectively stating the speed at which the agent can perceive objects. The environment manager maintains the agent's current orientation and maximum sphere of attention, and relays very basic object information to the appropriate perception agent. This agent then reconstructs detailed attribute perceptions based on 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: those elements of perception not within the agent itself (objects obscured by poor local visibility, or dim light, or other time-specific elements) are handled outside of the agent itself, as well as away from the main environment manager.

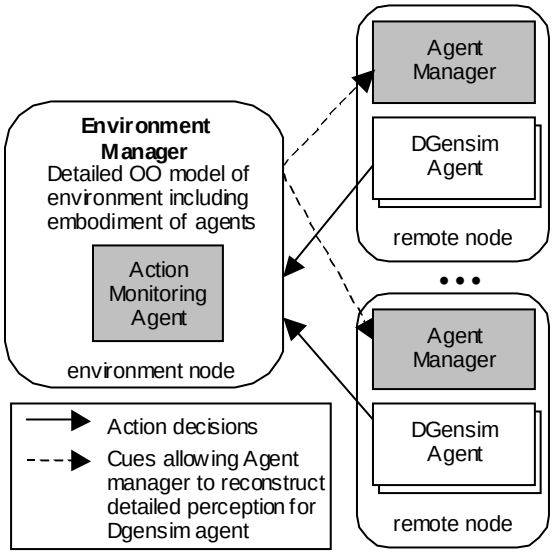

Figure 1. Overview of the DGensim approach.

The primary motivation of DGensim is unchanged from that of the original Gensim system: to provide a generic platform for multi-agent systems research. However, like breadth in a single-agent simulator, there are issues of breadth and generality in a distributed simulator that must also be dealt with. 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 vary considerably (this particular element is equally true regardless of distribution, e.g. [Jinxiong and Sartor, 1994; Fröhlich, 2000]). Similarly, the number of expected agents (e.g. DIS [IEEE, 1993] vs. DDD [Song and Kleinman, 1994]) also varies immensely. More pervasive issues such as the integration of different protocols (e.g. the DIS [IEEE, 1993] and HLA [DOD, 1998] standards) also surround this spectrum. In short, the range of variability in applications is several orders of magnitude larger in a distributed setting. Given that our primary motivation is the support of

intelligent agents and the examination of their interactions, we currently assume a local network. Even with this restrictions, perfect generality is impossible without weakening a tool to the point of uselessness, and so tradeoffs are made as they were in the original Gensim system to balance utility with generality.

This system is in ongoing development using Allegro Common Lisp under Linux. One of our earlier intentions 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 and extension discussed above. The code-data interchangeability inherent in Lisp also, as mentioned above, directly supports the mobility of agents within this scheme as well.

We have recently received a grant from the Canada Foundation for Innovation to explore issues of heterogeneous computing within this framework, in order to increase the ability of this approach to handle issues such as reliability and interoperability in the development of large-scale multi-agent systems. It is hoped that this work will further increase the breath of agents supportable under the more widely varying distributed computing situations discussed above.

Bibliography

[Amin, 2000] Amin, Massoud, "Toward Self-Healing Infrastructure systems", *IEEE Computer*, 33:8 (44-53).

[Anderson, 2000] Anderson, J., "Providing a Broad Spectrum of Agents in Spatially-Explicit Simulation Models: The Gensim Approach", in Gimblett, R. (Ed.), *Integrating Geographic Information Systems and Agent-Based Modelling Techniques*, Oxford University Press, 2000 (to appear), pp. 1-38.

[Anderson, 1997] 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, 1995] *Constraint-Directed Improvisation for Everyday Activities*, Ph.D. dissertation, Department of Computer Science, University of Manitoba, March, 1995. 397pp.

[Anderson and Evans, 1996] 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 and Evans, 1995] 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 and Evans, 1994] 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, 1998a] Balch, T., *Behavioral Diversity in Learning Robot Teams*, Ph.D. dissertation, Department of Computer Science, Georgia Institute of Technology, December, 1998. 204 pp.

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

[Deadman, 1999] 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 and Gimblett, 1994] 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.

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

[Fröhlich, 2000] Fröhlich, Torsten, The Virtual Oceanarium, Comm. ACM 43:7(94-101).

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

[Hanks et al., 1993] 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.

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

[Jinxiong and Sartor, 1994] 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, 1996] 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 et. al., 1988] 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.

[Song and Kleinman, 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.