The Distributed Simulation of Multi-Agent Systems

Brian Logan, Georgios Theodoropoulos

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. In this paper, we discuss the application of distributed discrete-event simulation techniques to the simulation of multi-agent systems. We identify the efficient distribution of the agents' environment as a key problem in the simulation of agent-based systems, and present an approach to the decomposition of the environment which facilitates load balancing.**

Keywords— **agents, distributed simulation, interest management, load balancing.**

I. 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. An *agent* can be viewed 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 [1]. 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. 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 [2], [3].

While agents offer great promise, adoption of this new technology has been hampered by the limitations of current development tools and methodologies. Multi-agent systems are often extremely complex and it can be difficult to formally verify their properties [3]. As a result, design and implementation remains largely experimental, and experimental approaches are likely to remain important for the foreseeable future. In this context, simulation has a key role to play in the development of agent-based systems, allowing the agent designer to learn more about the behaviour of a system or to investigate the implications of alternative architectures, and the agent researcher to probe the relationships between agent architectures, envrionments and behaviour. The use of simulation allows a degree of control over experimental conditions and facilitates the replication of results in a way that is difficult or impossible with a prototype or fielded system, and it allows the agent designer or researcher to focus on a particular aspect of the system, deferring problems which are not central to the research or which are beyond the capabilities of current AI technology [4], [5].

Simulation has traditionally played an important role in agent research and a wide range of testbeds have been developed to support the design and analysis of agent architectures and systems, e.g., [6], [4], [7], [8]. However this work suffers from two main problems. The first problem is one of generality: no one testbed is, or can be, appropriate to all agents and environments [9]. More importantly, even if a suitable testbed can be found for a given problem, the assumptions made by the testbed can make it difficult to generalise the results obtained. Such generalisations are an essential step in establishing functional relationships between the designs of agents, their behaviour and the environment in which they are embedded. However, demonstrating that a particular result holds across a range of agent architectures and environments often requires using a number of different testbeds, with the consequent difficulties of ensuring consistency in the experimental conditions. The second problem is one of computational resources. The computational requirements of simulations of many multi-agent 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 [10]. 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 [11], [12]. These limitations have led 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 [13], [14].

What is required is a general, distributed simulation framework for multi-agent systems. Such a framework, capable of supporting a wide variety of agents and environments, would facilitate generalisation by ensuring that different implementations are subject to identical assumptions. In addition, the use of distributed simulation techniques would allow us to exploit the processing power of many machines to study larger and more complex multi-agent systems [5].[1]

This paper discusses the application of distributed discrete-event simulation techniques to the simulation of multi-agent systems. We identify the efficient distribution of the agents' en-

Brian Logan is a lecturer in the School of Computer Science and IT at the University of Nottingham, UK. E-mail:bsl@cs.nott.ac.uk.

Georgios Theodoropoulos is a lecturer in the School of Computer Science at the University of Birmingham, UK. E-mail:gkt@cs.bham.ac.uk

[1] Anderson [5] argues that the use of distributed simulation has important benefits other than that of increased computational power, including simpler and more accurate timing models and more natural implementations of perception, resulting in improved simulations.

vironment as a key problem in the simulation of agent-based systems and present an approach to the decomposition of the environment which facilitates load balancing. In the next section, we consider the problem of modelling agents and their environment and briefly describe the logical process paradigm which forms the starting point for our work. In section 3 we outline the problem of all-to-all communication of the shared state variables characteristic of multi-agent systems and in section 4 we briefly summarise existing approaches to minimising broadcast communication developed in the context of large scale, real time simulations. In section 5 we describe a new approach to partitioning the shared state based on the notion of 'spheres of influence' and in section 6 we briefly report the results of experiments attempting to characterise the spheres of influence in a simple predator and prey simulation. In sections 7, 8 & 9 we sketch load balancing algorithms, an approach to synchronisation and a simulation architecture based on these ideas. We conclude with a list of open problems and our plans for future work.

II. Modelling Multi-Agent Systems

Our aim is to simulate a wide range of agent-based systems, from a single agent in a complex environment, e.g., an agent controlling a chemical process plant or a simulated pilot agent in a military simulation, to many agents in a simple environment, e.g., an environment consisting almost entirely of other agents such as user agents, broker agents and resource agents in an INTERNET environment.

The first problem is how to model the agents and their environment. Various approaches for exploiting parallelism at different levels in simulation problems have been developed [15], [16]. 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.

We model agents and their environment as one or more Logical Processes. There are many approaches to constructing agent-based systems, and many different agent 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.

Similarly, we assume that objects and processes within the agents' environment are modelled as one or more *Environment Logical Processes* (ELP). There are many ways in which the agents' environment can be decomposed into ELPs. For example, the blocks in a simple 'blocks world' environment could each be modelled as a separate ELP, as could the physics of stacking blocks etc. Alternatively, all the blocks could form part of a single 'blocks system' ELP. 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 agents must react to and act within the environment.[2]

III. The Problem of Shared State

At any point the state of the simulation is defined by the values of the state variables maintained by the ALPs and ELPs. ALPs and ELPs interact via events, modelled as timestamped messages. The purpose of this interaction is to exchange information regarding the values of those shared state variables which define the agent's manifest environment and the interfaces between the ELPs. We say a state variable is *shared* if it is read or updated by external events generated by more than one logical process.

In a conventional decentralised event-driven distributed simulation LPs interact with each other in a small number of well defined ways. Even if the interactions are stochastic, the type of interaction and its possible outcomes are known in advance. The topology of the simulation is determined by the topology of the simulated system and its decomposition into LPs, and is largely static.

In contrast, agents are *autonomous*. 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. 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 which normally checks a news service every half hour may decide to check the service less often if nothing 'interesting' is happening, or a delivery robot may decide to modify its planned route to include a recharging station in order to top up its batteries.

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

[2]Existing attempts to build distributed simulations of agent based systems have often adopted such a centralised approach in which the agents' environment forms part of a central time-driven simulation engine [11], [12], [5].

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 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 this access set 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.

IV. Interest Management

The problem of avoiding broadcast communication has been addressed mainly in the context of real-time large scale simulations where it is termed Interest Management [17]. Interest Management techniques utilise filtering mechanisms based on *interest expressions* (IEs) to provide the processes in the simulation with only that subset of information which is relevant to them (e.g., based on their location or other application-specific attributes). The data of interest to a process is referred to as its *Domain of Interest* (DOI). Special entities in the simulation, referred to as *Interest Managers*, are responsible for filtering generated data and forwarding it to the interested processes based on their IEs [17]. The region of the multi-dimensional parameter space in which an Interest Manager is responsible for managing data transmission is referred to as its *Domain of Responsibility* (DOR).

Various Interest Management schemes have been devised, utilising different communication models and filtering schemes [17]. In most existing systems, Interest Management is realised via the use of IP multicast addressing, whereby data is sent to a selected subnet of all potential receivers. A multicast group is defined for each message type, grid cell (spatial location) or region in a multidimensional parameter space in the simulation. Typically, the definition of the multicast groups of receivers is static, based on a priori knowledge of communication patterns between the processes in the simulation [18], [19], [20], [21], [22]. For example, The High Level Architecture (HLA) utilises the *routing space* construct, a multi-dimensional coordinate system whereby simulation federates express their interest in receiving data (subscription regions) or declare their responsibility for publishing data (update regions) [23]. In existing HLA implementations, the routing space is subdivided into a predefined array of fixed size cells and each grid cell is assigned a multicast group which remains fixed throughout the simulation; a process joins those multicast groups whose associated grid cells overlap the process subscription region.

Static, grid-based Interest Management schemes have the disadvantage that they do not adapt to the dynamic changes in the communication patterns between the processes during the simulation and are therefore incapable of balancing the communication and computational load, with the result that performance is often poor. Furthermore, in order to filter out all irrelevant data, grid-based filtering requires a reduced cell size, which in turn implies an increase in the number of multicast groups, a limited resource with high management overhead. Some early systems, such as JPSD [20] and STOW-E [24] did exhibit some degree of dynamism in their filtering schemes. More recently, there have been a few attempts to define alternative dynamic schemes for Interest Management concentrating mainly on the dynamic configuration of multicast groups within the context of HLA. For example, Berrached et al. [25] examine hierarchical grid implementations and a hybrid grid/clustering scheme of update regions to dynamically reconfigure multicast groups while Morse et al. [26] report on preliminary investigations on a dynamic algorithm for dynamic multicast grouping for HLA. Saville et al. [27] describe GRIDS, a generic runtime infrastructure which utilises dynamic instantiation of Java classes in order to achieve Interest Management. The Joint MEASURE system [28], [29], [30] is implemented on top of HLA and utilises event distribution and predictive encounter controllers to efficiently manage interactions among entities. However, despite these efforts, the problem of dynamic interest management remains largely unsolved.

In the remainder of this paper we present a new approach to dynamic Interest Management, which targets the simulation of agent-based systems. Our approach is not confined to grids and rectangular regions of multidimensional parameter space and does not rely on the support provided by the TCP/IP protocols. Rather, it is based on the notion of *spheres of influence* [31], which are used to dynamically decompose and distribute the shared state so that bottlenecks and broadcast communication are minimised. In addition, our approach aims to exploit this decomposition in order to perform load balancing. Although load balancing has been studied extensively in the context of conventional distributed simulations [32], [33], [34], [35], [36], [37], it has received very little attention in relation to Interest Management, and work in this area to date is only preliminary [17], [38], [39], [40].

V. Spheres of Influence

We assume that each ALP/ELP is capable of generating and responding to a finite number of event types, and a specification of the possible input and output event types forms the interface between the ALPs and ELPs. Different types of events will typically have different effects on the shared state, 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 shared state. 'Sphere' is used here metaphorically, to indicate those parts of the shared state immediately affected by an instance of an event of a particular type 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 or environment logical process 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* 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 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.[4]

We can use the spheres of influence of the events generated by each ALP and ELP to derive an idealised decomposition of the shared state into logical processes. We define the sphere of influence of an agent or environmental logical process p_i over the time interval $[t_1, t_2]$, $s(p_i)$, as the union of the spheres of influence of the events generated by the ALP/ELP over the interval. Intersecting the spheres of influence for each event generated by the LP gives a partial order over sets of state variables for the LP over the interval $[t_1, t_2]$, in which those sets of variables which have been accessed by the largest number of events come first, followed by those less frequently accessed, and so on. The rank of a variable v_j for LP p_i over the interval $[t_1, t_2]$, $r(v_j, p_i)$ is the number of events in whose sphere of influence v_j lies.

Intersecting the spheres of influence for each LP gives a partial order over sets of state variables, the least elements of which are those sets of state variables which have been accessed by the largest groups of LPs over the interval $[t_1, t_2]$. This partial order can be seen as a measure of the difficulty of associating variables with a particular ALP or ELP: the state variables which are members of the sets which are first in the order are required by the largest number of ALPs and/or ELPs, whereas those sets of state variables which come last are required by only a single LP.

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 LPs 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.

[4]This is sometimes called the 'ramification problem' and is a special case of the frame problem which has been extensively studied in AI. We are not attempting to solve it.

VI. Characterising Spheres of Influence

We are currently conducting experiments to characterise the spheres of influence in a number of agent-based simulations. In this section, we report the preliminary results of one of these experiments from a simple predator and prey simulation.[5]

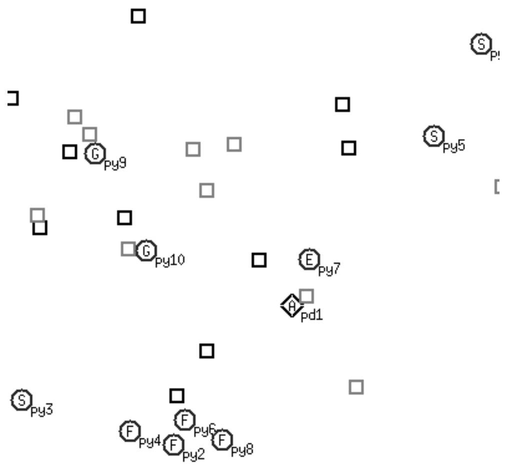


Fig. 1. A snapshot of the predator and prey simulation

The simulation was originally developed to study the effect of different herding or flocking behaviours on predation rates in a population of simulated predators and prey. The simulation consists of a number of predator and prey agents and a toroidal environment containing variable numbers of randomly distributed obstacles and food items. Each agent has a number of sensors with differing ranges, which allow the agent to sense objects and other agents within a circular region centred at its current position. For example, predators sense the position of prey and prey sense the location of predators, food and other prey. Each agent also has a collection of basic behaviours. In the case of predators, these include wandering (random motion in the environment), resting and attacking prey.[6] The behaviours of prey agents include wandering, eating food, escaping from predators, and flocking, which causes the prey agents to form groups. Figure 1 shows a snapshot of the simulation: the predator is indicated by a diamond, the prey by circles and obstacles and food by differently coloured squares. The letters inside the predator and prey indicate the agent's current behaviour, e.g., 'A' indicates attacking, 'E' escaping, 'F' flocking, 'G' grazing and so on. The simulation was developed using the SIM_AGENT toolkit [41], a sequential, centralised, time-driven simulator for multi-agent systems.

The predator and prey simulation is typical of many simulations in the multi-agent systems literature and is a useful test case for our approach. The simulation has a large shared state (representing the environment of the agents), and at any given time, each agent accesses only a subset of the state. This subset changes over time, as the agents move to escape predators or to find food. The agent's actions change the environment and

[5]We are grateful to Nick Hawes for providing the code for the simulation.

[6]If a predator 'catches' a prey agent, the prey agent is removed from the simulation and the predator gets a food reward.

hence the behaviour of other agents: when a prey agent eats some food, the food becomes unavailable to other prey agents which may 'starve' as a consequence. In addition, there is a reasonable probability that at least some of the state variables accessed by one agent will also be accessed by other agents, both because predators tend to follow prey, and because of the flocking of prey.

In the results presented below, we have made a number of simplifying assumptions. The environment is limited to grid 400 units by 400 units in size, and we assume that the state contains only location information, giving a shared state of 160,000 variables. In addition, we consider only the largest sphere of influence for each agent: the state variables representing a circular region of the environment of radius 100 units in the case of a predator, and a smaller region of radius 50 units in the case of the prey. These sets of state variables correspond to the spheres of influence of the predator's and prey's vision events respectively, which return the location and motion of all agents within the agent's respective sensor ranges.[7] A predator agent accesses approximately 19.6% of the state variables at each timestep (31,417 variables). Of these, between 96.8% and 100% will be accessed by the predator at the next timestep, due to limitations on the speed of the predator. In contrast, a prey agent accesses only 4.9% of the state variables at any timestep, of which between 94.9% and 100% will be accessed at the next timestep.

TABLE I

PROBABILITY THAT A RANDOMLY SELECTED STATE VARIABLE WILL BE ACCESSED BY n AGENTS.

Prey	0	1	2	3	4	5
5	0.675	0.257	0.058	0.008	0.001	0
10	0.567	0.282	0.111	0.031	0.007	0.001

Table I shows the probability that a randomly selected state variable will be accessed by exactly 0, 1, 2 etc. agents at any given timestep for two populations: 1 predator and 5 prey, and 1 predator and 10 prey. Initial placement of the agents, food and obstacles was random, each simulation was run for 100 timesteps and the results have been averaged over 50 runs. As expected, the increased number of agents results in an increase in the probability that any given state variable will be accessed by more than one agent from 0.325 to 0.433. However, the probability of a randomly selected variable being accessed by three or more agents is still relatively small at less than 4%.

From the data in Table I, it is apparent that the conditional probability of a state variable being accessed by at least two agents given that it is accessed by at least one is 0.206 in the case of 5 prey and 0.346 in the case of 10 prey.

[7]In the version of the simulation described above, we have used variables representing discrete spatial locations for ease of exposition, however this is not central to our argument. For example, we could have used a single state variable to record the (real-valued) position of each agent, obstacle and food item in the environment. This requires fewer state variables when the number of agents is small (and is the default approach adopted by several agent simulation systems, for example, Gensim [42] and SIM_AGENT [41]), but means that every state variable must be accessed in order to determine which objects in the environment (agents, obstacles or food items) are visible to an agent. Unless the state variables are grouped in some way based on their values (see, e.g., [5], [30]), this means that all state variables lie within the sphere of influence of each vision event and hence of each agent.

TABLE II

PROBABILITY THAT A RANDOMLY SELECTED STATE VARIABLE ACCESSED BY ONE PREY AGENT WILL BE ACCESSED BY EXACTLY n PREY AGENTS.

Prey	1	2	3	4	5
5	0.584	0.211	0.050	0.002	0
10	0.419	0.332	0.104	0.029	0.009

Further analysis of the pattern of state accesses by prey agents is reported in Table II, which gives the probability that a randomly selected member of the set of state variables accessed by one agent is also accessed by exactly 1, 2, 3 etc. other prey agents. As can be seen, even in the case of 10 prey, the probability of a state variable being shared by three or more agents is still quite small, at less than 15%.

These data suggest that, while significant numbers of variables are shared between agents, relatively few variables are shared by more than three agents. Moreover, each agent accesses a relatively small proportion of the total state at any given time.

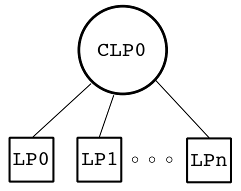
VII. DISTRIBUTING THE STATE

The decomposition of the state is achieved by means of an additional set of Logical Processes, namely *Communication Logical Processes* (*CLPs*). The CLPs act as Interest Managers. Each CLP maintains a subset of the state variables and the interaction of ALPs and ELPs is via the variables maintained by the CLPs. CLPs enable the clustering of ALPs and ELPs with overlapping spheres of influence and facilitate load balancing. 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.

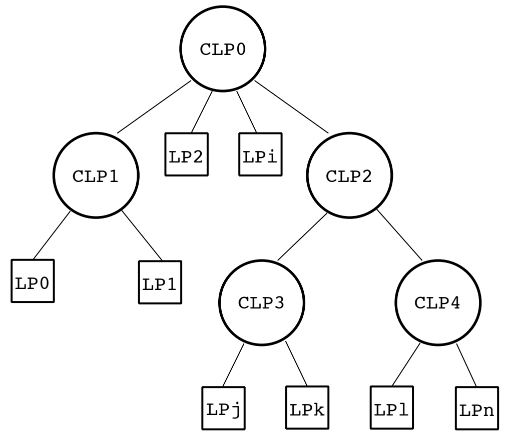
We now sketch an algorithm for the decomposition of the shared state into CLPs. Initially, the whole of the shared state is handled by a single CLP, as depicted in Figure 2(a). All read and update events from all ALPs and ELPs are all directed to this single CLP, as is all inter-agent communication.

As simulation progresses, the CLP performs a dynamic analysis of the pattern and frequency of state accesses and computes an approximation of the agents' 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 of the partial order over the spheres of influence. Those groups of ALPs and ELPs whose events and actions have formulated the new CLP(s) 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 2(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. 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



(a)



(b)

Fig. 2. Generating the tree of CLPs.

event types are relevant to its child ELPs, ALPs and CLPs and to its parent CLP.

We define the cost of accessing a variable v_j for an agent or environment logical process p_i as the rank of v_j for p_i, $r(v_j, p_i)$, times the number of CLPs which must be traversed to reach v_j during the interval $[t_1, t_2]$, $l(v_j, p_i)$, i.e., the cost of accessing variables in the local CLP is 0. Then the cost to an ALP/ELP p_i of accessing all the variables in its sphere of influence $s(p_i)$ is:

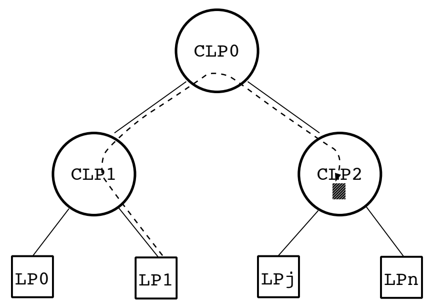
$$\sum_{v_j \in s(p_i)} l(v_j, p_i) \times r(v_j, p_i)$$

and the total access cost for all LPs $p_1, \dots, p_n$ of a particular decomposition over the interval $[t_1, t_2]$ is:

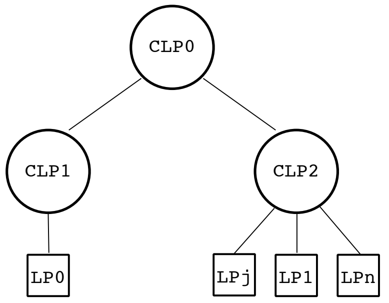
$$\sum_{i=1}^{n} \sum_{v_j \in s(p_i)} l(v_j, p_i) \times r(v_j, p_i)$$

The optimal decomposition over the interval $[t_1, t_2]$ is one which minimises the total access cost.

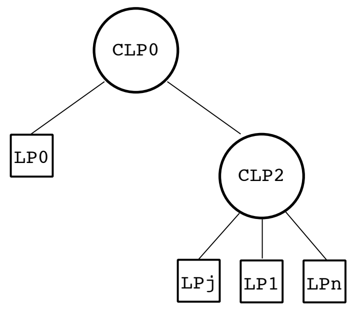
As the total number and distribution of instances of each event type generated by an ALP/ELP varies, so the partial order over the spheres of influence changes, and the structure of the tree must change accordingly to reflect the ALPs/ELPs' current behaviour and keep the communication and computational load balanced. This may be achieved in two ways, namely by changing the position of the ALP/ELP 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.



(a)



(b)



(c)

Fig. 3. ALP/ELP migration and merging of CLPs.

For example, Figures 3(a) and 3(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 3(c). Alternatively, the subset of state variables accessed by LP1 in CLP2 could have been moved to CLP1.

VIII. Synchronisation

In decentralised, event-driven distributed simulation, 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 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 problem in event-driven distributed simulation is to ensure that the LPs always process messages in increasing timestamp order, and hence faithfully and accurately implement the causal dependencies and partial ordering of events dictated by the causality principle in the modelled system. Synchronous approaches utilise global synchronisation schemes (implemented in a centralised or decentralised fashion) to force the LPs to advance together in lock step. This guarantees that the causality principle is not violated but the potential for speedup is limited. In contrast, in asynchronous simulation, LPs operate asynchronously, advancing at completely different rates, simultaneously processing events which occur at completely different simulated times. This approach has greater potential for speedup, but additional synchronisation mechanisms are required to ensure that the LPs adhere to the *local causality constraint* and process messages in increasing timestamp order [43], [44].

Two main approaches have been developed to ensure that the local causality constraint is not violated is asynchronous simulation, namely *conservative* [45] and *optimistic* [46]. Conservative approaches strictly avoid causality errors but can introduce deadlock problems. In addition, conservative techniques rely heavily on the concept of lookahead, and are thus typically suitable only for applications with good lookahead properties. However the autonomous, pro-active behaviour of agents severely restrict the ability to predict events in the model [31], [47]. Conservative protocols also typically require a static partition and configuration of the distributed model, and systems with dynamic behaviour, such as agent-based systems, are in general difficult to model [16], [48], [15]. Optimistic approaches allow the processes to advance optimistically 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). GVT is generally the smallest local clock value amongst all the LPs, and is periodically computed and distributed to all the logical processes. In contrast to conservative approaches, optimistic approaches can accommodate the dynamic creation of logical processes and do not require the prediction of future events for their efficient operation. However some authors have argued that the overhead of storing the recent history of each process is likely to outweigh the gains of employing an optimistic strategy when simulating multi-agent systems [13], [47].

Synchronisation is further complicated by the introduction of Interest Management. To date, most work on Interest Management has been carried out within the context of large-scale, real-time simulations where synchronisation is straightforward, as, at any instant, all processes (federates) are approximately at the same wall-clock time. However in logical time simulations, different LPs will typically be at different logical times, and Interest Management can introduce temporal coherency errors in the Simulation [49]. Thus far, only a limited amount of work has been done in this area, mainly for HLA and similar grid-based filtering schemes [22], [49].

The proposed framework supports both synchronous approaches and conservative and optimistic synchronisation protocols, as it seems likely that no one synchronisation technique will be appropriate in all situations. For example, the tree structure of the proposed simulation architecture facilitates the implementation of synchronous schemes, such as those employed by e.g., JAMES [13], [14]. Much recent work on deliberation for agent-based systems has focussed on 'anytime' techniques, i.e., techniques which produce solutions of monotonically increasing quality given more computational resources [50], [51]. However many other forms of deliberation are interruptible, in the sense that they can be run for a fixed period of time (or number of node expansions), and return enough state to continue the computation if a solution has not been found. This approach is commonly used in single-threaded hybrid reactive-deliberative architectures when it is necessary to place an upper bound on response time, e.g., [52]. When simulating agents which use such timesliced deliberation techniques, we can run the deliberation for a single timeslice, at the end of which either a solution has been found (or a solution of sufficient quality has been found if we are using anytime techniques) and we know how long the deliberation will take, or we know that deliberation is guaranteed to take at least the duration of the timeslice. By incrementing simulated time in increments equal to the deliberation timeslice, we can allow the rest of the simulation to proceed in parallel with the agent's deliberation.

Alternatively, we can adopt a 'moderately optimistic' strategy as proposed by Uhrmacher and Gugler [47], which splits simulation and external deliberation into two separate threads and allows simulation and deliberation to proceed concurrently, utilising simulation events as synchronisation points. The simulation is delayed to guarantee at each step that no rollback beyond the last state can occur, thus minimising the amount of state that must be stored. This has the advantage that we need make no assumptions about the interruptibility of the agent's deliberation, at the cost of the machinery and space to allow rollback to the last state.

In other cases, it may be more appropriate to model the agent's deliberation as a sequence of internal events corresponding to steps in the deliberation, allowing the rest of the simulation to progress up the timestamp of the most recent deliberative event. For example, the ACT-R agent architecture [53] associates a 'latency' with each of the rules defining the behaviour of the agent, which specifies how long the rule takes to execute. Coupled with access to ACT-R's internal state (see below), this could form the basis of an optimistic synchronisation strategy, in which simulation of the agent's deliberation proceeds optimistically in parallel with the rest of the simulation.[8] To reduce the cost of rollbacks, hybrid schemes such as Time Windows [54], [55] or Bounded Lag [56] may be utilised.

[8]ACT-R agents are not mobile, and the firing of an individual rule typically involves only limited changes to the agent's state, so the amount of state that must be stored in the event of a rollback is relatively small.

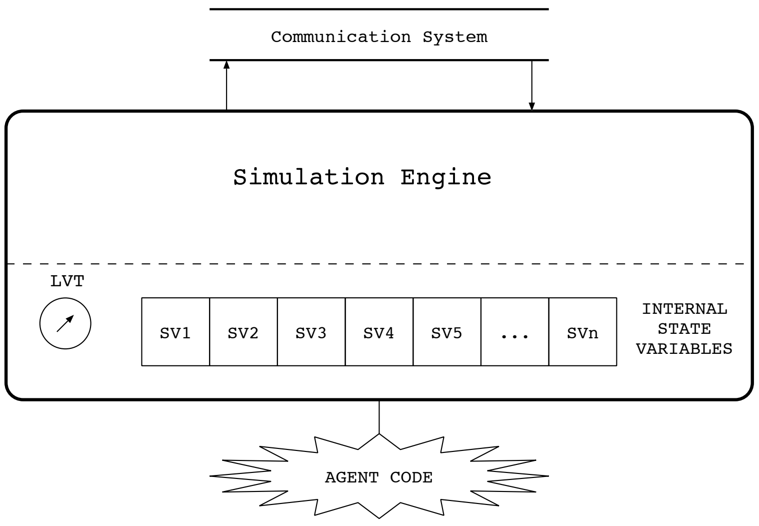


Fig. 4. The Agent Logical Process.

IX. The Simulation Architecture

The simulation consists of three types of Logical Processes, namely ALPs, ELPs, and CLPs. In this section we outline the architecture and operation of these processes assuming optimistic synchronisation.

A. The Agent and Environment Logical Processes

Figure 4 presents the basic architecture of an ALP. A simulation engine acts upon and interfaces to the agent code. The simulation engine performs four 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; it converts the agent's actions into messages for communication to other LPs; and it performs all necessary synchronisation.

The agent code could consist of, e.g., a model and its associated processor, or an existing agent implementation. The architecture facilitates the reuse of existing agent models and implementations, since all that is required is the development of appropriate interfaces to the simulation engine. Clearly, for this to be possible, the simulation engine must be able to access the agent's state and interrupt the agent and restart it on the updated state (e.g., in the event of a rollback). The ease with which this can be accomplished depends on the architecture of the agent. For example, architectures and toolkits such as SIM_AGENT [41], SOAR [57] and ACT-R [53] maintain a single, centralised representation of the agent's internal state, e.g., a working memory or goal stack, making rollback relatively straightforward. If the agent's representation is distributed, as in a neural net, or embedded in program variables, interfacing it to the simulation engine may be more difficult.

The structure of an ELP is similar to that of an ALP with the 'agent code' replaced by the models or implementation code necessary to simulate the relevant object(s) and process(es) in the environment. As with agents, a considerable amount of work has been done on developing simulations of environments which are, or could be, used in simulations of multi-agent systems.

B. The Communication Logical Process

The CLPs have a dual role in the model. Firstly they act as 'routing nodes', routing query and action-reporting messages from ALPs and ELPs through the tree 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 5. Each CLP maintains a *State/Action Buffer* (*SAB*) which keeps the shared state variables as well as their recent history (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, the 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.

C. Accessing the Shared State

The problem of sharing variables between LPs in distributed simulations has been addressed before in the context of other applications [58], [59], and an overview of the different approaches can be found in [60]. The distributed implementation of the mechanism that will allow ALPs and ELPs to access the shared state variables maintained by the CLPs will depend on the characteristics of the simulation host machine (e.g., shared or distributed memory multiprocessor, cluster of workstations etc.).

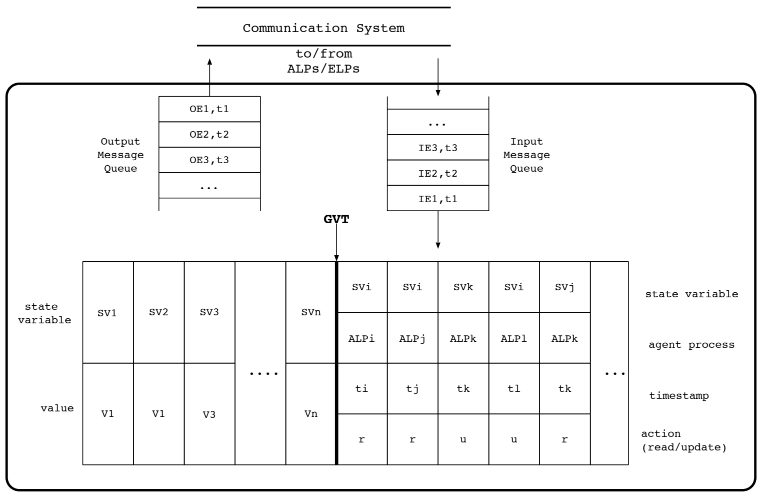


Fig. 5. The Communication Logical Process.

Within the context of the proposed framework, reading the environment state will generally involve the interested ALP issuing a timestamped query (Interest Expression) to a CLP. The CLP will respond by providing the agent with 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 the CLP's LVT reaches the timestamp of the query or it could be allowed to continue its operation based on some optimistic assumptions. Alternatively, the agent may register its interest in a particular set of state variables (its DOI) to the corresponding CLP, which will then inform the agent of any updates of these state variables[9].

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

X. Summary

In this paper we have described a framework for the distributed simulation of multi-agent systems which aims to overcome some of the deficiencies of the ad-hoc, centralised, time-driven simulation approaches typically employed for agent simulation. Our framework uses the notion of 'spheres of influence' as a basis for dynamically partitioning the shared state of the simulation model into logical processes, and we have sketched an algorithm for dynamically partitioning the simulation to perform load balancing. We have presented preliminary results of experiments to characterise the spheres of influence in a simple agent-based simulation which suggests that our approach may be feasible. However, further work is required to establish the general applicability of this approach.

In addition, a number of challenging issues have to be addressed before our approach can be realised. Techniques are required to obtain global snapshots of the distributed simulation and approximate the spheres of influence at any instant, e.g. [61], [62]. Furthermore, algorithms for redistributing the state and reorganising the tree to approximate the spheres of influence and balance the load to achieve high simulation performance must be developed. This will require the definition of appropriate performance metrics and cost functions which take into account the relevant characteristics of both the host platform (e.g., network configuration, CPU and memory architecture etc.) and the dynamics of the simulated systems (e.g., frequency of interactions and state accesses etc.). To this end, a range of alternative solutions may be envisaged, from the periodic redistribution of the whole state and construction of the tree of CLPs from scratch, to the gradual moving of LPs and state variables through different levels in a dynamically reconfigured tree. In addition, CLPs should be able to respond to various events/queries issued by the LPs regarding their environment. As the state information required to respond to these may be distributed through the tree, appropriate routing algorithms are needed to enable the CLPs to locate this information; this is clearly non-trivial. Finally, appropriate synchronisation protocols have to be developed.

In future work we hope to address these issues with the ultimate goal of implementing a generic distributed simulation kernel for multi-agent systems.

[9]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).

References

[1] Michael Wooldridge and Nick R. Jennings, "Intelligent agents: Theory and practice," *Knowledge Engineering Review*, vol. 10, no. 2, pp. 115–152, Jun 1995.

[2] Jeffrey Bradshaw, Ed., *Software Agents*, AAAI Press, Menlo Park, CA, 1997.

[3] N. R. Jennings and M. Wooldridge, "Applications of intelligent agents," in *Agent Technology: Foundations, Applications, Markets*, N. R. Jennings and M. Wooldridge, Eds., pp. 3–28. Springer-Verlag, 1998.

[4] Paul R. Cohen, Michael L. Greenberg, David M. Hart, and Adele E. Howe, "Trial by fire: Understanding the design requirements for agents in complex environments," *AI Magazine*, vol. 10, no. 3, pp. 32–48, Fall 1989.

[5] John Anderson, "A generic distributed simulation system for intelligent agent design and evaluation," in *Proceedings of the Tenth Conference on AI, Simulation and Planning, AIS-2000*, Hessam S. Sarjoughian, François E. Cellier, Michael M. Marefat, and Jerzy W. Rozenblit, Eds. March 2000, pp. 36–44, Society for Computer Simulation International.

[6] Martha E. Pollack and Marc Ringuette, “Introducing the Tileworld: Experimentally evaluating agent architectures,” in *Proceedings of the Ninth National Conference on Artificial Intelligence*, Boston, MA, 1990, AAAI, pp. 183–189.

[7] Thomas A. Montgomery and Edmund H. Durfee, “Using MICE to study intelligent dynamic coordination,” in *Proceedings of the Second International Conference on Tools for Artificial Intelligence*. 1990, pp. 438–444, IEEE.

[8] S. M. Atkin, D. L. Westbrook, P. R. Cohen, and G. D. Jorstad., “AFS and HAC: Domain general agent simulation and control.,” in *Software Tools for Developing Agents: Papers from the 1998 Workshop*, Jeremy Baxter and Brian Logan, Eds. July 1998, pp. 89–96, AAAI Press, Technical Report WS–98–10.

[9] Steve Hanks, Martha E. Pollack, and Paul R. Cohen, “Benchmarks, testbeds, contolled experimentation and the design of agent architectures,” *AI Magazine*, vol. 14, no. 4, pp. 17–42, 1993.

[10] Aaron Sloman, “What’s an AI toolkit for?,” in *Software Tools for Developing Agents: Papers from the 1998 Workshop*, Jeremy Baxter and Brian Logan, Eds. July 1998, pp. 1–10, AAAI Press, Technical Report WS–98–10.

[11] J. Baxter and R. T. Hepplewhite, “Broad agents for intelligent battlefield simulation,” in *Proceedings of the 6th Computer Generated Forces and Behavioural Representation*. Institute of Simulation and Training, 1996.

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

[13] A. M. Uhrmacher, P. Tyschler, and D. Tyschler, “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*, 1998, pp. 15–20.

[14] Bernd Schattenberg and Adelinde Uhrmacher, “Planning agents in JAMES,” *Proceedings of the IEEE*, 2000.

[15] A. Ferscha and S. K. Tripathi, “Parallel and distributed simulation of discrete event systems,” Tech. Rep. CS.TR.3336, University of Maryland, 1994.

[16] R. Fujimoto, “Parallel discrete event simulation,” *Communications of the ACM*, vol. 33, no. 10, pp. 31–53, October 1990.

[17] Katherine L. Morse, “Interest management in large scale distributed simulations,” Tech. Rep. 96-27, Department of Information and Computer Science, University of California, Irvine, 1996.

[18] Joshua Smith, Kevin Russo, and Lawrence Schuette, “Prototype multicast IP implementation in ModSAF,” in *Proceedings of the Twelfth Workshop on Standards for the Interoperability of Distributed Simulations*, 1995, pp. 175–178.

[19] Thomas W. Mastaglio and Robert Callahan, “A large-scale complex virtual environment for team training,” *IEEE Computer*, vol. 28, no. 7, pp. 49–56, July 1995.

[20] Michael Macedonia, Michael Zyda, David Pratt, and Paul Barham, “Exploiting reality with multicast groups: a network architecture for large-scale virtual environments,” in *Virtual Reality Annual International Symposium*, March 1995, pp. 2–10.

[21] James O. Calvin, Carol J. Chiang, and Daniel J. Van Hook, “Data subscription,” in *Proceedings of the Twelfth Workshop on Standards for the Interoperability of Distributed Simulations*, March 1995, pp. 807–813.

[22] Jeff S. Steinman and Frederick Weiland, “Parallel proximity detection and the distribution list algorithm,” in *Proceedings of the 1994 Workshop on Parallel and Distributed Simulation*, July 1994, pp. 3–11.

[23] Defence Modeling and Simulation Office, *High Level Architecture RTI Interface Specification, Version 1.3*, 1998.

[24] D. Van Hook, J. Calvin, M. Newton, and D. Fusco, “An approach to DIS scaleability,” in *Proceedings of the 11th Workshop on Standards for the Interoperability of Distributed Simulations*, 1994, pp. 347–356.

[25] A. Berrached, M. Beheshti, O. Sirisaengtaksin, and de Korvin A., “Alternative approaches to multicast group allocation in HLA data distribution,” in *Proceedings of the 1998 Spring Simulation Interoperability Workshop*, 1998.

[26] Katherine L. Morse, Lubomir Bic, Michael Dillencourt, and Kevin Tsai, “Multicast grouping for dynamic data distribution management,” in *Proceedings of the 31st Society for Computer Simulation Conference (SCSC ’99)*, 1999.

[27] J. Saville, “Interest management: Dynamic group multicasting using mobile java policies,” in *Proceedings of the 1997 Fall Simulation Interoperability Workshop*, 1997, number 97F-SIW-020.

[28] S. B. Hall, B. P. Zeigler, and H. Sarjoughian, “Joint MEASURE: Distributed simulation issues in a mission effectiveness analytic simulator,” in *Proceedings of the Simulation Interoperability Workshop*, Orlando, FL, 1999.

[29] S. B. Hall, “Using Joint MEASURE to study tradeoffs between network traffic reduction and fidelity of HLA compliant pursuer/evader simulations,” in *Proceedings of the Summer Simulation Conference*, Vancouver, Canada, 2000, Society for Computer Simulation.

[30] Hessam S. Sarjoughian, Bernard P. Zeigler, and Steven B. Hall, “A layered modeling and simulation architecture for agent-based system development,” *Proceedings of the IEEE*, 2000.

[31] Georgios Theodoropoulos and Brian Logan, “A framework for the distributed simulation of agent-based systems,” in *Modelling and Simulation: a tool for the next millenium, Proceedings of the 13th European Simulation Multiconference (ESM’99)*, Helena Szczerbicka, Ed. SCS, Society for Computer Simulation International, June 1999, vol. 1, pp. 58–65, Society for Computer Simulation International.

[32] C. Burdorf and J. Marti, “Load balancing strategies for Time Warp on multi-user workstations,” *The Computer Journal*, vol. 36, no. 2, pp. 168–176, 1993.

[33] D. W. Glazer and C. Tropper, “On process migration and load balancing in Time-Warp,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 3, no. 4, pp. 318–327, 1993.

[34] A. Goldberg, “Virtual time synchronisation of replicated processes,” in *Proceedings of 6th Workshop on Parallel and Distributed Simulation*. Society for Computer Simulation, 1992, pp. 107–116, Society for Computer Simulation.

[35] P. L. Reiher and D. Jefferson, “Dynamic load management in the Time-Warp operating system,” *Transactions of the Society for Computer Simulation*, vol. 7, no. 2, pp. 91–120, 1990.

[36] R. Schlagenhaft, M. Ruhwandl, C. Sporrer, and H. Bauer, “Dynamic load balancing of a multi-cluster simulation on a network of workstations,” in *Proceedings of 9th Workshop on Parallel and Distributed Simulation*. Society for Computer Simulation, 1995, pp. 175–180, Society for Computer Simulation.

[37] C. Carothers and R. Fujimoto, “Background execution of Time-Warp programs,” in *Proceedings of 10th Workshop on Parallel and Distributed Simulation*. Society for Computer Simulation, 1996, Society for Computer Simulation.

[38] P. Messina, D. Davis, S. Brunette, T. Gottshock, D. Curkendall, L. Ekroot, C. Miller, L. Plesea, L. Craymer, H. Siegel, C. Lawson, D. Fusco, and W. Owen, “Synthetic forces express: A new initiative in scalable computing for military simulation,” in *Proceedings of the 1997 Spring Simulation Interoperability Workshop*. IST, 1997.

[39] E. White and M. Myjak, “A conceptual model for simulation load balancing,” in *Proceedings of the 1998 Spring Simulation Interoperability Workshop*, 1998.

[40] M. Myjak, S. Sharp, W. Shu, J. Riehl, D. Berkley, P. Nguyen, S. Camplin, and M. Roche, “Implementing object transfer in the HLA,” Tech. Rep., 1999.

[41] A. Sloman and R. Poli, “SIM_AGENT: A toolkit for exploring agent designs,” in *Intelligent Agents II: Agent Theories Architectures and Languages (ATAL-95)*, Mike Wooldridge, Joerg Mueller, and Milind Tambe, Eds., pp. 392–407. Springer–Verlag, 1996.

[42] John Anderson and Mark Evans, “A generic simulation system for intelligent agent designs,” *Applied Artificial Intelligence*, vol. 9, no. 5, pp. 527–562, October 1995.

[43] L. Lamport, “Time, clocks and the ordering of events in distributed systems,” *Communications of the ACM*, vol. 21, no. 7, pp. 558–565, July 1978.

[44] J. Misra, “Distributed discrete-event simulation,” *ACM Computing Surveys*, vol. 18, no. 1, pp. 39–65, March 1986.

[45] K. M. Chandy and J. Misra, “Asynchronous distributed simulation via a sequence of parallel computations,” *Communications of the ACM*, vol. 24, no. 11, pp. 198–205, November 1981.

[46] D. Jefferson and H. Sowizral, “Fast concurrent simulation using the Time Warp mechanism,” in *Proceedings of the SCS Distributed Simulation Conference*, 1985, SCS Simulation Series, pp. 63–69.

[47] A. M. Uhrmacher and K. Gugler, “Distributed, parallel simulation of multiple, deliberative agents,” in *Proceedings of the nth Parallel and Distributed Simulation Conference (PADS’2000)*, Bologna, May 2000, IEEE, pp. 101–110.

[48] V. Jha and R. L. Bagrodia, “Transparent implementation of conservative algorithms in parallel simulation languages,” in *Proceedings of the 1993 Winter Simulation Conference*, December 1993, pp. 677–686.

[49] I. Tacic and R. Fujimoto, “Synchronised data distribution management in distributed simulations,” in *Proceedings of the 12th Workshop on Parallel and Distributed Simulation (PADS98)*, 1998, pp. 108–115.

[50] Tom Dean and Mark Boddy, “An analysis of time-dependent planning,” in *Proceedings of the Seventh National Conference on Artificial Intelligence (AAAI’88)*. AAAI, 1988, pp. 49–54.

[51] Brian Logan and Natasha Alechina, "*A** with bounded costs," in *Proceedings of the Fifteenth National Conference on Artificial Intelligence, AAAI-98*, Menlo Park CA & Cambridge MA, 1998, AAAI, pp. 444–449, AAAI Press/MIT Press.

[52] Brian Logan, "Route planning with ordered constraints," in *Proceedings of the 16th Workshop of the UK Planning and Scheduling Special Interest Group*. University of Durham, Dec 1997, pp. 133–144.

[53] John R. Anderson and Christian Libiere, *The Atomic Components of Thought*, Lawrence Erlbaum Associates, 1998.

[54] L. M. Sokol, D. P. Briscoe, and A. P. Wieland, "MTW: A strategy for scheduling discrete simulation events for concurrent simulation," in *Proceedings of the SCS Multiconference on Distributed Simulation*. Society for Computer Simulation, July 1988, SCS Simulation Series, pp. 34–42.

[55] R. Ayani and H. Rajaei, "Parallel simulation using conservative time windows," in *Proceedings of the 1992 Winter Simulation Conference*, December 1992, pp. 709–717.

[56] B. D. Lubachevsky, "Bounded Lag distributed discrete event simulation," in *Proceedings of the 1988 SCS Multiconference on Distributed Simulation*. Society for Computer Simulation, July 1988, SCS Simulation Series, pp. 183–191.

[57] J. E. Laird, A. Newell, and P. S. Rosenbloom, "SOAR: An architecture for general intelligence," *Artificial Intelligence*, vol. 33, pp. 1–64, 1987.

[58] K. Ghosh and R. Fujimoto, "Parallel discrete event simulation using space-time memory," in *Proceedings of the International Conference on Parallel Processing*, 1990, vol. III, pp. 201–208.

[59] D. Conklin, J. Cleary, and B. Unger, "The Sharks World: A study in distributed simulation design," in *Proceedings of the 1990 Multiconference on Distributed Simulation,*, San Diego, 1990, pp. 157–160.

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

[61] K. M. Chandy and L. Lamport, "Distributed snapshots: Determining global states of distributed systems," *ACM Transactions on Computer Systems*, vol. 3, no. 1, pp. 63–75, 1985.

[62] O. Babaoglou and K. Marzullo, "Consistent global states of distributed systems: Fundamental concepts and mechanisms," Technical Report UBLCS-93-1, Laboratory for Computer Science, University of Bologna, January 1993.

Brian Logan is a lecturer in the School of Computer Science and IT at the University of Nottingham, UK. He received a PhD in design theory from the University of Strathclyde, UK in 1986. His research interests include the specification, design and implementation of agent-based systems, including logics and ontologies for agent-based systems and software tools for building agents. Before moving to Nottingham, he was a member of the Cognition and Affect group at the University of Birmingham, where he worked on agent-related projects funded by the UK Defence Evaluation and Research Agency and the Leverhulme Trust, developing architectures for autonomous intelligent agents capable of complex decision making under constraints such as incomplete and uncertain information and time pressure.

Georgios Theodoropoulos received a Diploma degree in Computer Engineering from the University of Patras, Greece in 1989 and MSc and PhD degrees in Computer Science from the University of Manchester, U.K. in 1991 and 1995 respectively. Since February 1998 he has been a Lecturer in the School of Computer Science, University of Birmingham, U.K. teaching courses on Hardware Engineering and Computer Networks. His research interests include parallel and distributed systems, computer and network architectures and modelling and distributed simulation.