# A FRAMEWORK FOR THE DISTRIBUTED SIMULATION OF AGENT-BASED SYSTEMS

Georgios Theodoropoulos and Brian Logan
School of Computer Science, University of Birmingham
Birmingham, B15 2TT UK
g.k.theodoropoulos@cs.bham.ac.uk

## ABSTRACT

Agent-based systems are increasingly being applied in a wide range of areas including telecommunications, business process modelling, computer games, control of mobile robots and military simulations. Such systems are typically extremely complex and it is often useful to be able to simulate an agent-based system to learn more about its behaviour or investigate the implications of alternative architectures. However conventional distributed simulation techniques cannot easily be applied to the problem of modelling the interaction of a system of autonomous components. In this paper, we propose a general framework for the distributed simulation of agent-based systems and sketch an algorithm for the efficient distribution of agents and their environment across multiple processors.

## INTRODUCTION

There has been considerable recent interest in *agent-based systems*, systems based on autonomous software and/or hardware components (agents) which cooperate within an environment to perform some task. Agent-based systems offer advantages when independently developed components must inter-operate in a heterogeneous environment, e.g., the INTERNET, and agent-based systems are increasingly being applied in a wide range of areas including telecommunications, business process modelling, computer games, control of mobile robots and military simulations (Bradshaw 1997, Jennings & Wooldridge 1998).

Such systems are often extremely complex and it can be difficult to formally verify their properties. As a result, the development of agent-based systems is typically rather ad-hoc, and though many theories, architectures and implementation languages have been proposed, their relative merits are unclear. For this reason, and in those cases where the agent-based system is itself a simulation of another system, it is often useful to be able to *simulate* an agent-based system to learn more about its behaviour or investigate the implications of alternative architectures.

However, the computational requirements of simulations of agent-based systems far exceed the capabilities of conventional sequential von Neumann computer systems. Each agent is typically a complex system in its own right (e.g., with sensing, planning, inference etc. capabilities), requiring considerable computational resources, and many agents may be required to investigate the behaviour of the system as a whole or even the behaviour of a single agent (Sloman 1998). One solution to this problem is to attempt to exploit the high degree of parallelism inherent in agent-based systems. However work to date has tended to employ various ad-hoc approaches to parallel simulation, e.g., distributing the agents over a network of processors interacting via some communication protocol, and has often yielded relatively poor performance (Baxter & Hepplewhite 1996, Vincent, Horling, Wagner & Lesser 1998).

These limitations have led several researchers to explore the application of distributed simulation techniques to agent-based systems. For example, the JAMES system uses the parallel DEVS framework to model mobile, deliberative agents (Uhrmacher, Tyschler & Tyschler 1998). In this paper, we identify the efficient distribution of the agents' environment as a key problem in the parallel distributed simulation of agent-based systems and present an alternative approach to the decomposition of the environment which facilitates load balancing. We describe some of the problems in applying conventional distributed simulation techniques to agent-based systems and outline how these techniques can be

adapted to form the basis of a simulation framework which allows the reuse of existing agent and simulation code. In the next section, we briefly describe the logical process paradigm which forms the starting point for our work. In subsequent sections we describe the special problems of modelling the agents and their environment within this approach, and identify the efficient distribution of a large shared state as a key problem in modelling agent-based systems. We then briefly describe an approach to partitioning the shared state based on 'spheres of influence' and sketch load balancing algorithms and a simulation architecture based on these ideas. In the conclusion, we briefly describe how this framework might be implemented, and list some open problems.

## THE LOGICAL PROCESS PARADIGM

Various approaches for exploiting parallelism at different levels in simulation problems have been developed (Ferscha & Tripathi 1994, Fujimoto 1990). Decentralised, event-driven distributed simulation is particularly suitable for modelling systems with inherent asynchronous parallelism, such as agent-based systems. This approach seeks to divide the simulation model into a network of concurrent *Logical Processes* (*LPs*), each maintaining and processing a disjoint portion of the state space of the system. State changes are modelled as timestamped events in the simulation. From an LP's point of view, two types of events are distinguished; namely internal events which have a causal impact only to the state variables of the LP, and external events which may also have an impact on the states of other LPs. External events are typically modelled as timestamped messages exchanged between the LPs involved.

LPs operate asynchronously, advancing at completely different rates. Each LP maintains a local clock with the current value of the simulated time, Local Virtual Time (LVT); this value represents the process's local view of the global simulated time and denotes how far in the simulated time the corresponding process has progressed. An LP will repeatedly accept and process messages arriving on its input links advancing its LVT and possibly generating, as a result, a number of messages on its output links. The timestamp of an output message is the LVT of the LP when the message was sent.

A fundamental issue in decentralised, event-driven distributed simulation is to ensure that the LPs obey the *local causality constraint*, always processing messages in increasing timestamp order and hence faithfully and accurately implementing the causal dependencies and partial ordering of events dictated by the causality principle in the modelled agent-based system (Lamport 1978, Misra 1986). Two major approaches have been developed to ensure that the local causality constraint is not violated, namely *conservative* (Chandy & Misra 1981) and *optimistic* (Jefferson & Sowizral 1985). The former strictly avoids causality errors but can introduce deadlock problems. The latter allows the processes to optimistically advance in simulated time, detecting and recovering from causality errors by means of a *rollback* mechanism which forces processes to undo past operations. For the rollback of the simulation to be feasible, each process must hold information regarding its recent history (e.g., previous state vectors, processed input events, and previously sent output messages) up to last 'correct time' (referred to as the *Global Virtual Time* (GVT)), which generally is the smallest local clock value amongst all independent processes and which is periodically computed and distributed to all the logical processes.

In a conventional decentralised event-driven distributed simulation, the aim is to subdivide the system into a network of concurrent logical processes each of which models some object(s) or process(es) in the simulated system. Each LP typically shares a small number of event types (state variables) with other LPs and interacts with them in a small number of well-defined ways.[1] The topology of the simulation is determined by the topology of the simulated system and its decomposition into LPs, and is largely static.

However, the inherent characteristics of agent-based systems complicates the construction of distributed simulation models, and introduces several new and challenging problems. In the remainder of this paper, we briefly outline some of the problems of applying the logical process paradigm to the simulation of agent-based systems and sketch a distributed simulation framework which aims to address these problems.

[1] If the interactions are stochastic, we assume that the type of interaction and its possible outcomes are known in advance.

## MODELLING MULTI-AGENT SYSTEMS

The term *agent* is used in many different senses in the agent-based systems literature (see, for example, (Franklin & Graesser 1997)). For the purposes of this paper, we shall define an *agent* as a self-contained, concurrently executing thread of control that encapsulates some state and communicates with its environment and possibly other agents via some sort of message passing (Wooldridge & Jennings 1995).

There are many approaches to constructing agent-based systems, and many different architectures have been proposed. In many cases, these architectures offer considerable opportunities for parallel simulation in their own right. However, for ease of exposition and to facilitate the reuse of existing code and libraries for the development of agent-based systems, in this paper we shall model each agent as a single *Agent Logical Process* (ALP), and will not consider simulation of the agent's internal operation further.

A key characteristic of an agent is the idea of *autonomy*, and the ability to generate its own goals is often taken to be a defining characteristic of an 'autonomous agent'. The autonomous generation of goals implies that the agent has inbuilt desires or preferences determined by the developer of the agent system. Typically, such desires are sensitive to the current state of both the environment and the agent; situations which give rise to a new goal when the agent is in one state may not give rise to goals when the agent is in another state, e.g., when it is attending to a higher priority goal.

Unlike a conventional (passive) LP, the actions performed by an agent are therefore not simply a function of events in its environment: in the absence of input events, an agent can still produce output events in response to autonomous processes within the agent. For example, a WWW agent may decide to check a news service once every half hour, or a delivery robot may decide to modify its planned route to include a recharging station in order to top up its batteries.

Agents are embedded in an environment. The *environment* of an agent is that part of the world or computational system 'inhabited' by the agent. The environment may contain other agents whose environments are disjoint with or only partially overlap with the environment of a given agent. For any given agent, the other agents will themselves form part of the environment of the agent. However, for ease of exposition, we shall distinguish between agents and the environment proper.

We assume that objects and processes within the agents' environment are modelled as one or more *Environment Logical Processes* (ELP). As with ALPs, there is often considerable scope for the parallel simulation of the individual environment components, but for reasons of simplicity and to facilitate the reuse of existing code we do not consider this further. However, unlike ALPs, ELPs are not autonomous, generating output events only in response to incoming events.

There are many ways in which the agents' environment can be decomposed into ELPs. The appropriate 'grain size' of the simulation will depend both on the application and on practical considerations, such as the availability of existing simulation code. While there are obvious advantages in reusing part or all of an existing simulation, this can result in an inappropriate grain size which makes it difficult to parallelise the model. For example, modelling the environment as a single logical process can create a bottleneck in the simulation which degrades its performance, since all all agents must react to and act within the environment.[2]

Different kinds of agent have differing degrees of access to different parts of the environment. For example, a WWW agent has in principle complete access to any site on the INTERNET. Conversely, a synthetic pilot agent in a military training simulation typically only has access to a small part of its environment. The degree of access is dependent on the range of the agent's sensors (read access) and the actions it can perform (write access). However, in many cases, an agent can effectively change the topology of the environment, either by changing (autonomously) the type or frequency of the events it generates, by changing its own state (autonomously), for example, by moving from one part of the environment to another, or by changing the topology of the rest of the environment, for example, if the agent moves a bomb from one part of the environment to another.

It is therefore difficult to determine an appropriate simulation topology *a priori*. As a result, a simulation of a multi-agent system typically requires a (very) large set of shared variables which could, in principle, be accessed or updated by the agents (if they were in the right position at the right time etc.). Which variables the agents can in fact access/update depends both

[2]The few existing attempts to built distributed simulations of agent based systems adopt such a centralised approach in which the agents' environment forms part of a central time-driven simulation engine (Baxter & Hepplewhite 1996, Vincent et al. 1998).

on the state of the agents (e.g., their position) and the state of the rest of the environment. However, the information required to determine the set of variables which will be affected is itself distributed. The resulting all-to-all communication of the shared state variables is extremely costly and results in the loss of many of the advantages of distributed simulation.

Therefore, what is required is an alternative approach to decompose and distribute the shared state, which minimises bottlenecks and broadcast communication and by implication, maximises performance. In the next section, we outline an idealised decomposition of the shared state into logical processes.

## SPHERES OF INFLUENCE

ALPs interact with ELPs and other ALPs via events, modelled as timestamped messages. The purpose of this interaction is to exchange information regarding the values of the shared state variables which define the agent's manifest environment and the interfaces between the ELPs.

We assume that each ALP/ELP is capable of generating and responding to at most a finite number of event types, and a specification of the possible input and output event types forms the interface both between individual logical processes and between the logical processes and their environment. Different types of events will typically have different effects on the environment, and, in general, events of a given type will affect only certain types of state variables (all other things being equal). For example, a 'move event' generated when a robot moves forward by 1 metre will only affect the current position of the robot.

Another way of expressing this is to say that different types of event have different *spheres of influence* within the environment. 'Sphere' is used here metaphorically, to indicate the region of the environment immediately affected by an instance of an event of a particular type generated by a given agent and with a given timestamp. More precisely, we define the 'sphere of influence' of an event as the set of state variables read or updated as a consequence of the event. The sphere of influence depends on the type of event (e.g., sensor events or motion events), the state of the agent which generated the event (e.g., its position in space in the case of a robot, or the machines to which it currently has a network connection in the case of a WWW agent) and the state of the environment. The sphere of influence of an event is limited to the *immediate* and *predictable* consequences of the event rather than its ultimate effects, which depend both on the current configuration of the environment and the (autonomous) actions of other agents in response to the event.[3] For example, a 'move event' has the immediate and predictable effect of changing the position of the agent which generated the event. It may also have the further effect of rendering the agent visible to other agents, e.g., if the move event brings the agent within the visual range of another agent, or moves it out from behind an obstruction.

We can use the spheres of influence of the events generated by each agent to derive an idealised decomposition of the shared state into logical processes. The number of events of each type which have been generated by each agent can be used to compute a set of 'weighted spheres of influence', i.e., the sets of state variables affected by each *event*. Intersecting the sets of state variables for each event gives a partial order over state variables for each agent and each timepoint, in which those variables which have been most frequently read or updated come first, followed by those less frequently accessed, and finally those variables which are not affected by any of the events generated by the agent. Intersecting the partial orders for each agent gives another partial order over state variables, the least elements of which are those state variables which have been affected by the largest groups of agents.

The resulting partial order defines an idealised decomposition of the shared state into logical processes, in which the least elements contain those sets of variables which have been most frequently accessed by a particular group of ALPs and/or ELPs. Any approach to the decomposition of the shared state into logical processes should, insofar as is possible, reflect this ordering. However, any implementation can only approximate this idealised decomposition, since calculating it requires information about the global environment, and obtaining this information would not be efficient in a distributed environment. Moreover, this ordering will change with time, as the state of the environment and the relative number of events of each type produced by the agents changes, and any implementation will have to trade off the cost of reorganising the tree to reflect the ideal decomposition against the increase in communication costs due to increased broadcast communication.

[3] This should not be surprising: the computation of such ultimate effects is, after all, the purpose of the simulation.

## DISTRIBUTING THE STATE

Within the proposed framework, the decomposition of the shared state into spheres of influence is achieved by means of an additional set of Logical Processes, namely *Communication Logical Processes* (*CLPs*) to enable the clustering of ALPs and ELPs and facilitate load balancing. The CLPs each maintain a subset of the shared state variables and the interaction of ALPs and ELPs is via the variables maintained by the CLPs. The CLPs in effect define the communication channels between the ALPs and ELPs. The partitioning of the shared state is performed dynamically, in response to the events generated by the ALPs and ELPs in the simulation. Thus, the number and distribution of CLPs is not fixed, but varies during the simulation.

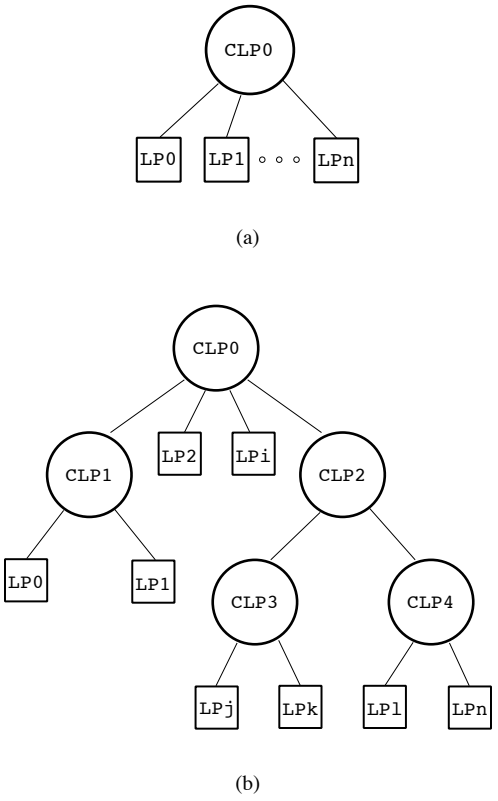


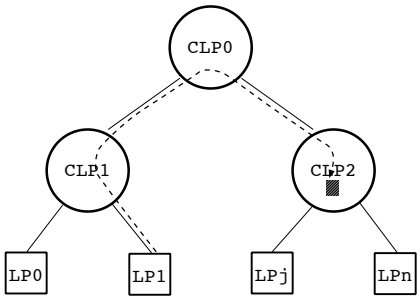

Figure 1: Generating the tree of CLPs.

We now sketch an algorithm for the decomposition of the shared state into CLPs. Initially, the whole of the shared state is handled by a single CLP, as depicted in Figure 1(a). All read and update events from all ALPs and ELPs are all directed to this single CLP, as is all inter-agent communication.

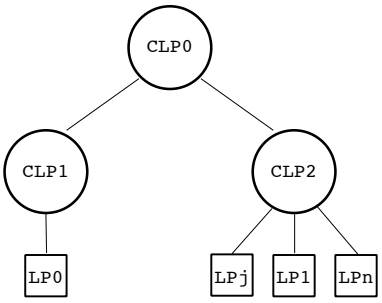
As simulation progresses, the CLP performs a dynamic analysis of the message traffic between LPs as well as the pattern and frequency of state accesses and computes an approximation of the spheres of influence. If the load increases to the point that the CLP becomes a bottleneck (e.g. when message traffic exceeds a predefined threshold), the CLP creates one or more new CLPs, to which it assigns those disjoint subsets of the state variables that form the least elements in its approximation to the partial order over the spheres of influence. Those groups of ALPs and ELPs whose events and actions have formulated the new CLP(s)' spheres of influence, communicate directly with the corresponding new CLP. The process then repeats with the newly created CLP(s) monitoring the load and generating additional CLPs as required to keep the overall simulation load on the CLPs within bounds (Figure 1(b)).

This behaviour naturally leads to a tree structure, where the ALPs/ELPs are the leaves and the CLPs the intermediate nodes of the tree. Each of the CLPs corresponds approximately to one or more spheres of influence and models the corresponding unique and disjoint section of the shared state space of the simulation. Events by the ALPs/ELPs which refer to state variables not maintained by their parent CLP will be routed through the tree to the appropriate CLP node. This can be accomplished by recording in each CLP routing information specifying which event types are relevant to its child ELPs, ALPs and CLPs and to its parent CLP.

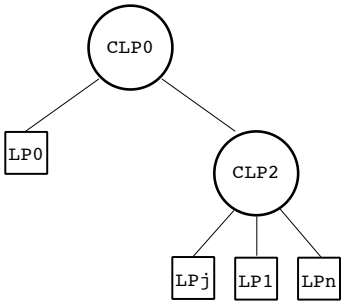
As the total number and distribution of instances of each event type generated by an ALP/ELP varies, so the partial order over the spheres of influence changes, and the structure of the tree must change accordingly to reflect the ALPs/ELPs' current behaviour and keep the communication and computational load balanced. This may be achieved in two ways, namely by changing the position of the ALP/ELP process in the tree, and by relocating state in the tree. State may be relocated either by moving subsets of the state variables from one CLP to another, or by merging CLPs upwards and then (possibly) splitting them again in a different way. For example, Figures 2(a) and 2(b) illustrate the migration of an ALP/ELP (LP1) in the tree, to bring it closer to the part of the state it most frequently accesses (denoted by the shaded area in CLP2). If this reduces the load handled by CLP1 sufficiently, it can be merged with CLP0, as depicted in Figure 2(c). Alternatively, the subset of state variables accessed by LP1 in CLP2




(a)




(b)




(c)

Figure 2: ALP/ELP migration and merging of CLPs.

could have been moved to CLP1.

## SIMULATION ISSUES

This section discusses implementation issues that will have to be addressed within the context of the proposed simulation framework.

From the two main approaches, namely conservative and optimistic, that have been developed to address the causality problem in distributed simulations, the latter emerges as the more suitable for the simulation framework proposed in this paper. In contrast with their conservative counterparts, optimistic approaches can accommodate the dynamic creation of logical processes and do not generally base its efficient operation on the prediction of future events (lookahead) (Fujimoto 1990, Ferscha & Tripathi 1994).

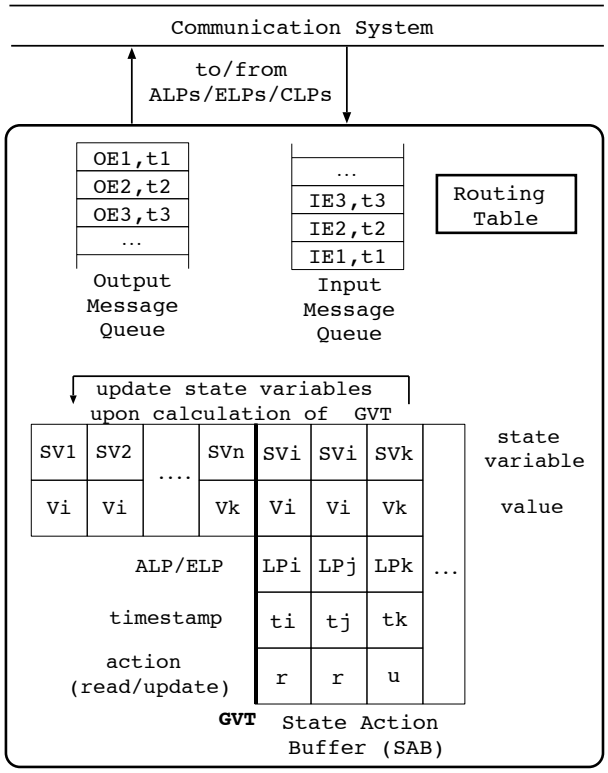



Figure 3: The Communication Logical Process.

Another important issue in the distributed implementation of the simulation, is the need of the ALPs/ELPs to access the state variables maintained by the CLPs in the simulation in order to model external actions of the agents. These actions are related to and have a causal impact on the environment of the agent and range from simple observations of and beliefs about the environment to direct interventions in the state of the environment.

The problem of implementing shared variables in distributed simulations has been addressed before in the context of other applications, e.g. (Ghosh & Fujimoto

1990); in (Mehl & Hammes 1993) more general algorithms for the implementation of shared variables are described.

Within the context of the proposed framework, reading the environment state will generally involve the interested ALP sending a timestamped query message to a CLP. The CLP will respond by sending the agent a copy of the values of the requested set of state variables which were valid at the time denoted by the request timestamp. Several different approaches may be followed in the case where the query timestamp refers to the future of the CLP depending on the type of behaviour of the simulated agent; e.g. the ALP could block until CLP's LVT reaches the instance of the query or it could be allowed to proceed its operation based on some optimistic assumptions. Alternatively, the agent may register its interest in a particular set of state variables to the corresponding CLP; the CLP will then send a copy of the state variables to the agent at regular intervals[4].

Updating the state of an CLP process requires the agent process to inform the CLP process of its external actions sending a message of the type *<action, timestamp>*, the timestamp denoting the simulated time that the action was performed by the agent.

## The Communication Logical Process

The CLPs have a dual role in the model. Firstly they act as 'routing nodes' in the model, routing through the tree messages exchanged between ALPs and ELPs as well as query and action-reporting messages to the appropriate CLPs. Secondly, they maintain the shared state, acting as a communication channel between ALPs and ELPs.

A high level view of the structure of the CLP is provided in Figure 3. Each CLP maintains a *State/Action Buffer* (*SAB*) which keeps the shared state variables as well as their recent history (i.e. since the most recent calculation of the GVT) , namely, the ALP/ELP that accessed each variable and the time and type of access (read or update). Upon receiving a message reporting an action by an ALP, the CLP will check whether this action is valid, that is, whether it has a causal relationship with an action of another agent in the simulated past. If the action is valid, CLP will keep a record of the agent's action in its SAB, and forward the message to the appropriate ELP process to compute the effects of the agent's action on the environment. If this results in further changes to the shared state, the ELP will notify the CLP to update the corresponding state variables. In order to model the effects of the agent's actions on the environment, ELPs may need to read the values of other shared state variables in the CLP by issuing query messages. The CLP will also activate the rollback mechanisms for ALPs and ELPs whose simulated past is affected by the action of the agent. If the action of the agent is not valid, the agent is forced to rollback.

[4]Of course, an agent may choose not to query the environment at all and make decisions based on certain assumptions regarding the environment state (e.g. non-monotonic reasoning).

## The Agent and Environment Logical Processes

Figure 4 presents the basic architecture of an ALP, which is the standard architecture of an LP in a typical distributed simulation (Ferscha & Tripathi 1994): an optimistic simulation engine handles the internal state of the modelled agent and performs all the required synchronisation operations maintaining the local clock (LVT) and the global view of the simulated time (GVT) of the ALP.

The simulation engine acts upon and interfaces to the agent code. More precisely, the simulation engine performs three main functions: it converts messages from other LPs into the format required by the agent, e.g., perceptual data, KQML messages etc.; it manages the agent's private internal state; and it converts the agent's actions into messages for communication to other LPs. This architecture facilitates the reuse of agent code from other existing systems, since all that is required is the development of appropriate interfaces to the simulation engine.

The structure of an ELP is similar to that of an ALP with the 'agent code' replaced by the code necessary to simulate the relevant object(s) and process(es) in the environment.

## SUMMARY

In this paper we have described a framework for the distributed simulation of agent-based systems. This framework aims to overcome the deficiencies of the ad-hoc, centralised, time-driven simulation approaches typically employed to simulate such systems by utilising a generic, decentralised, event-driven simulation engine. We have introduced the notion of 'spheres of influence' as a basis for partitioning the simulation model

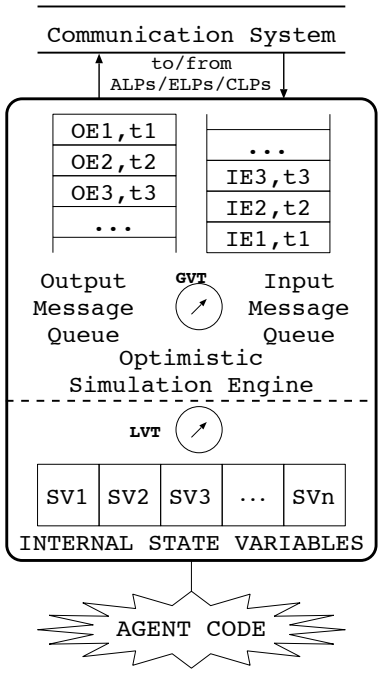



Figure 4: The Agent Logical Process.

into logical processes, and we have described an algorithm for dynamically partitioning the simulation to perform load balancing. The objective of this research is to develop a general distributed simulation environment for agent-based systems. Future work will include further investigation of the use of spheres of influence for dynamic partitioning and load balancing in the tree, the development of suitable optimistic synchronisation protocols, and ultimately the implementation of a generic simulation kernel.

# References


Baxter, J. & Hepplewhite, R. T. (1996), Broad agents for intelligent battlefield simulation, *in* 'Proceedings of the 6th Computer Generated Forces and Behavioural Representation', Institute of Simulation and Training.

Bradshaw, J., ed. (1997), *Software Agents*, AAAI Press, Menlo Park, CA.

Chandy, K. M. & Misra, J. (1981), 'Asynchronous distributed simulation via a sequence of parallel computations', *Communications of the ACM* **24**(11), 198–205.

Ferscha, A. & Tripathi, S. K. (1994), Parallel and distributed simulation of discrete event systems, Technical Report CS.TR.3336, University of Maryland.

Franklin, S. & Graesser, A. (1997), Is it an agent, or just a program?: A taxonomy for autonomous agents, *in* 'Intelligent Agents III: Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages', Vol. 1193 of *Lecture Notes in Artificial Intelligence*, Springer-Verlag, pp. 21–35.

Fujimoto, R. (1990), 'Parallel discrete event simulation', *Communications of the ACM* **33**(10), 31–53.

Ghosh, K. & Fujimoto, R. (1990), Parallel discrete event simulation using space-time memory, *in* 'Proceedings of the International Conference on Parallel Processing', Vol. III, pp. 201–208.

Jefferson, D. & Sowizral, H. (1985), Fast concurrent simulation using the time warp mechanism, *in* 'Proceedings of the SCS Distributed Simulation Conference', SCS Simulation Series, pp. 63–69.

Jennings, N. R. & Wooldridge, M. (1998), Applications of intelligent agents, *in* N. R. Jennings & M. Wooldridge, eds, 'Agent Technology: Foundations, Applications, Markets', Springer-Verlag, pp. 3–28.

Lamport, L. (1978), 'Time, clocks and the ordering of events in distributed systems', *Communications of the ACM* **21**(7), 558–565.

Mehl, H. & Hammes, S. (1993), Shared variables in distributed simulation, *in* 'Proceedings of the 7th Workshop on Parallel and Distributed Simulation (PADS93)', pp. 16–19.

Misra, J. (1986), 'Distributed discrete-event simulation', *ACM Computing Surveys* **18**(1), 39–65.

Sloman, A. (1998), What's an ai toolkit for?, *in* 'Software Tools for Developing Agents: Papers from the 1998 Workshop', number Technical Report WS–98–10, AAAI Press, pp. 1–10.

Uhrmacher, A. M., Tyschler, P. & Tyschler, D. (1998), Modeling mobile agents, *in* 'Proceedings of the International Conference on Web-based Modeling and Simulation, part of the 1998 SCS Western Multiconference on Computer Simulation', pp. 15–20.

Vincent, R., Horling, B., Wagner, T. & Lesser, V. (1998), Survivability simulator for multi-agent adaptive coordination, *in* 'Proceedings of the International Conference on Web-Based Modeling and Simulation 1998 (WMC'98)'.

Wooldridge, M. & Jennings, N. R. (1995), 'Intelligent agents: Theory and practice', *Knowledge Engineering Review* **10**(2), 115–152.