A Generic Simulation System for Intelligent Agent Designs

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

Appears in *Applied Artificial Intelligence*, Volume 9, Number 5, October, 1995, pp. 527-562.

Abstract

Intelligent agents designed to perform in the real world should by definition be tested and evaluated in the real world. However, this is impossible in many situations: a lack of resources may rule out construction of a complete robotic environment, for example, or the desired domain may be physically inaccessible for testing. In such situations, the use of a simulation system to provide an environment in which to test and examine the intelligent system is necessitated. In the past, such systems have acquired a poor reputation within the AI community, mainly due to the sometimes grandiose claims of systems that are tested solely under simulated conditions. In this paper we explore the conditions under which simulation is justified, examine the inadequacies of currently available systems for the testing and examination of intelligent agents, and describe Gensim, a new system designed to address these inadequacies. Rather than providing a single, parameterized domain, Gensim provides a collection of facilities allowing users to design complete environments for examining and testing intelligent agents. The system also provides a clean interface, allowing widely differing types of agents to be studied. While some bias is unavoidable, these facilities are designed to be as widely applicable as possible.

1. Introduction: Simulation and Intelligent Systems Research

Intelligent agents designed to perform in the real world should by definition be tested and evaluated in the real world. However, intelligent systems continue to be developed using simulated worlds despite this obvious fact: typically, an agent is designed to demonstrate some aspect of intelligence and is tested in simulation, with the assumption that the demonstrated behaviour will scale appropriately to the real world. However, the continuing disparity between the relatively small number of deployed systems and the large number of systems that perform only in simulated environments has come to be one of the strongest and most often cited arguments against the utility and future of intelligent systems (e.g. [Dreyfus, 1981; McDermott, 1981]). This criticism is not undeserved. In many early cases, simple simulated

worlds were chosen innocently, under the mistaken assumption that the differences between such environments and the real world were insignificant. However, many systems are tested in simulated environments that make unreasonable simplifying assumptions in order to avoid issues that would be problematic to the system. Very few of these are deliberate attempts to mislead, but the poor science demonstrated in the unrealistic assumptions made by such systems has led to suspicion of any modern intelligent system that performs in a simulated environment.

The difficulty with this suspicion is that it confuses problems that may arise from using a *simplified* environment with those that arise from using a *simulated* environment. *Simplified* environments can never be like the real world, and only intensive analysis of the assumptions made in constructing these worlds can gauge the applicability of the results obtained from them. A simulated environment need not be an overly simplified model of reality. Indeed, even simplification is not in and of itself a problem. Pollack (in [Hanks et al., 1993]) goes to great pains to illustrate that not only is simplification not a problem, it is as absolutely necessary for experimentation in AI as it is in any science. Simplification is required to isolate a given phenomena in a complex system, and is a critical tool to any scientific endeavour. Any difficulty lies in the lack of analysis of assumptions made when using a simplified world. Rather than qualifying results obtained from such research, general (and sometimes grandiose) claims are often made that are not logically entailed by the research itself.

When (in AI) we state that the environment a given agent operates in is a *simulated* one, it means only that the objects around the agent and the physics of the world itself exist only in the form of a computer program. The agent interacts with information provided by the program rather than the physical objects themselves. This program can be as complex or as simple as desired: simulation does not necessarily imply over-simplification. The suspicion of simulated environments is justified in that in many cases, simplification is hidden beneath simulation (due to the lack of analysis described above). In any truly scientific effort, however, the two issues should be dealt with separately.

Despite this overall suspicion, there are many logical reasons for using simulated environments for testing intelligent agents. The foremost of these concerns the nature of the field of Artificial Intelligence itself. AI is an immature science, and as such it does not yet possess a wealth of broadly accepted theories to form a concrete foundation for ongoing research. The result of this is that researchers in learning, for example, cannot make use of a generally accepted theory of planning to qualify or simplify their research. Similarly, researchers in vision cannot make use of a generally accepted theory of learning. In both of these cases, any overlap from one subfield to another entails making assumptions about unsolved problems in the other

subfield (qualifying and possibly compromising one's own research) or embarking on a number of potentially extremely time-consuming research projects in order to provide a more solid foundation for the original research. Nowhere is this overlap felt more than research involving intelligent agents. Almost any research project in this area involves the integration of a number of areas (e.g. vision, knowledge representation, robotics), with the necessity of solving open problems in those areas or making assumptions about the nature of those problems. Thus, if one develops a theoretical architecture for an intelligent agent, unless one intends to solve *every* open problem in areas such as computer vision and robotics, one is forced to make (possibly incorrect) assumptions about how these various peripheral aspects will operate when more complete theories are available.

The use of a simulator in intelligent agent research is thus largely an issue of practicality: the large collection of interrelated problems encountered when implementing an intelligent agent necessitates the use of a simulator to aid in accounting for those pieces of the theory that are not complete. The focus of any individual research project in intelligent agency may be quite narrow (e.g. demonstrating the utility of a new form of representation within an intelligent agent). In such cases, it is irrelevant (from the point of view of the original research) to provide complete answers to peripheral problems such as sensing, given the effort required to solve these problems. Admittedly, peripheral areas such as sensing are not independent of other aspects of intelligent agency, but can be de-emphasized as long as the assumptions made to deal with them are explicit and reasonable, and that results from such research are interpreted in light of those assumptions. Indeed, Pollack (in [Hanks et al., 1993]) points out that the assumptions we make about peripheral areas can suggest further experiments involving these areas: an important consideration given AI's current stage of maturity. In addition to conserving research resources, simulators are also often required because of a lack of physical resources. Few research facilities can afford the staggering cost of supplying enough robotic technology to meet the needs of all the intelligent systems research projects they support [Etzioni and Segal, 1992] nor the manpower to maintain this equipment.

There are many reasons beyond resource limitations for choosing to use a simulator in the development of intelligent agent designs. A software testbed can, in many ways, provide a great deal more control over the testing environment when compared to testing intelligent systems in the real world. A simulator can present a common environment across many trials, providing exactly the same initial state and planned course of events for each, and can be used to isolate the testing environment from interference from aspects not controllable in the real world [Cohen et al., 1989]. Because of these control abilities, domains can also be saved and shared, allowing simulation environments to serve as a broad metric for comparing intelligent agents [Howe, 1993; Hanks et al., 1993]. Using simulated domains, a long series of

trials can be performed with much greater speed than could ever be expected of the real world. Simulated environments can also duplicate worlds that are inaccessible for testing, or are too dangerous to risk physical loss of equipment during system development. Simulated environments are generally more easily modified than the physical world to suit new testing conditions, and simulators can also be used to create worlds that are more demanding than any physical world. This allows the developer to test an intelligent system at extreme conditions (e.g. a high frequency of problem cases) that might be difficult to stage in a physical environment.

Still other advantages come about as a result of using software over hardware. Current hardware technology causes many problems in demonstrating AI systems: for example, robotic units often break down and cause errors unrelated to the intelligent system under evaluation. Such failures can sometimes occur with great frequency, slowing down the research process considerably [Etzioni and Segal, 1992]. Given that most cutting-edge research will be near or beyond the limit of current hardware technology to adequately support, it will often be more reliable to examine the behaviour of the system under simulation rather than in the real world. Indeed, many laboratories with a wealth of robotic equipment continue to use simulation for precisely these reasons. Etzioni and Segal [1992] also point out that current robotic technology can also severely limit realism: manipulators and visual sensors often cannot perform to the degree that is required in some domains. This necessitates the same kinds of simplifying assumptions for which simulators are criticized. Indeed, many projects involving the use of robotics artificially manipulate and simplify the physical domains in which they operate to cope with hardware limitations [Etzioni, 1993].

This is not to say that simulation is always an adequate substitute for reality: the use of a simulator for evaluating intelligent agents is by no means without disadvantages. Howe [1993] points out that the very ease of use that simulators bring to the testing process can allow researchers to construct experiments too quickly, without the clearly stated hypotheses and careful methodology that physical domains encourage. However, the most obvious difficulty with employing a simulator has already been alluded to: the ability to make simplifying assumptions (even to avoid issues that are impossible to deal with otherwise) can easily lead to making broader, invalid assumptions about the way the world works [Agre, 1988]. This problem has also been noted in other fields where simulation is employed: In general, the control that simulation models allow can also allow researchers to model only those aspects of an environment that they choose to be significant, ignoring other aspects that may have a much greater impact on results than is initially assumed [Friedland, 1977].

In the past, such assumptions have led to an overall lack of understanding of the physical world and how agents operate in it (which in turn have resulted

in many of the limitations of classical planning theories). However, relying on completely implemented peripheral systems, as many recent systems attempt to do, forces much of the overall research effort to be expended on intricacies that may not be the real focus of research. It also forces the reasoning in such systems to operate at a low level, and in general restricts such systems to simple environments: the focus is on making complete simple systems rather than complex systems that leave some peripheral questions unanswered. Systems that include complete sensory and effectory apparatus (e.g. [Agre, 1988; Chapman, 1990; Maes and Brooks, 1990] have shown that the complete implementation approach will work in reactive domains where few if any high-level, long-term decisions are required. However, more complex domains involving extensive coordination and high-level decision-making are beyond the scope of such approaches. Clearly, both approaches that focus on the complete implementation of simple systems as well as those that focus on given aspects of complex systems have their roles to play in intelligent systems research. Both perspectives approach the same puzzle from opposite ends, and each makes its own assumptions. What is important from a scientific viewpoint is that those assumptions be explicit and that results produced be explained in light of those assumptions.

Despite the obvious need for simulation in the design and testing of intelligent agents, the tools currently available for this purpose are less than ideal. Many are useful only for testing particular types of agents, or even one particular agent design. Others provide only one simple domain for testing. Because of this, many intelligent agent research projects (e.g. [Hammond et al., 1990; Agre and Horswill, 1993]) simply develop their own simulation systems particular to the agent and domain of interest, rather than attempt to constrain their research into a form suitable for an existing simulation tool. In our own work, we found this to be unfeasible. Because of the redundancy involved in re-implementing simulation technology for various projects, and the modifications required to cope with changing designs during the evolution of a single project, we desired a simulation system that could be tailored for use with a wide variety of agents in a wide variety of environments. In this paper, we review the specific requirements for such a simulator, and illustrate the problems with existing systems in light of these requirements. We then describe *Gensim*, a simulation system designed to function as a generic testbed for intelligent agent designs, and describe the features that make it uniquely suitable for these purposes.

2. Requirements of a Simulation System for Intelligent Agents

Simulation systems have become an important tool for examining real systems under research conditions, in fields ranging from physiology to natural resource management. A simulator provides a model of a real system, allowing the modeller to study how that real system behaves under conditions of interest, and examining the consequences on the entire system

of a change in some aspect. However, a simulation system for intelligent agents has an additional motivation that brings much additional complexity to such a system. In addition to providing an accurate account of change in a modelled environment, a simulation system for intelligent agent designs must provide a virtual reality for the agents existing in the simulated environment. It must provide a view of the world to intelligent agents through their own perceptual systems, and integrate the actions of these these agents (through their effectory systems) with change from other sources in the simulated environment [Anderson and Evans, 1994].

In a simulation system for intelligent agents, neither of these two roles can be emphasized at the expense of the other. The current stigma attached to using simulation systems in conjunction with intelligent systems development illustrates what happens when sacrifices in the realism of a simulated environment are made. Similarly, if the accuracy of the environment surrounding a collection of intelligent agents is stressed over the accurate interaction between the agents and that environment, the performance of those agents will not reflect reality.

The ability to provide a virtual reality for intelligent agents is a broad goal of simulation testbeds for intelligent agent development. Any individual research project will have many more specific needs, however. We may desire to examine a single agent design across a variety of problem domains, or to examine the performance of a range of agents within a single domain. The particular focus of each research project dictates additional requirements: one may be interested in decision-making in a single agent, for example, or cooperative behaviour between multiple agents. To deal with such a range of interests, any simulation system designed for intelligent agent testing must be generic: it should make as few assumptions as possible about the nature of the environments it is modelling, and the agents that inhabit those environments. It is impossible to completely eliminate all such assumptions, simply because agents and the simulator must communicate with one another in order to provide perceptual information to agents and to inform the simulator of the agent's decisions. Such communication mechanisms constrain the design of agents and the design of the simulator itself. However, like all other assumptions, their impact should be minimized.

In addition to placing as few constraints as possible on potential agents and environments, a generic simulator must endeavour to provide the features necessary to support the wide range of projects in intelligent agent research described above. In order to support testing of various agent designs across a wide variety of environments, for example, a simulation system for intelligent agents must be modular. Each agent, ideally, should be a completely separate computational process from the rest of the simulator, and should have an identical interface, so that agents can be easily interchanged. Such a simulation system must also be able to support multiple intelligent

agents in a single environment. While many projects involve the reasoning methods of only a single agent, few realistic domains are exclusively single-agent domains, and one common criticism of an intelligent agent design is a lack of provision for reasoning about other agents. Thus, the ability to support multi-agent simulations is crucial for a generic simulator. Even though much of the responsibility for coordinating actions between multiple agents rests with the agents themselves (e.g. having the ability to reason about how other agents may interfere with one's actions, and preventing and correcting for such interference), special support for multiple agents must be implemented in a simulator as well. This includes aspects such as separate storage space for the knowledge of multiple agents and the ability to handle change from several different agents at the same time.

Given that a generic simulator is to be used to run an arbitrary collection of agents in a given domain, the relationship between agents and their simulated environment must also be clear and explicit in order to ensure the accuracy of the simulation. Among other things, any assumptions about the timing of actions, chains of causation over time, sensory abilities, and way in agents communicate (e.g. refer to objects) with the simulator must be clearly understood by modellers to ensure accuracy and utility.

A lack of precise description of such features has traditionally been a problem in simulation systems for intelligent agents. Details of communication between an agent and simulator are often sketchy, as are assumptions made by the simulator about the internal operations of agents and other parts of the domain. Indeed, basic concepts such as what the simulator considers an "action" or an "event" to be is often left to speculation. This makes it exceedingly difficult to rate the suitability of a simulator for a given application.

The interface between the agent and the environment should also be as simple as possible, to reflect the real world (and rely on fewer assumptions about the domain and agents). The input of the simulator (output of primary agent processes) should ideally be the direct actions that the agent is performing on each object in the environment (e.g. grasp teapot; lift teapot). The agent will have its own expected set of results for each action it is carrying out, and the simulated environment confirms or refutes these expectations and carries on a simulation of a causal consequences of the actions of the agent. For example, the agent may pick up the teapot and expect the teapot to be in its hand. The simulator, upon receiving the information that the agent is carrying out this action, can then update the environment accordingly: depending on the physics defined for the environment, and the particular conditions in effect at the time, the teapot may or may not actually wind up in the agent's hand. The agent can then be informed of what actually occurred.

Finally, a generic simulator should be able to control many aspects of the functioning of the environment. The actual domain features that must be controlled vary from experiment to experiment, but a generic simulator should provide for control of elements that are common to many domains (e.g. how much sensory information is presented to the agent, how often the agent receives such information, how often the environment may be altered).

These criteria represent the basic requirements of a generic system for testing and examining designs for intelligent agents. While they were arrived at through the analysis of the the needs of our own ongoing research projects [Evans et al., 1992; Anderson, 1995], we believe these projects to be representative of current research trends in intelligent agency, both in terms of their general objectives and their requirements for simulated environments. Other suggestions for requirements exist: for example, [Hanks et al., 1993] review specific issues in intelligent agent research, with the implication that simulation must look toward providing support for such issues. Many of these points are subsumed by the very fact that such a simulator must be as generic as possible. For example, supporting a wide range of environments entails several of the requirements of Hanks et al.[1993], such as providing for external events and using a well-defined model of time.

3. Existing Simulation Systems

In recent years, several simulation testbeds have been developed for the purpose of testing and examining intelligent agent designs. These include: Phoenix [Cohen et al., 1989; Howe and Cohen, 1990], a generic simulator that is used as part of a larger system for controlling fire-fighting agents; Tileworld [Pollack and Ringuette, 1990], a grid-based system for examining the performance of intelligent agents in domains where the stability and importance of goals varies; Mice [Durfee and Montgomery, 1989; Montgomery et al., 1992], a multi-agent testbed also based on a grid-like structure; and Ars Magna [Engelson and Bertani, 1992], a sophisticated system which attempts to accurately simulate the functionality available in current robotic technology.

Each of these simulators is in itself a very powerful system, and each has particular strong points that make it well-suited to some aspect of intelligent agent research. Mice, for example, provides highly specialized facilities for multi-agent interaction, is highly programmable (thus providing for a wide range of control over the environment), and provides a concise interface between agents and the simulator. Tileworld, while being somewhat less general than Mice, provides an even more easily modified environment, in that it is a single grid, with temporary holes the agent is expected to fill by moving tiles, rather than a programmable environment. The system provides various parameters (e.g. the length of time a hole will exist) that allow the user to vary the environment. The system has also been

demonstrated to be of use in several domains, and has been used outside its original development group for the testing of problem-solving algorithms [Kinney and Georgeff, 1991]. Phoenix provides facilities that have allowed its developers to construct a very complex fire-fighting environment, both in terms of the interaction of multiple agents and the number, kind, and distribution of objects in the environment. The system also allows agents to have differing, limited perspectives of the complete environment. Finally, Ars Magna provides a toolkit for constructing complex environments that allow simulation of current robotics technology. The system provides for limiting the perceptual abilities of agents, and has a large number of tuneable parameters.

Despite being very useful in particular environments, none of these systems come close to meeting the requirements defined in the previous section. Phoenix, for example, is a specialized system developed to support a particular agent design and a particular environment. While its authors claim that the system is useful in other areas [Cohen et al., 1989], it would seem to be difficult to adapt the system to any environment that did not bear a strong resemblance to its fire-fighting domain. The system also provides only limited multi-agent capabilities (a centralized form of control), a very limited number of adjustable parameters, and no programmable facilities for constructing a new domain. Ars Magna has similar difficulties. While providing a large number of facilities for constructing variations on a mobile robot domain, it is not clear how one could begin to implement other types of domains using this tool, even though its authors claim that the indoor robot simulator could "probably approximate" other robotic environments as well [Engelson and Bertani, 1992]. While the systems interface functions are well-described, the relationship between agents and the simulator is not. The internal operations of the simulator are also poorly described, making an attempt to construct a different variety of domain difficult.

Tileworld and Mice also suffer many difficulties. Tileworld's tile-shifting domain, while providing many tuneable parameters, is very closely tied to the simulator itself, making any attempt to implement some other domain using the tool formidable. The domain is also much simpler in terms of its overall structure than the domains supported by either Ars Magna or Phoenix. Part of the desire for choosing a simple domain (from the point of view of the interaction of agents and the environment) such as tile-shifting likely arises from the desire to avoid perceptual complications. However, this greatly biases the types of agents the system can implement, as it is impossible to provide domain controls for such critical features as the information-gathering abilities of agents, because the domain is too simple to require any. Tileworld also supports only a single agent, and the interface between the simulator and the single agent it supports is also not described in detail.

Mice, while being well-suited to multi-agent domains, is almost entirely concerned with the interactions of agents, to the detriment of the rest of the environment. The system is severely limited in supporting dynamic events, for example, because the only changes in the environment are those made by its intelligent agents. The semantics of Mice domains are also awkward: each action and event is considered to be discrete, regardless of the amount of time that action or event requires. Thus if an agent moves from one square to the next, and that movement takes four time units, the agent completes the move in the first time unit, and then must wait three more time units until something new can happen to it [Montgomery et al., 1992]. Mice is also heavily biased toward multi-agent coordination experiments: most of the events that can occur in Mice environment, for example, involve what happens when agents bump into one another, or "capture" or "link" other agents (these are generic terms with specific meanings in a particular domain). Mice domains are also fairly simplistic: while they may be spatially complex, the number and types of objects in environments implemented using the tool is generally small (e.g. tiles, nondescript symbols representing predators and prey). Agents also do not have the kinds of information-gathering abilities necessary to simulate real world environments.

Each of the above systems has many individual strengths and weaknesses, only the most significant of which are described here. More information on each of these systems from the standpoint of testing intelligent agent designs may be found in [Anderson, 1995]. Hanks et al. [1993] also describes many further difficulties with the Tileworld system in particular. Considered broadly however, these simulators may be divided into two groups with respect to their abilities to serve as a generic testbed for intelligent agents. Some systems offer complex simulations of specialized domains (e.g. Phoenix, Ars Magna). Such simulators are often designed as part of a specific agent architecture or domain implementation, and serve well in that capacity. However, they generally prove inadequate when an attempt is made to adapt them to a new domain or to a new type of agent. Other simulators (e.g. Tileworld, Mice) are more general, but provide a great deal of variation on one (usually simple) domain. These simulators are adequate for testing simple strategies in problem-solving, but are inadequate for representing more complex domains.

While each of these simulation systems is well-suited to some environments, none provide the features necessary to make them truly generic. Adapting any of these simulators to an agent design or domain that it is not biased toward would likely require a major effort. This is unsuitable from the point of view of developing intelligent agents, as well as from that of studying simulated domains. Agent architectures and domains are highly likely to change during development, sometimes drastically, and if a simulator cannot provide support for such changes, users are likely to spend most of their time adapting one system or another to their changing designs. The only

alternatives currently available to this are adapt one of the many special-purpose simulators that have been developed in conjunction with particular agent designs (e.g. [Hammond et al., 1990; Agre and Horswill, 1992]), or to make use of other forms of simulation, such as qualitative physics systems (e.g. [Kuipers, 1986]). Taking the former route involves exactly the same problems as adapting the pseudo-generic systems described above. The latter choice, on the other hand, provides detailed simulation of a particular environment, but none of the specialized features necessary to integrate intelligent agents within a simulated environment.

The difficulty inherent in applying existing simulation systems to the task of examining a broad range of agents in various domains has led to the development of *Gensim*, a generic software testbed for intelligent agents that addresses all of the criteria presented in Section 2. The remainder of this paper describes the Gensim system in detail. The primary motivation in the development of this system was to provide simulation for our own intelligent agent research: after examining the prospects of modifying the systems described above, it was decided that it would be more fruitful to design a new system that could be used for ongoing development of agent architectures and domains in which to examine these architectures. The study of existing systems also pointed out an obvious need for such a generic testbed: a system satisfying all the requirement presented in Section 2 would bridge the gap between the two classes of simulators described above, providing a basis for future research in the design and evaluation of intelligent agents.

4. Gensim

Gensim is a Lisp-based object-oriented simulation system designed explicitly for the testing and examination of intelligent agent designs. The system supports multiple agents, each of which may consist of multiple timeshared processes, has a concise interface between agents and the rest of the environment, clearly defines all restrictions and assumptions regarding the agent-simulator relationship (e.g. action timing, perception) and provides the ability to control many aspects of the simulation itself. Gensim also has a modular design, making it simple to take the basic simulator and program any additional features required to support a given agent or domain structure.

Rather than simply providing a flexible domain, as is done in Ars Magna or Tileworld, Gensim provides flexibility in the simulation process itself. That is, rather than providing extensive domain parameters, Gensim provides the parameters and functions necessary for the user to define their own domain and interfaces for agents. This requires greater initial effort on the part of the user, in order to define or modify a domain, but makes Gensim much more widely applicable than any of the simulation systems described in the previous Section.

A high-level overview of Gensim is illustrated in Figure 1. The simulator itself manages the environment, represented as a collection of objects. This environment is objective: it is the universal set from which all intelligent agents' perspectives are defined. The simulator also possesses a collection of procedural knowledge describing the actions that agents can perform on these objects, as well as the physical events that can occur to these objects outside of the influence of any agent.

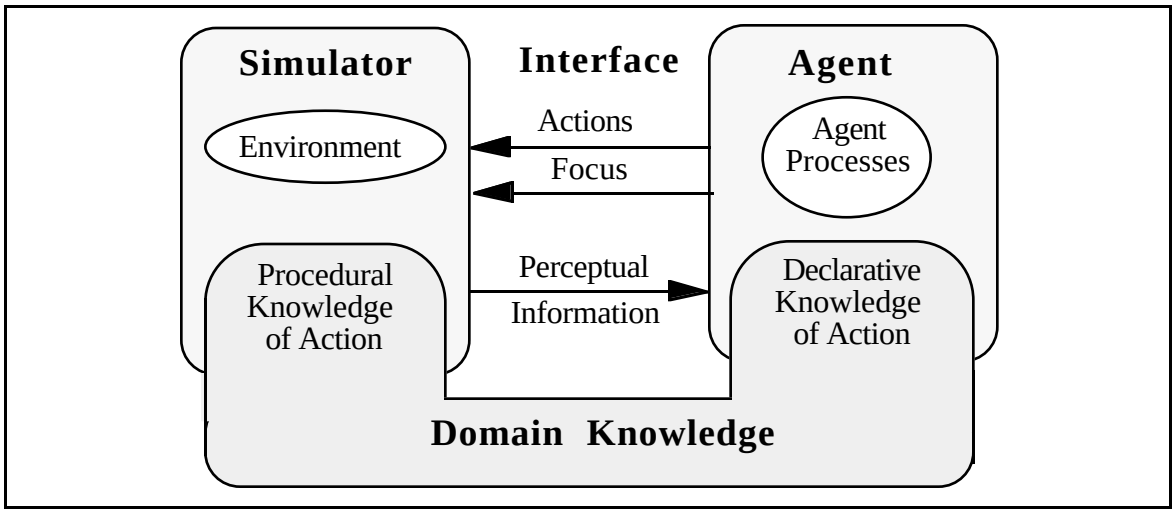

Figure 1. The high-level structure of Gensim.

A collection of agents is also defined, each of with its own view of the environment based on its sensing ability and memory. The reasoning abilities of an agent are implemented as a set of timeshared processes, which collectively allow the agent to perceive the environment around itself and act on the basis of those perceptions. As each action is carried out by the agent, its effects on the world (both short-term and long-term) are manifested by the simulator. Agents are viewed as "black boxes" by the simulator, in that Gensim neither knows nor cares how the agents arrive at their decisions for action. As indicated in Figure 1, some commonality in domain knowledge is required in order for agents to interact with the simulator. For example, the actions that the agent can select from (represented largely in declarative form) must be identifiable by the simulator, which then uses its own (largely procedural) knowledge of action to modify the environment appropriately. Agents also inform the simulator of their interests in objects in the environment (their *focus*) in order to provide appropriate sensory feedback. However, emphasis is on making agent's knowledge distinct from that of the simulator, and the illustration in Figure 1 should not be interpreted as implying that agents somehow physically share the simulator's knowledge. Rather, the simulator provides an objective description of the domain, consisting of all the objects in the environment (including the agents themselves), and causal knowledge of how those objects interact with one

another. Each agent maintains its own (usually limited) perspectives of this environment, much as the real world operates.

The basic design goal of Gensim was to support as wide a variety of agent and domain designs as possible. To this end, Gensim was designed to keep agent and simulator knowledge as separate as possible, and thus limit the knowledge each must have of the other. A complete environment is defined for the simulator, and agents are expected to have their own knowledge of the objects in the domain and how they operate. This is more complex than simply allowing an agent to share the same internal objects manipulated by the simulator, but allows much more flexibility in implementing complex domains. In particular, it allows agents' perspectives to differ, and limits agents' knowledge to that information the designer of the domain wishes them to possess.

In spite of this design goal, it is impossible to ensure that a simulation system can support *any* domain or agent design. Certain basic design concessions must be made in order to use a domain or agent design with Gensim, just as concessions must be made for any two systems to operate together or communicate with one another. In particular, the domain must be organized to obey the timing principles on which Gensim is based.

4.1. Timing in Gensim

As mentioned in Section 2, the operation of the intelligent agents in a given environment should proceed in parallel with the changing environment. While this would be ideal, Gensim is currently implemented on a serial machine, and thus must simulate parallelism using timesharing. Gensim supports multiple agents, each of which may consist of multiple processes, and performs its own timesharing of those processes. The simulation of agent operations is done by continuously cycling through each process of each agent. As agent processes run, they collectively perform actions and make requests for perceptual information from the simulator. After each process of each agent has been executed, the simulator updates the environment based on the actions of the agents (and other independent events), prepares sensory information for each of the agents based on this modified world and a focus description from each agent, and cycles through the agents again.

In simple simulators (e.g. [Agre and Horswill, 1992]), the basic time interval around which the system is organized is often the time it takes an agent to decide on an action to carry out. Each action in such a system is thus based on information gleaned from the world at a single point in time. In the real world, however, sensory information arrives continuously: by taking its time, the agent can get many views of the environment on which to base its decisions for action. The agent pays for this ability proportionally through the risk of unanticipated and unrecoverable change in the world around itself.

Gensim supports this by organizing the system's timing around the interval at which sensory information is presented to the agent. For the purposes of discussion, we will call this interval P.

By designing the system around the time required to gather information about the world, an agent can be "interrupted" with new information about the world every P seconds. This interval is one of the standard tuneable parameters in a Gensim environment, and is static throughout a simulation. The amount of time a given agent is allowed to deliberate its choice of action (the number of P intervals) is not limited by Gensim in any way. Each P interval gives the agent further sensory information, and the agent can deliberate for as many intervals as desired, facing the consequences of change and missed opportunities in the simulated world. This discrete sensing interval is not unrealistic. Human vision, for example is not as continuous as it might seem on the surface. Experiments have shown that change over intervals of approximately sixty milliseconds or less are perceived as continuous by human vision [Graham, 1965]. So long as a simulator presented sensory information at a rate greater or equal to this, a series of discrete "snapshots" would be indistinguishable from a continuous process by a human and could be accurately treated as such. This same principle can be applied to computational agents.

In addition to the sensory interval P, a simulation system for intelligent agents must be concerned with two further timing cycles. The first of these is A, the rate at which an agent commits to actions. In any realistic environment, A cannot be regular: an agent must be allowed to react immediately or take as much time to deliberate as it desires. The other is E, the rate at which changes are made to the environment. In the real world, change occurs continuously, but by the same argument used when defining the P interval, a small discrete interval is indistinguishable from a continuous process by a rationally-bounded agent. In an objective simulation of a given environment, this E interval directly affects the accuracy of the simulation. By the principles discussed above, the longer the E interval, the less continuous the simulation will appear to be. However, since in the case of Gensim the simulation exists strictly for the benefit of one or more agents, no change need occur except at those times when information is presented to those agents. Because of this, E is a fixed length interval equal to the shortest agent P interval in a particular simulation.

In an ideal situation, agents and the environment would proceed in parallel, in real time. Agents would deliberate continuously, committing to actions at irregular points in time, and the world would proceed in parallel, interrupting the agent with new sensory information every P seconds. In Gensim, however, two complications arise. First, Gensim is implemented on a serial machine, making it impossible for the agents and environment to run in parallel. Also, while agents must operate in real time, an

environment may consist of many hundreds or thousands of objects, and it may simply be impossible to maintain the state of all those objects in real time. Restrictions could be made to force the simulator to operate in real time, but these would at the same time severely restrict the range of environments that can be implemented.

These complications are dealt with in Gensim through timesharing, allowing each agent process to run for precisely the length of one P interval. After each process has been run, the environment is updated (based on the actions of agents as well as independent events that have occurred). A new set of sensory information is then provided, and agents may continue their own processing. This timing (for a single agent with a single process) is illustrated in Figure 2. Here, the agent processes are run repeatedly, each time for the length of one P interval. An agent may carry out an action during that time, or may deliberate for a number of such intervals (e.g., the first action the agent takes is deliberated upon for three intervals, the second for one interval, etc.). After each interval, the environment is updated. These E intervals are regular with respect to simulation time (in that each represents the changes that occurred during the previous P interval) but are irregular with respect to real time: the environment may change significantly one cycle, and only trivially the next.

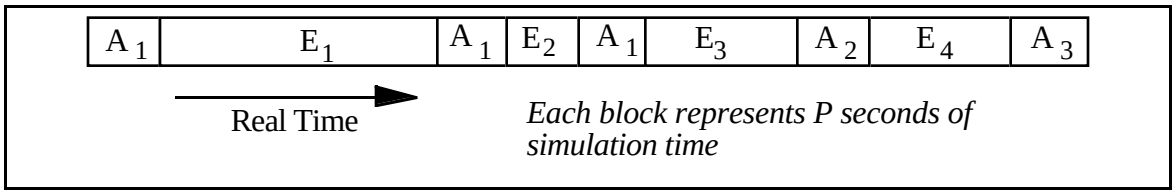

Figure 2. Timeline showing interleaving of agent/simulator processes.

The next time the agent process is run, it has new sensory information available to it based on the environmental changes that occurred The agent may, of course, simply choose to ignore this new information and continue on with what it was doing previously. Agent processes are implemented as LISP functions, and users can explicitly specify that a given process is to run for some fraction or multiple of an P interval if this basic timing is not sufficient. Figure 2 shows only one agent interacting with the simulator; however, the same basic timing applies to multiple agents, or agents with multiple processes. All are granted specified time-slices, representing one single unit of time being shared by all agents, following which the simulator updates the environment appropriately.

We believe this timing to be adequate for most domains. It can be criticized, however, on one significant point: since the processes of an agent (and indeed, the agents themselves) are serially timeshared, they cannot simulate some of the interactions possible in truly parallel processes. For example, an

agent might consist of two processes: one to recognize objects in which it has an interest, and another to decide what to do based on the objects it has seen. When these run in parallel, the acting process would process input from the recognition process as it became available. This is a much more complex interaction than that possible under timesharing, where the recognition process must recognize all the objects it can in a given interval and then pass them all to the acting process. This is a legitimate criticism, but like the accuracy of the sensory interval, the effects of this depend entirely on the interval size. Like other applications of timesharing (e.g. operating systems), the smaller the time-slices involved, the more transparent the timesharing will appear.

4.2. Managing Environmental Change

Now that the basic timing of Gensim has been described, the operations that occur during agent and simulator time cycles can be examined. As stated in previous Sections, the primary purpose of a simulator is managing change in a virtual world. In Gensim, this world is modelled using an independent frame-based system developed by the authors. Each physical object in the domain is described by a frame [Minsky, 1980], and these frames are organized in a hierarchical fashion. The system also supports message-passing, procedural attachment to frames, a form of multiple inheritance, and the use of multiple frame hierarchies. The ability to manage multiple frame hierarchies is critical in a multi-agent simulator, since the simulator must keep track of the objective world and allow agents to maintain their own models of the world using this same method of representation if they choose to do so. Alternatively, each agent can also make use of any other knowledge representation system it requires: using the internal frame system is advantageous in that it allows simpler interaction with the simulated world, but is not required. The various attributes recorded for each object in the simulated world will, of course, differ by domain. However, the system assumes that each object has a given `LOCATION` attribute, and that the locations of all objects are recorded in a similar format. Since the hierarchy is organized primarily by class, retrieval by class is always an efficient process. We recognize that retrieval by location may also be common in some domains, and provide the option of indexing objects by location.

Change in the simulated world is managed through two facilities: agents perform *actions* that can alter the world; and *events* (which may or may not be independent of an agent's actions) may also occur. An event is the occurrence of change in the domain from an unspecified source. For example, a ball may hit a wall and bounce off; a toaster may pop; or the wind may knock something off the shelf. Note than none of these examples is *directly* attributable to any agent: for example, the ball may have been thrown by an agent, but from the point of view of providing an accurate physics for the domain, this no longer matters. An event is defined as a block of

environmental changes that occur over a simulator interval. Events may propagate over time into an *event series*, which allows the simulator to represent continuous change over time. The simulator manages an *event queue*, each entry of which consists of a point in time at which an event is to occur, and the name of a routine that will cause the desired changes in the domain to be manifested. Functions are provided for creating random elements in an event, and for spawning new events in the queue from within an event (creating an event series).

Events are defined procedurally and are attached to the domain objects they affect (this implementation will be described in greater detail presently). A ball, for example, may have a `TRAVEL-THROUGH-AIR` event defined for it, which moves the ball through the air in a given direction at a particular speed. During the length of time the event runs, it may reduce the speed of the ball and check to see if it hits something in the environment. In either case, the `TRAVEL-THROUGH-AIR` event will insert a new event in the queue (to move the ball further along the next time the simulator runs, or to make the ball stop or bounce if it has hit something). Events may interfere with one another: in the above example, a ball might strike another ball and divert it from its path. Once again, this is part of the physics of the domain to be simulated, and routines are provided to dynamically insert, modify, re-order and delete event queue entries to allow the user to implement this physics.

The accuracy of a domain depends a great deal on how events are implemented. Since each update of the world represents a specific time interval, and since that interval can change from run to run, an event should be designed to utilize the time factor involved rather than simply performing the same discrete operation regardless of how much time is used. For example, an event might move an agent one "unit" in a given direction on a grid, or it might move an agent at a given velocity for the amount of time available. The former results in a poor simulation should the unit of time vary from run to run, while the latter is unaffected. As mentioned previously, the amount of control allotted to the user by a simulator such as Gensim makes it possible for poorly-designed simulations to be designed as easily as well-designed simulations. It is up to the user to use the facilities provided in a manner appropriate to the domain and to the degree of accuracy desired.

Actions are closely related to events. Unlike events, change induced by an action has an explicit source: an action is performed by an agent with the intent of accomplishing some objective. Actions in Gensim consist of three components: an *intention* component, representing the agent's internal justifications for and expectations of the action; an *agent-causal* component, consisting of the immediate physical actions the agent performs in order to bring about it's intentions; and a *domain-causal* component, consisting of the changes that occur later in time due to time delays or physical interactions

with other objects in the environment. Thus a ball being thrown by an agent and breaking a window consists of the agent's intention to throw the ball (the reasoning and justification behind its decision), the actual arm motion that causes the ball to be thrown, and the course of the ball, culminating in its collision with the window. The agent knows only of its intention and the motions that it itself carries out; the agent's knowledge of what happens after the ball leaves the agent's hand is entirely dependent on the agent's perception (seeing what unfolds) and/or the accuracy of the agent's prior knowledge of the physics of the domain (predicting what will happen).

The simulator is concerned with manifesting the physical effects of an action on the environment. Thus, actions within the simulator itself are represented in two parts: the immediate changes that would have taken place during the cycle in which the agent performed the action, and a series of future events the action sets in motion. These are inserted into the event queue as the simulator manifests changes based on the action. An action can cause any number of future events to occur: however, each of these events must be independent of the others. For example, an action like `THROW` performed by an agent might cause the ball to travel through the air and eventually through a glass window. The `THROW` action cannot set up this series of dependent events. Rather, it must simply cause the ball to leave the agent's hand in the current time interval and have some velocity in a given direction. The implementation of the action can then set up a `TRAVEL-THROUGH-AIR` event, as described above. This event can in turn propagate itself, and the resulting sequence of events implements the desired behaviour. A sequence of such events is shown in Figure 3. An agent throws a ball, which travels-through-air during a number of intervals, hits something and bounces, and will eventually come to a stop. Note that only one of this series of events will be in the event queue at a time, since each one spawns the next. This sequence is also non-deterministic: as the future unfolds, existing scheduled events can be modified or cancelled by other actions and events through the use of the event facilities described earlier.

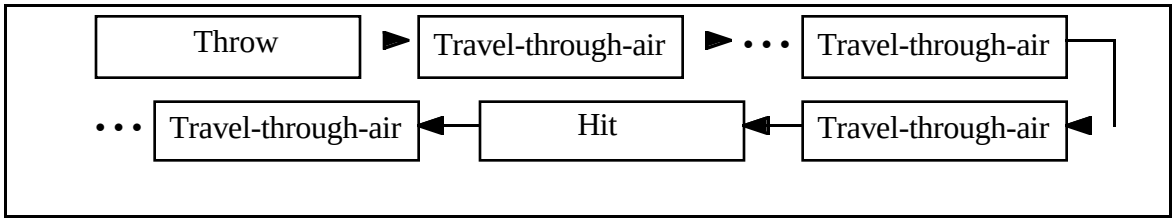

Figure 3. An action spawning events in the event queue.

As per events, the representation of actions described here is strictly the perspective of Gensim, and *not* that of any particular agent. Each agent can and generally does have is own representation of the actions it can perform, and will naturally have its own expectations of what carrying out the action

will do to the world. This knowledge is stored in its own separate knowledge base, along with other forms of knowledge about the domain. The actions defined as part of the environment, on the other hand, define the objective physics of the domain independent of any agent. As agents carry out actions during their own time-slices, they communicate the actions (names and parameters) to the simulator. The simulator then updates the environment by invoking its own procedural representations of these actions through message-passing.

Agents use actions (and indirectly, events) as mechanisms for change in the world. As each agent process is run, an agent performs actions and records their performance in a data structure accessible to the simulator. After all agents have been run in a given cycle, part of the task of the simulator is taking all these actions and updating the environment based on their effects (e.g. altering objects, inserting or modifying future events). Some effects will be as the agents anticipated, while others (due to random ill effects, misinformation on the agent's part, or interaction with other actions on the same cycle) will not.

The manner in which the actions of agents are viewed by Gensim is very different from that of previous simulation systems. In the past, simulators have often been described as "carrying out" or "executing" the actions of an agent: such phrases invite an inappropriate comparison between the agent-simulator relationship and the relationship between and classical planner and its executor. The real world has almost no comparison with such an executor, and neither should any simulator intending to model it. The real world is not a servant that carries out the commands of a disembodied agent. Rather, the agent physically participates in an ongoing interaction with other objects that collectively constitute the world. A simulator, it follows, should provide an environment that can be changed, rather than serving as a method for accomplishing this change.

In keeping with this view, Gensim does not allow agents to "instruct" the simulator in any way; rather, agents commit to and carry out actions during the time periods in which they are active (i.e. during A_i time-slices shown in Figure 2). When an agent commits to an action such as "Throw the ball", it is *not* viewed as a "throw" instruction that is carried out by the simulator, returning some result to the agent. Rather, the agent is viewed to have carried out the agent-causal portion of the action during the previous A_i interval. The simulator, having been given knowledge that the agent has performed this action, in turn alters the affected parts of the environment during the time interval in which it is active. That is, the simulator changes the environment based on the agent-causal portion of the action, and continues the changes indicated by the domain-causal portion of the action over time.

This distributed view of action, placing control of the agent-causal portion within the agent and the the domain-causal portion in the simulator, emphasizes the dual nature of action: an agent will always have some responsibility, with the physics of the environment providing the rest. It also takes care of many semantic problems that are evident in previous simulators. In a timeshared simulator, when an agent "instructs" the simulator to perform some action, not only is it difficult to place a locus of control for the action, it is also difficult to provide accurate timing for the action. When can an agent assume that the action has occurred? When do results come about? Disregarding questions such as these lead to the awkward action/event processing of simulators such as MICE. Since the action itself is entirely contained within the bounds of the simulator in such primitive models, the agent itself really has nothing to do with carrying out the action. It simply decides that it wants something to happen, and the simulator magically makes it so. Gensim, on the other hand, allows agents to be viewed as active participants in ongoing interaction with the world, rather than as passive decision-makers.

Emphasis on event-based processing is a relatively recent phenomena in AI, stemming mainly from recent work in reactive architectures. Because of this, most well-known theories of action concentrate on reasoning about specific actions in plans, rather than the connection between an action and future events. However, the view of actions and events used by Gensim does have similarities to some more modern theories. In particular, Lansky [1986] presents a view of change as composed of sequences of events. Agent reasoning about change then becomes reasoning about the history of events that have occurred. However, this model concentrates on agent's internal reasoning about change, rather than actually modelling action in a simulator. Since agent's views and representations of actions are independent of the Gensim model, an agent can use this or any other representation internally to reason about action.

This view of activity is also similar to some approaches in temporal reasoning. For example, McDermott [1982] represents event causation using the predicate `ECAUSE(P,E1,E2,RF,I)`, with the following semantics: event `E2` follows event `E1` after a delay in interval `I` unless `P` (a series of exceptions) becomes true. `RF` is an real number representing where in the time interval of `E1` the delay begins, with 0.0 indicating the beginning of `E1` and 1.0 indicating the end. This performs well theoretically and is useful from the standpoint of analyzing causation (again, a function internal to agents not normally useful in a simulator). However, much of the representation is impractical. For example, consider the requirement of knowing of the length of `E1` in order to implement `RF`: clearly we cannot know how long `E1` will be until it is completed. McDermott's approach also omits some implementation-related aspects (e.g. multiple effects, links between actions and events, variable `I` intervals) provided by Gensim.

4.3. Perception

In addition to performing actions, agents also interact with the simulated world through perception. Perception is the most difficult design aspect of any simulator for intelligent agents: on one hand, it is a crucial aspect of virtually any domain, while on the other, simplification of perception is one of the primary motivators for using a simulator, and the greater the simplifying assumptions, the more limited the simulator will be.

The perception abilities of intelligent agents are much more active processes than might first be assumed. We do not simply see whatever our eyes happened to be directed toward, for example: we pick and choose from the image, discerning what interests us. What we have already seen also alters further perception, helping to define a focus for perceptual efforts. In more computational terms, a set of *anticipatory schema* (domain-specific knowledge indicating the sensory information in which the agent has an interest) acts as a filter for the vast amount of knowledge available from the world. This knowledge directs the agent to explore the world, looking for given pieces of information. This exploration directs the agent to specific information (objects) in the environment, which then modifies the anticipations the agent has for future sensations [Neisser, 1976]. Within this cycle, one of the basic problems of implementation is that an agent must be allowed to specify its interest in given aspects of the environment, but must also be given access to information independent of those interests. Perception must exist to confirm the agent's expectations of the world, but also must provide the agent with new information (not necessarily what it expects to see or is directed toward) in order to form the expectations of the world that guide its sensory abilities [Niesser, 1976].

Gensim attempts to follow this model as closely as is practical. However, some aspects of perception are simplified, in order to simplify the simulator itself and to conform to one of the basic design goals of Gensim: a simple interface between agents and the simulated environment. One of the major simplifications concerns the internal processing within this cycle. The cycle described above is based on a hierarchical model of vision [Goldstein, 1984], with the environment itself dictating bottom-up processing (processing a visual image into various edges and features), and domain knowledge possessed by the agent directing top-down processing (processing from features to objects using expectations). Contrary to this model, perception in Gensim is defined at the object level. That is, an agent senses a combination of complete objects and specific sensory aspects of those objects (e.g. a ball, or the fact that it is red or round), rather than examining the individual edges and features that make up an image of the object itself.

This partially invalidates the hierarchical model, in that we gloss over the lower-level feature processing that allows the agent to distinguish objects.

However, removing the complexity of low-level vision from the design of an agent is comparatively the most computationally significant sensory simplification that can be made. In addition, some of the essential character of the hierarchical model of vision is still preserved, in that the information presented to an agent after each P interval is selected based on interests expressed to the simulator at the end of the previous P interval.

The ability of an agent to express a focus for its sensory abilities is provided in Gensim through a sensory request mechanism. This mechanism allows an agent to express interest in certain aspects of its environment, and the methods by which Gensim fulfils sensory requests allows the agent to receive limited sensory information not only about its focus of interest, but also about objects outside that focus. This provides the balance between anticipation and exploration characteristic of the hierarchical model described above.

Sensory requests operate in much the same manner as actions, as described in Section 4.2. An agent makes a sensory request for some particular set of information, and this request is recorded by the simulator. After all agent processes have been run on a particular cycle, the simulator updates the environment according to the actions and events that have occurred during the interval, and then prepares sensory information based on each agent's requests.

Two possible sensory requests are implemented in Gensim (additional facilities may be defined as agent actions). An agent is allowed to explicitly gather information about a given object through an explicitly defined `LOOK` request. An agent can also `SCAN` in a given direction (recall that directions and locations are part of the definition of a domain) to get a general overview of objects in its vicinity.

Part of the definition of a Gensim domain is the definition of domain-specific aspects of `LOOK` and `SCAN`. Like actions, the knowledge required to fulfil sensory requests is defined in procedural form as part of a domain definition and attached to the simulator's knowledge of the agents themselves. The ability of an agent to `LOOK` and `SCAN` can differ from agent to agent. For example, the procedural implementation of `SCAN` for one agent may result in descriptions of a given number of objects of a certain size (i.e. the agent sees large objects first). A differing implementation may have the agent see all objects within a given distance. The actual methods employed by Gensim to describe objects to agents are described in the next Section.

Each agent has a finite bandwidth (defined as simulator's knowledge of the agent), implying that they can perceive only a certain number of objects at a time. In addition to this bandwidth, each agent has a limitation on the number of `SCAN` and `LOOK` operations it can perform in a cycle. The abilities of scanning and looking can also be made mutually exclusive to one another within a cycle. These limitations can be used to enforce bandwidth

information very strictly, or to have a fairly loose concept of bandwidth, depending on the domain. If any bandwidth is exceeded (e.g. if there are three looks, and the first two present all available objects, or if there are the number of looks and scans exceeds the limit), the information is simply not made available to the agent. Note that these bandwidth limitations implement the ability of senses to return information to the agent, *not* the ability of the agent to process that information. The timing of agent processes may enforce a much stricter limitation on the ability to process sensory information than the sensors have to provide it.

Information outside of the sensory focus defined by an agent's requests is provided by filling any remaining bandwidth with objects according to agent-specific biases. An agent may be biased toward perceiving large objects before small ones, or colourful objects before dull objects, or in the case of more sophisticated agents, objects directly connected to the agent's ongoing activities before less useful objects [Anderson, 1995].

In addition to providing visual abilities, we must recognize that perception often involves integrating information from several senses [Neisser, 1976]. This is a point often ignored by simulation systems, but is addressed in Gensim through two additional components to perception. First, an event may cause some noise or other obvious disturbance in the environment. This can be represented using a `SIGNAL` command (e.g. (`SIGNAL PHONE-RING SOME-LOCATION`) is used in one current Gensim application [Anderson, 1995] to allow agents to perceive a ringing telephone). Signals are passed as is to agent processes when they are run; therefore the agent must have some concept of what the simulator is signalling (in this case, what a "bang" is). Once again, a limit can be indicated for the number of signals an agent can accept in a cycle, with additional signals being lost. This signalling ability allows limited non-visual perception, and in many domains may be used to reinforce visual perception rather than as an alternative to it.

Finally, limited sensory information at a level higher than the basic object level is available. When an agent carries out actions, it is possible when updating the environment based on those actions to signal the fact that an error has occurred during the course of the action. That is, if an agent picks up a ball, and the simulator decides that the course of change has caused the agent's hands to slip and the ball to drop, the agent may (at the discretion of the developer of the domain) be informed of the fact that an error has occurred, rather than forcing it to look and see that the ball has been dropped.

Since the agent's interests in terms of sensory information are expressed only once per cycle (P interval), the ability of Gensim to simulate real-world sensing depends largely on the length of this interval chosen for a particular simulation run. As mentioned in Section 4.1, humans make use of extremely small intervals, giving them a great deal of control over the

information received by senses. Even though the sensory bandwidth is limited (mainly in terms of selective attention), the ability to give new anticipatory schemas quickly allow humans to maximize that bandwidth. Thus, a large P interval limits the ability of an agent to control senses.

4.4. Agent-Simulator Communication

One important aspect of perception and action has been omitted in previous Sections. For an agent to perform an action using some object, or be able to look at a given object, communication must take place: the agent must somehow describe the object(s) it is interested in to the simulator. If an agent picks up a ball, for example, it must somehow inform the simulator which ball the action has been performed with. Similarly, if the agent wants more information about "that thing over there", it must describe the object as best it can, in order for the simulator to differentiate it from other objects and thus respond with the correct information.

Communication is one aspect of the relationship between an agent and the real world that can never be completely approximated by a simulator. The reason for this is that no communication whatsoever takes place when an agent interacts with the real world (at least in the sense that communication is normally thought of). When an agent wants to pick up an object, for example, it just does so: it doesn't have to communicate this fact to the world. This is in part because the world does not exist for the benefit of an agent, the way a simulator does: any agent is simply an object in the world, like any other physical object. It is also due to the fact that the real world keeps track of itself: an agent's actions physically alter objects, rather than indirectly manipulating some virtual representation of those objects. A simulator, on the other hand, keeps a detailed representation of every object in the world, and changes in the world are manifested by alterations in these abstract entities. Because it no longer occurs naturally, change initiated by agents must be communicated to the simulator in some way. A stronger tie between the agent and the rest of the world is thus necessary.

However, in most simulators, this tie is far too strong. Many simulators do not even use separate representations of objects for agent and simulator: the identical physical chunk of knowledge that describes an object to the simulator also describes it to any agent. When compared to the real world, this is like taking each object in a physical environment, labelling it with a unique agreed-upon symbol, and referencing everything in that fashion. This "trafficking in constant symbols" [Agre and Chapman, 1989] is widespread, unrealistic, and completely unacceptable. Supporting multiple agents already eliminates the use of symbolic labels on all domain objects to provide perception to agents, since each agent must have its own perspective in any useful multi-agent domain. Even if this point is ignored, perception would still be extremely limited if agents were forced to use the same internal object

structures that the simulator manipulates. There would, for example, be no mechanism for limiting access to certain visual aspects, since the agent would automatically acquire all the knowledge of an object maintained by the simulator.

As stated above, there is no complete answer to this problem, as the use of a simulator itself introduces the need for artificial communication. Indeed, the problem of linking perception to internal representation is a severe one, that goes far beyond simulation applications [Etzioni, 1993]. The question thus becomes one of providing these communication abilities in a way that is as unobtrusive as possible to the rest of the architecture of an agent. There are two comparisons that can be made to the way in which humans specify objects in ongoing activity that aid us in this task. Suppose Henry is looking across the surface of his desk, looking for a pen (assuming that there are several on the desk). One could argue that when Henry sees a specific pen he wants, he guides his hand to pick up the pen by referring to it in an indexical-functional or deictic [Agre, 1988] manner. That is, the pen he wants to pick up is designated by reference to Henry himself or the objects Henry knows about. Thus the pen may become `THE-PEN-BY-MY-LEFT-HAND`, or the `RED-PEN-BY-THE-COFFEE-CUP`.

On the other hand, it could be argued that Henry, having looked at the desk, would designate the pen he desires using some internal symbol (the label or designation for this symbol being unimportant, `Q1V479` having as much meaning internally as `PEN-11`). In this case, Henry could guide his hand by referring to the pen as `PEN-11` (or `Q1V479`, or whatever): the internal information kept about this pen (estimates of distance from Henry's hand, for example) obtained through vision would guide Henry to the pen.

Both of these methods allow agents to designate objects in a manner distinct from their designation within the simulator. Equally importantly, neither endows the agent with any special ability to access the simulator's objective knowledge. The former method, by requiring the agent to describe an object from its own perspective, puts the onus on the agent to generate a description if it does not already represent objects in such a fashion. This may be inappropriate in time-constrained environments, since an agent may have to spend most of its time generating symbolic descriptions. Thus, if multiple agents were to be compared, those that already make use of a deictic representation would be artificially more successful, since they could spend their entire time allotment doing useful work as opposed to generating descriptions of objects for purposes of communication.

Because of this possible bias, Gensim supports both methods of description. The former type is known in Gensim as an *object descriptor*, and is a data structure that describes an object by its relationship to the agent itself or objects the agent knows about. Two examples of object descriptors appear in

Figure 4. Each object descriptor consists of a header identifying the structure, and a conjunction of clauses. Each clause states some condition that a domain object must satisfy to match the description. These may be standard comparisons (e.g. in the first example, we state among other things that the colour of the desired object is red), or may be arbitrarily complex user-defined conditions. When making use of such comparisons, descriptions may be recursive. The second example illustrates this: we compare the colour of the object to that of another object referred to from the agent's perspective (in this case, the ball that shares the same location as the agent itself).

The choice of attributes used to form the clauses of an object description is almost entirely up to the agent. Only one restriction is made, in the interests of computational efficiency. When processing an object description, the initial list of potential objects to which the description could refer consists of every object in the environment. In order to make this initial list considerably smaller, Gensim makes the assumption that one of the clauses will be an attribute under which domain objects are indexed. This is by default a class, and thus a clause describing the class of the desired object is required. As previously mentioned, locations may also be used as an index on objects (the lack of a class clause in the second example in Figure 4 indicates this, and also illustrates the type of symbolic location label that is typical of such an index).

```
(DESC (equal class ball)                  = "The red ball that weighs 5
      (equal color red)                      pounds sitting by the sink"
      (less-than weight 5)
      (equal location by-sink))

(DESC (equal location (1 1))              = "The object at location 1,1
      (equal color                           that is the same color as the
        (DESC (equal class ball)             ball that is where I am"
              (equal location self)))))
```

Figure 4. Two examples of object descriptors.

The use of these descriptors enforces a strict barrier between the knowledge of the agent and that of the simulator. However, as mentioned previously, the simulator and agent must have certain concepts in common in order to be able to communicate with one another. In the above examples, both must agree on the concept of "red", what a "ball" is, and what a "weight" attribute describes. This implementation of object descriptors also requires that the simulator and agents share class names for objects, as well as the same representation of locations. This violates to some degree the desired distinction between agent and simulator knowledge bases. It also involves a relatively minor sacrifice in agent autonomy [Barker et al., 1992]. However, they are absolutely required for any communication to take place: there can be no communication without some agreement on the definition of objects.

Thus, overall, we view the physical separation of agent and simulator knowledge as essential, but shared aspects of that representation as equally essential.

As mentioned previously, there are some cases where this method of referring to objects is impractical. In addition to the previously cited example, there may simply be situations where it is desirable to measure the performance of agents without including the overhead of generating descriptions of objects. This is especially useful to those who view this method of communication as artificial, for which there is indeed some argument. In these cases, Gensim provides *object references*, which allow an agent to reference its own internal symbol for an object when informing the simulator of actions or sensory requests. An object reference is a much simpler structure than an object descriptor, consisting only of the agent's symbol for the object and a marker labelling the structure as an object reference. When an agent refers to an object via an object reference, a description is created by the simulator at translation time (that is, when it comes time to update the world based on the action that contains the reference, or to fulfil a sensory request on the same basis). This is done by allowing the simulator access to the agent's knowledge base. An object descriptor is constructed by the simulator based on the agent's knowledge of the object, and is evaluated as described above. Because it requires the simulator to access to the agent's own knowledge base, object descriptions can be used only in agents that make use of the simulator's own internal frame representation system for knowledge representation. This form of communication violates the basic tenet of complete separation of agent and simulator knowledge, and also limits autonomy, but may be required in some situations. Note that this does not create the same "trafficking" problem described by Agre and Chapman [1990]: here, the simulator is examining the agent to see what symbols it uses internally to describe a given object, *not* the reverse. Both methods of communication are necessarily imperfect, but only by virtue of the fact that in the real world no communication of this sort is necessary. In that such communication is unavoidable in a simulator, we view both of these alternatives as valid approaches.

Thus far, we have only discussed the abilities provided by Gensim to aid in communication from agent to simulator. Another vital ability, however, is the ability for the simulator to effectively communicate sensory information to agents. Two of these methods, signals and errors, have already been discussed. However, in many domains, the majority of information communicated to agents by the simulator will be in the form of direct responses to sensory requests. As mentioned above, a primary motivation is the separation of agent and simulator knowledge bases. While the simulator can access agent frames in order to generate object descriptions, allowing direct agent access to simulator frames is unacceptable. Thus descriptions of the objects maintained by the simulator are provided to agents as sensory

information. This is performed in a computationally simple manner: in fulfilling the sensory request, the simulator has access to all information about a particular object. The simulator simply singles out all attributes of the object marked as VISIBLE (part of the definition of an object), and adds them as a group to the buffer containing the sensory information for the agent. Thus an agent might request information about "the object in my hand", and be told that it is a white cup filled with tea. Examples of the kind of information that is given to agents as a result of sensory requests is illustrated in Figure 5. In each of the two examples shown, all the attributes given as sensory information are attributes maintained about the objects in question that have been labelled as VISIBLE. As mentioned previously, information obtained from all sensory requests are kept in a single buffer, and it is naturally the agent's responsibility to link these descriptions to objects in its own knowledge base or insert them as new objects if the agent is maintaining a world model. Thus if an agent knows about a yellow ball at a given location, and sees such an object at a different location, it is up to the agent to decide if the object has moved or if the latter object is physically distinct from the first.

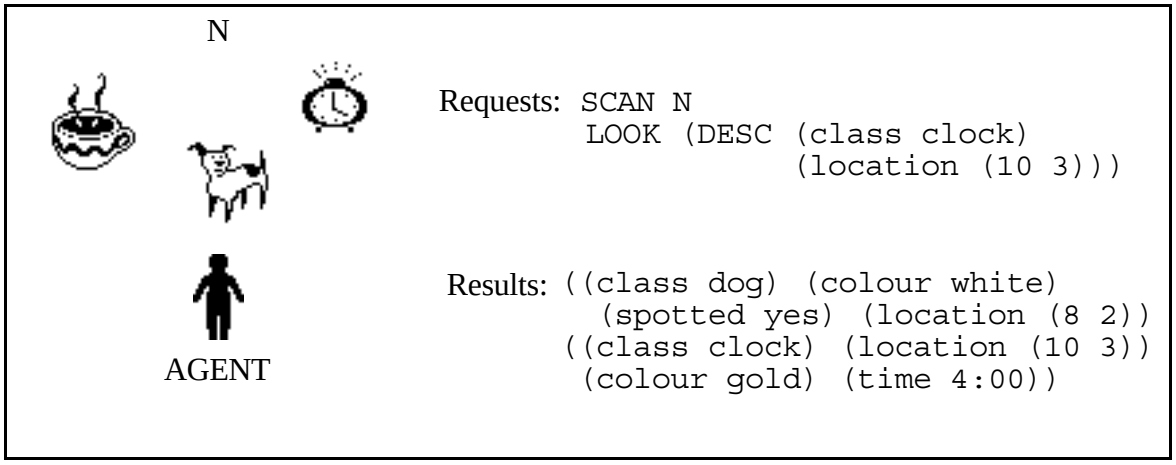

Figure 5. Example sensory requests and results.

4.5. Defining a Gensim Domain

Having described the operation of the facilities provided by Gensim, we can now examine how the various entities that make up a Gensim environment are defined. In designing Gensim, we have emphasized the development of common facilities for constructing simulated domains, rather than facilities for supporting a single specialized domain, as has been done previously. Gensim provides definition facilities (functions and macros) to allow users to easily describe aspects of the domain where names and function will vary (e.g. agents, actions, domain objects). Other aspects of a domain that have a stable

purpose (e.g. locations, domain-specific aspects of vision) are provided through user-defined functions with specific names.

There are essentially two aspects to defining any domain in Gensim. First, objects that exist in the domain must be defined, and the simulator's universal knowledge of how those objects may be manipulated (the objective physics of the domain) must be described. After an objective domain is available, each intelligent agent and its associated perspective of the simulator's objective knowledge must also be defined. As mentioned throughout this paper, simulations including intelligent agents must treat those agents both as objects within the simulator and decision-making processes in their own right.

```
(DEFAGENT BILL :PROCESSES '(P1 P2 P3) :PARTITION 'BILL-FRAMES
               :STATE-VARIABLES '(X Y Z))

(DEFUN P1 ()
;recognize and handle errors from the previous cycle
....)
(DEFUN P2 ()
;recognize objects given sensory information from the simulator
...)
(DEFUN P3 ()
;choose an action based on the current situation
...)

(DEFACTION 'THROW 'THROW-OBJECT :ATTACH-TO BILL)
(DEFUN THROW-OBJECT (some-object)
 ;send a throw message to some-object, and update the changes this
 ;action makes to bill
 (SEND 'THROW TO-BE-THROWN)
...)

(DEFEVENT 'TRAVEL-THROUGH-AIR 'TRAVEL :ATTACH-TO BALL)
(DEFUN TRAVEL (DIRECTION)
  ;Procedural aspects of an event: make the ball move
  ;in the given direction, and check for collisions, inserting new
events
  ;in the event queue as appropriate
...)
```

Figure 6. Definition of Agents, Actions, and Events.

The former category is accomplished through an explicit DEFAGENT macro. This facility defines the names of the processes an agent consists of and allows overriding of the simulation parameters that control how long each process runs. It creates a frame partition to hold the agent's knowledge (should the agent wish to make use of the built-in knowledge representation mechanisms), and defines a frame in the simulator's knowledge base to hold

information about the physical nature of the agent. It also allows the user to specify *state variables*, whose values are to be saved and restored each time an agent process is run. Together, the agent partition and this list of variables provide the execution context of the agent's processes. This facility also allows the user to use specific functions to the agent's knowledge base and reset it after each run. Figure 6 illustrates an example of this process: an agent named `BILL` is defined, consisting of three processes (`P1`,`P2`,`P3`), and a frame partition known as `BILL-FRAMES`. Functions for initializing and resetting the agent's knowledge base are names, and the values of several variables (`X`,`Y`,`Z`) will be saved and restored as part of the execution context.

The second aspect of agent definition, the internal knowledge and processing of an agent, is largely left up to the design of the user. Functions must be defined to match those described in the agent definition above, but their internal workings are left unconstrained. For example, the definition in Figure 6 is for a reactive agent used in a Gensim domain we have constructed. The first two processes recognize errors and objects, essentially integrating sensory information, and the third uses this new knowledge to select an appropriate action from a finite set of possibilities.

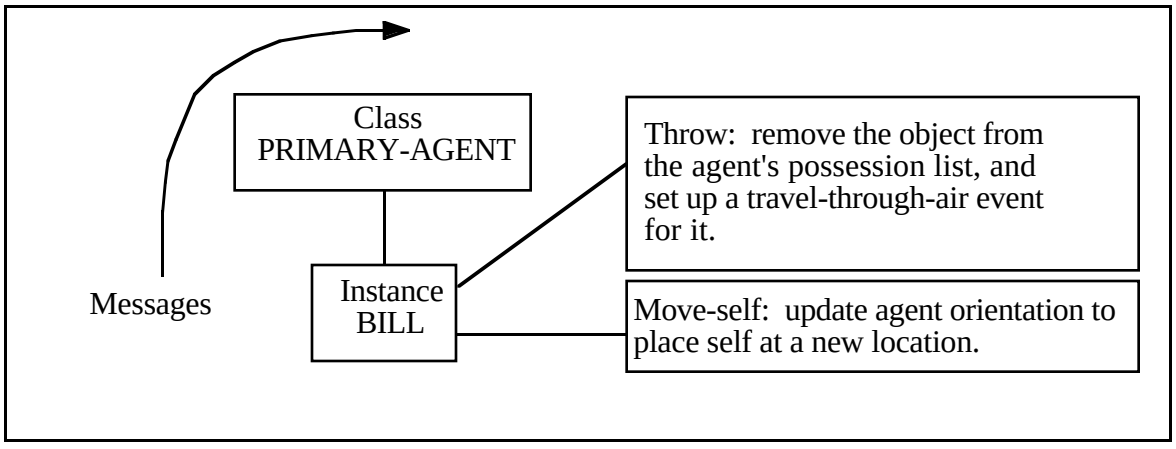

Figure 7. Actions attached to a particular agent.

As mentioned previously, an agent can make use of any internal representation for action it desires. However, the agents' internal representations of its own abilities (or those of others) is essentially a model of the simulator's own knowledge of action. A procedural definition of each action (possibly specific to particular agents) must therefore exist within the simulator, in order for it to be able to update the world accordingly as agents go about their own courses of activity. These definitions are provided using an explicit `DEFACTION` facility, an example of the use of which is also illustrated in Figure 6. The `DEFACTION` macro names an action (in this case, the action is called `THROW`), associates a procedural definition with the action (the procedure `THROW-OBJECT` in this case), and links the action to a particular

agent or group of agents known to the simulator. In the example, this action definition is particular to Bill, but it is also possible to construct an action that is applicable to several or all agents.

An abstract view of this structure is shown in Figure 7. Bill has two actions which it can perform, throwing an object and moving itself. These are done through message passing, in the same manner as the event processing described in Section 4.2. The THROW action in this case, sends a throw message to an object, and the object can then proceed as dictated by the physics. If a ball were thrown by Bill, the result would be the sequence of events previously shown in Figure 4: a string of TRAVEL-THROUGH-AIR events, ending with hitting another object or running out of energy and halting. Once again, this is the *simulator's* knowledge of the agent and what it can accomplish. Such actions must be defined within the agent's own knowledge base in any manner in order to provide the agent with a suitable view of its own abilities.

Thus, agent actions operate as follows: as agent processes are executed by the simulator, actions are performed. Once all processes for every agent have been carried out, a cycle is completed, and the environment is updated. One of the tasks to be performed during this update is to reflect change made by the agents' actions. This is done by taking all the actions performed by each agent in turn, and for each action, sending a message to the frames containing information about that particular agent. This invokes the simulator's procedural representation of that action, bringing about change to the environment. The name of the message to be handled is the same as the name of the action (e.g. a THROW message is sent to the frame defining the simulator's knowledge of Bill after Bill indicates the performance of a THROW action), and is defined by DEFACTION, as is the name of the procedure to manifest the changes that action makes on the environment. The call to DEFACTION shown in Figure 6, for example defines the name of the procedure used to handle the effects of Bill's THROW action to be THROW-OBJECT. Once again, this is the simulator's view of the agent's performance of an action: it defines only what the simulator needs to manifest change. Internally, the agent may reason about action in whatever way it desires, from a simple STRIPS formalism, to making use of the facilities for action provided by Gensim (e.g. an agent could manage its own internal event queue), to custom-designed representations for actions.

As mentioned in Section 4.2, events are handled in a manner similar to that of actions, except that the procedures describing the effects of an event are attached to the objects participating in the event (using a DEFEVENT facility, an example of which is shown in Figure 6). To use the example from section 4.2, if a ball can participate in a TRAVEL-THROUGH-AIR event defined on it, TRAVEL-THROUGH-AIR will exist as a procedural attachment to the particular ball on on some class structure above it. When TRAVEL-THROUGH-AIR is taken from the event queue, an appropriate message is sent to the object

participating in the event, activating the procedural attachment. The object-oriented approach used by Gensim to implement actions and events is modular and allows the physics of a domain to be defined and extended easily.

In addition to intelligent agents, Gensim also supports two further agent types required for defining a wide range of realistic domains. *Controlled* agents represent entities in the environment which are a source of change, but may not necessarily fit the criteria to be considered an intelligent agent. In a kitchen domain, for example, there are objects such as toasters and ovens, which are by no means intelligent, but still have effects on the world (bread to become brown, food to be cooked, etc.). These agents are similar in some ways to intelligent agents, but are much simpler in terms of simulation: they have no deliberative aspects, and therefore no computational processes to be run. They do, however, have events associated with them: a toaster, for example, would require the definition of a `toast` event, which will presumably be implemented to cause any bread in the toaster to get dark and the toaster to eventually `pop`, and an `unplug` action, which would stop the process. Controlled agents are activated (usually through the actions of primary agents), and contribute events to the event queue during the interval in which the simulator is updating the environment.

Other sources of change in any simulated environment may arise spontaneously from random elements in the world: for example, while an intelligent agent could throw a ball and knock a lamp off a table, a strong wind could also blow the same lamp off the table independently of any intelligent agent. Change of this type in a Gensim domain is supported by the definition of *random* agents. Random agents are similar to controlled agents, in that they have no deliberative aspects and induce change by triggering events. Random agents are run each cycle, and contribute random aspects as specified in their definition. Together, Random and Controlled agents support the definition much more complex and dynamic domains than would be possible otherwise. Previously, complete scripts have been used (e.g. in Phoenix) to support dynamic change [Howe, 1993]. The use of controlled and random agents, on the other hand, allows us to support ongoing events in a dynamic fashion, by having sequences of events unfold through the event queue described in Section 4.2. The event queue can also support scripted changed in the environment, simply by pre-loading events before running a simulation.

As mentioned at the beginning of this Section, part of the definition of the domain falls outside of the automated facilities described thus far. The most obvious example of this is the definition of a physical map of the domain. This is done by first setting a parameter in the simulator to record the valid directions supported by the domain (e.g. left, right, up,down,forward,back), and their inverses (if supported). Following that, the map itself must be

implemented in a function called NEXT-LOCATION, which accepts a current location and a direction and returns the next location. This forms the basis for all movement in the domain, and also defines the format of locations. To date, we have used both symbolic and grid-based locations in Gensim domains. NEXT-LOCATION is a simple mathematic function for grid-based domains, making their definition simpler.

Other aspects of domain definition are defined in much the same way. Standard functions exist for SCAN and LOOK (as described in Section 4.3), implementing the domain-specific aspects of these sensory requests. Users are expected to modify these functions accordingly to suit their domain. This sounds much more intimidating than the process actually is in practice: the functions that must be defined as part of a Gensim domain rely entirely on other aspects of the domain that the user has already described (e.g. what locations and directions are available), and require no knowledge of the internal structure of Gensim itself. Thus, it is no more complicated than making use of any other programming toolkit.

In addition to the facilities for programming complete domains, Gensim also provides a number of system and agent parameters that can be varied from run to run. These include the length of a process time-slice and various sensory options for individual agents.

4.6. Overall Gensim Algorithm

As a whole, Gensim is a complex software system. However, as is evident from previous Sections, this complexity is mainly due to the interaction of its many components. Each aspect of Gensim, from agent's senses to defining actions, is semantically explicit and relatively easy to make use of.

Now that all individual components have been examined, the overall algorithm of Gensim is easily described. This algorithm is shown in Figure 8. After initializing the system, Gensim enters a repetitive cycle of timesharing agents. Each agent process is run, during which agents perform actions and request sensory information. Before the world is updated by the simulator to reflect the changes made by these actions, any independent events are taken care of, and controlled and random agents are allowed to manifest their changes. This represents the changes that occur during the time period in which the agent was deliberating and performing its own actions. After this has been done, each agent in turn is examined, its actions extracted and the environment updated accordingly. Any errors that occur during this process may or may not be returned to the user, depending on the domain. Finally, the sensory requests of each agent are examined, and information based on these requests is prepared and given to agents in preparation for the next cycle.

```
repeat until system halted
 run each process of each agent
 get all events for this cycle from event queue
 update environment based on events
 update environment and event queue based on controlled agent actions
 update environment  and event queue based on random agent actions
 for each primary agent
  get all actions (translate object descriptions and references)
  update environment based on actions
  return any errors to agent
 end for
 for each primary agent
  get all sensory requests (translate descriptions and references)
  create descriptions based on sensory requests
  return information to agent
 end for
end repeat
```

Figure 8. Overall Gensim Algorithm.

5. Discussion

In this paper, we have described the main features and general operation of Gensim, a generic software testbed for intelligent agents. Gensim meets all the criteria specified in Section 2: it is modular, allowing the user to easily substitute agents or modify aspects of a domain, supports multiple agents and multiple processes, and the object-level interface between the agent and simulator is straightforward. The user can control many aspects of the simulation through system parameters, and can design a domain to keep track of any aspect not already built in to the simulator. Perhaps most importantly, however, the relationship between agents and the simulated world provided by Gensim is clear and explicit, unlike previous simulators: details of agent timing, actions and causality, and perception have all been explicitly described. This allows potential users to gauge the fit of their domain to Gensim quickly, rather than attempting to implement the domain and only later learning of semantic details that complicate the implementation. The wide variety of features provided by Gensim, coupled with an extendible design, make the system applicable to a much wider collection of domains and agent types than any of the simulators described here.

In a recent survey and analysis of simulation in AI [Hanks et al., 1993], Cohen describes three phases of research in which simulation plays a role. In an *exploratory* phase, a simulation testbed provides an environment in which agents can behave in interesting ways. In a *confirmatory* phase, the characterizations of behaviours of agents can be tightened through controlled

experimentation employing a testbed, performing experiments designed to test specific hypotheses. In the third phase, *generalization*, the researcher attempts to replicate results, demonstrating the findings of confirmatory research in less confined settings. Cohen notes that complex simulators and agents are to be preferred for the first two phases, in that they can support more interesting forms of behaviour. However, of all the systems examined in this paper (as well as others described by [Hanks et al., 1993]), Gensim is the only system that can adequately support the generalization phase. There is much argument as to how generalization can be performed (e.g., how some finding discovered through a given agent inhabiting the Tileworld can be shown to be a general feature). Hanks, for example, argues that it is difficult for any result produced in simulation to be generalized [Hanks et al., 1993]. However, generalization of results produced by a testbed can be performed in the same obvious way that results produced in the real world are generalized: the same behaviour can be shown to be true in domains that differ widely across many features. In order to perform such generalization, simulation technology *must* support the development of widely-differing, modular domains, that can share a common interface between agents. Systems such as Gensim are a large step in this direction.

In addition to supporting a wide variety of domains and agents, Gensim also provides support for representing varying degrees of autonomy in agent architectures. The facilities provided by Gensim allow variations in autonomy along three lines: knowledge representation, concept sharing, and communication. Gensim's knowledge definition facilities allow agents to directly refer to and reason about the exact objects maintained by the simulator for objective purposes (severely limiting autonomy); allow the agents to share their own representations of domain objects between one another (moderate autonomy); and also support complete independent world models for each agent (high autonomy). Domains can also be organized so that agents need only have a few concepts in common, such as a common reference for the location of objects (high autonomy), or must have common definitions of a large number of concepts (limiting autonomy). Finally, several communication methods for identifying objects referred to in agents' actions and sensory requests are available. Agents can simply specify the specific symbol or name of an object used by the simulator when referring to an object in an action or sensory request (low autonomy); can refer to its own name or symbol for the object, (moderate autonomy); or can specify objects in an indexical-functional or deictic [Agre, 1988] fashion, allowing an object to be described by its relationship to the agent itself or to other objects the agent knows about (high autonomy).

The price of all this flexibility is requiring the user to define all agents, operations, and interactions supported by the domain. In many cases, this may itself be a complex effort. As we have learned by experience, defining all the interactions in a domain may take a great deal of time, simply because all

the subtleties in the interaction of intelligent agents with a complex environment may not be immediately recognizable. Intricacies will often arise in a partial implementation of a domain, necessitating modifications from slight changes in the behaviour of objects to large-scale domain redevelopment. This is yet another reason to use a generic simulator such as Gensim over previous systems: a domain that may at first seem tailor-made to one of the other simulation tools described in this paper may only be found to be unsuitable after a partial implementation.

Gensim is implemented in Macintosh Common Lisp, and currently occupies approximately 150k of source code, including its knowledge representation system. The system is currently being used in two research projects at the University of Manitoba. One involves the implementation of improvising agents: intelligent agents that flexibly apply both plan knowledge and background knowledge of their task and environment to guide them through improvised activity. Gensim has been used in this application to illustrate improvised behaviour in several complex domains that involve both predictable interaction and dynamic external events, as well as both cognitive and physical activity [Anderson, 1995]. The other involves the implementation of a constraint-directed multi-agent coordination system [Evans et al., 1992]. We are also working to make Gensim more useful as a simulation tool for other other fields. For example, we are exploring the needs necessary for the system to be used in a natural resource management context, providing accurate simulation of domains that include intelligent agents, rather than simply providing a testbed for the agents themselves [Anderson and Evans, 1994].

To this point in time, we have concentrated largely on the simulation mechanisms and the interface between the environment and the intelligent agents the system supports, due to our desire to put Gensim to use in our own research as quickly as possible. We have delayed including many features that will be absolutely necessary for more widespread use of the system, such as a graphical interface. Currently, Gensim produces textual output informing the user of the domain changes that occur on each cycle, the actions taken by each agent, and the perceptual information given to each agent. We also intend to develop user interface features to simplify the actual programming of an environment, as well as methods of easily integrating large amounts of spatial data.

This research has significance to the field of artificial intelligence, in that improved environments for examining intelligent agents will facilitate more direct comparison of intelligent agent designs. While many of the contributions of this work are embodied in the software system itself, the overall design of Gensim makes many contributions on its own. We have examined common-sense, heuristically adequate implementations of action and perception for example, and placed an emphasis on strict separation of

agent and simulator knowledge, and stressed explicitly-defined interaction between the agents and the simulated world. Gensim is also significant to simulation and autonomy, in that it provides direct low-level support for agents of varying autonomy in multi-agent systems. Finally, this research is significant to the other areas where simulation plays an important role in decision making. In these areas, modelling of environments where the behaviour of intelligent agents is a significant factor in environmental change can be greatly improved by directly and accurately modelling the behaviour of those agents through the use of a tool such as Gensim. This will in turn further benefit AI, through improved simulation facilities and the adoption of more rigourous standards of modelling that are the norm in these fields.

Acknowledgements

The authors would like to thank David Scuse for many thoughtful discussions and much helpful advice concerning the many aspects of agent-simulator communication described in this paper. This work was supported in part through a University of Manitoba Fellowship.

References

[Agre, 1988] Agre, Philip E., *The Dynamic Structure of Everyday Life*, Ph.D. Dissertation, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, 1988. 282 pp.

[Agre and Chapman, 1989] Agre, Philip E. and David Chapman, *What are Plans For?*, Technical Report 1050, MIT AI Lab, 1989. 29 pp.

[Agre and Horswill, 1992] Agre, Philip E., and Ian D. Horswill, "Cultural Support for Improvisation", in *Proceedings of the Tenth National Conference on Artificial Intelligence*, San Jose, CA, July, 1992, pp. 363-368.

[Anderson, 1995] Anderson, John E., *Constraint-Directed Improvisation for Everyday Activities*, Ph.D. dissertation, Department of Computer Science, University of Manitoba, March, 1995. 384 pp.

[Anderson and Evans, 1994] Anderson, John E., and Mark Evans, "Intelligent Agent Modelling for Natural Resource Management", *International Journal of Mathematical and Computer Modelling* 19:9, September, 1994.

[Anderson and Evans, 1993] Anderson, John E., and Mark Evans, "Supporting Flexible Autonomy in a Simulation Environment for Intelligent Agent Designs", *Proceedings of the Fourth Annual Conference on AI, Simulation, and Planning in High Autonomy Systems*, Tucson, AZ, September, 1993, pp. 60-66.

[Barker et al., 1992] Barker, Ken, Mark Evans, and John Anderson, "Quantification of Autonomy in Multi-Agent Systems", in *Proceedings of the AAAI Cooperation Among Heterogeneous Intelligent Systems Workshop*, San Jose, CA, July, 1992.

[Chapman, 1990] Chapman, David, *Vision, Instruction, and Action*, Ph.D. Dissertation, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, 1990, 244 pp.

[Cohen et al., 1989] Cohen, Paul R., Michael L. Greenberg, David M. Hart, and Adele E. Howe, *Trial by Fire: Understanding the Design Requirements in Complex Environments*, COINS Technical Report 89-61, Department of Computer and Information Science, University of Massachusetts at Amherst, 1989.

[Dreyfus, 1981] Dreyfus, Hubert L., "From Micro-Worlds to Knowledge Representation: AI at an Impasse", in Haugeland, John (Ed.), *Mind Design* (Cambridge, MA: MIT Press), 1981, pp. 161-204.

[Durfee and Montgomery, 1989] Durfee, Edmund H., and Thomas A. Montgomery, "MICE: A Flexible Testbed for Intelligent Coordination Experiments", in *Proceedings of the Ninth Distributed Artificial Intelligence Workshop*, Eastsound, WA., September, 1989, pp. 25-40.

[Engelson and Bertani, 1992] Engelson, Sean P., and Niklas Bertani, *Ars Magna: The Abstract Robot Simulator Manual*, Department of Computer Science, Yale University, October 1992. 73 pp.

[Etzioni, 1993] Etzioni, Oren, "Intelligence without Robots: A Reply to Brooks", *AI Magazine* 14(4), Winter, 1993, pp. 7-13.

[Etzioni and Segal, 1992] Etzioni, Oren, and Richard Segal, "Softbots as Testbeds for Machine Learning", *Proceedings of the Machine Learning Workshop at AI/GI/VI '92*, University of British Columbia, May, 1992, pp. v1-v8.

[Evans et al., 1992] Evans, Mark, John Anderson, and Geoff Crysdale, "Achieving Flexible Autonomy in Multi-Agent Systems using Constraints", *Applied Artificial Intelligence* 6 (1992), pp. 103-126.

[Friedland, 1977] Friedland, Edward I., "Values and Environmental Modelling", in Hall, Charles, and John Day (Eds.), *Ecosystem Modelling in Theory and Practice* (New York: John Wiley and Sons), 1977, pp. 115-132.

[Goldstein, 1984] Goldstein, E. Bruce, *Sensation and Perception*, (Belmont, CA: Wadsworth Publishing Co.), 1984. 481 pp.

[Graham, 1965] Graham, Clarence H., "Perception of Movement", in Graham, Clarence H. (Ed.), *Vision and Visual Perception* (New York: Wiley), 1965, pp. 575-588.

[Hammond et al., 1990] Hammond, Kristian J., Timothy Converse, and Charles Martin, "Integrating Planning and Acting in a Case-Based Framework", *Proceedings of the Eighth National Conference on Artificial Intelligence*, Boston, 1990, pp. 292-297.

[Hanks et al., 1993] Hanks, Steve, Martha E. Pollack, and Paul R. Cohen, "Benchmarks, Test Beds, Controlled Experimentation, and the Design of Agent Architectures", *AI Magazine* 14(4), Winter, 1993, pp. 17-42.

[Howe, 1993] Howe, Adele, "Evaluating Planning through Simulation: An Example Using Phoenix", *Proceedings of the workshop on the Foundations of Automatic Planning: The Classical Approach and Beyond*, AAAI Technical Report SS-93-03, pp. 53-57.

[Howe and Cohen, 1990] Howe, Adele, and Paul R. Cohen, *Responding to Environmental Change*, Technical Report 90-23, Department of Computer and Information Science, University of Massachusetts at Amherst, 1990. 12 pp.

[Kinney and Georgeff, 1991] Kinney, David N., and Michael P. Georgeff, "Commitment and Effectiveness of Situated Agents", in Proceedings of the *Twelfth International Joint Conference on Artificial Intelligence*, Sydney, Australia, 1991, pp. 82-88.

[Kuipers, 1986] Kuipers, B., "Qualitative Simulation", *Artificial Intelligence* 29, 1986, pp. 289-338.

[Lansky, 1986] Lansky, Amy, "A Representation of Parallel Activity", in *Proceedings of the 1986 Workshop on Reasoning about Actions and Plans*, Timberline, OR, 1986, pp. 123-159.

[Maes and Brooks, 1990] Maes, Pattie, and Rodney A. Brooks, "Learning to Coordinate Behaviours", in *Proceedings of the Eighth National Conference on Artificial Intelligence*, Boston, MA, 1990, pp. 796-802.

[McDermott, 1981] McDermott, Drew V., "Artificial Intelligence Meets Natural Stupidity", in Haugeland, John (Ed.), *Mind Design* (Cambridge, MA: MIT Press), 1981, pp. 143-160.

[McDermott, 1982] McDermott, Drew V., "A Temporal Logic for Reasoning about Processes and Plans", *Cognitive Science* 6, 1982, pp. 101-155.

[Minsky, 1980] Minsky, Marvin, "A Framework for Representing Knowledge", in Haugeland, John, Ed., *Mind Design* (Cambridge, MA: MIT Press), 1981, pp. 95-128.

[Montgomery et al., 1992] Montgomery, Thomas A., Jaeho Lee, David J. Musliner, Edmund H. Durfee, Daniel Damouth, and Young-pa So, *MICE Users Guide*, Technical Report, Dept of Electrical Engineering and Computer Science, University of Michigan, January, 1992. 20 pp.

[Neisser, 1976] Neisser, Ulrich, *Cognition and Reality* (San Francisco: W. H. Freeman & Co.), 1976. 230 pp.

[Pollack and Ringuette, 1990] Pollack, Martha E., and Marc Ringuette, "Introducing the Tileworld: Experimentally evaluating Agent Architectures", in *Proceedings of the Eighth National Conference on Artificial Intelligence*, Boston, MA, 1990, pp. 183-189.