# HLA_AGENT: Distributed Simulation of Agent-Based Systems with HLA

Michael Lees[1], Brian Logan[1], Ton Oguara[2], and Georgios Theodoropoulos[2]

[1] School of Computer Science and IT
University of Nottingham, UK
{mhl,bsl}@cs.nott.ac.uk
[2] School of Computer Science
University of Birmingham, UK
{txo,gkt}@cs.bham.ac.uk

**Abstract.** In this paper we describe HLA_AGENT, a tool for the distributed simulation of agent-based systems, which integrates the SIM_AGENT agent toolkit and the High Level Architecture (HLA) simulator interoperability framework. Using a simple Tileworld scenario as an example, we show how the HLA can be used to flexibly distribute a SIM_AGENT simulation with different agents being simulated on different machines. The distribution is transparent in the sense that the existing SIM_AGENT code runs unmodified and the agents are unaware that other parts of the simulation are running remotely. We present some preliminary experimental results which illustrate the performance of HLA_AGENT on a Linux cluster running a distributed version of Tileworld and compare this with the original (non-distributed) SIM_AGENT version.

## 1 Introduction

Simulation has traditionally played an important role in agent research and a wide range of simulators and testbeds have been developed to support the design and analysis of agent architectures and systems [1,2,3,4,5,6]. However no one simulator or testbed is, or can be, appropriate to all agents and environments, and demonstrating that a particular result holds across a range of agent architectures and environments often requires using a number of different systems. Moreover, the computational requirements of simulations of many multi-agent systems far exceed the capabilities of a single computer. 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.

In this paper we present an approach to agent simulation which addresses both interoperability and scalability issues. We describe HLA_AGENT, a tool for the distributed simulation of agent-based systems, which integrates an existing agent toolkit, SIM_AGENT, and the High Level Architecture (HLA) [7]. Simulations developed using HLA_AGENT are capable of inter-operating with other HLA-compliant simulators and the objects and agents in the simulation can be flexibly distributed across multiple computers so as to make best use of available computing resources. The distribution is transparent to the

M. Bubak et al. (Eds.): ICCS 2004, LNCS 3038, pp. 881–888, 2004.

user simulation and symmetric in the sense that no additional management federates are required.

## 2 An Overview of SIM_AGENT

SIM_AGENT is an architecture-neutral toolkit originally developed to support the exploration of alternative agent architectures [4,8][1]. In SIM_AGENT, an agent consists of a collection of modules, e.g., perception, problem-solving, planning, communication etc. Groups of modules can execute either sequentially or concurrently and at different rates. Each module is implemented as a collection of condition-action rules in a high-level rule-based language. The rules match against data held in the agent's database, which holds the agent's model of its environment, its goal and plans etc. In addition, each agent also has some public data which is "visible" to other agents.

SIM_AGENT can be used both as a sequential, centralised, time-driven simulator for multi-agent systems, e.g., to simulate software agents in an Internet environment or physical agents and their environment, and as an agent implementation language, e.g., for software agents or the controller for a physical robot.

The toolkit is implemented in Pop-11, an AI programming language similar to Lisp, but with an Algol-like syntax. It defines two basic classes, `sim_object` and `sim_agent`, which can be extended (subclassed) to define the objects and agents required for a particular simulation scenario. The `sim_object` class is the foundation of all SIM_AGENT simulations: it provides slots (fields or instance variables) for the object's name, internal database, sensors, and rules together with slots which determine how many processing cycles each module will be allocated at each timestep and so on. The `sim_agent` class is a subclass of `sim_object` which provides simple message based communication primitives.

As an example, we briefly outline the design and implementation of a simple SIM_AGENT simulation, SIM_TILEWORLD. Tileworld is a well established testbed for agents [2,9]. It consists of an environment consisting of tiles, holes and obstacles, and one or more agents whose goal is to score as many points as possible by pushing tiles to fill in the holes. The environment is dynamic: tiles, holes and obstacles appear and disappear at rates controlled by the experimenter.

SIM_TILEWORLD defines three subclasses of the SIM_AGENT base class `sim_object` to represent holes, tiles and obstacles, together with two subclasses of `sim_agent` to represent the environment and the agents. The subclasses define additional slots to hold the relevant simulation attributes, e.g., the position of tiles, holes and obstacles, the types of tiles, the depth of holes, the tiles being carried by the agent etc. By convention, external data is held in slots, while internal data (such as which hole the agent intends to fill next) is held in the agent's database.

The simulation consists of two or more active objects (the environment and the agent(s)) and a variable number of passive objects (the tiles, holes and obstacles). At simulation startup, instances of the environment and agent classes are created and passed to the SIM_AGENT scheduler. At each simulation cycle, the environment agent causes tiles,

[1] See http://www.cs.bham.ac.uk/~axs/cog_affect/sim_agent.html

obstacles and holes to be created and deleted according to user-defined probabilities. The scheduler then runs the Tileworld agents which perceive the new environment and run their rules on their internal databases updated with any new sense data. Each agent chooses an external action to perform at this cycle (e.g., moving to or pushing a tile) which is queued for execution at the end of the cycle. The cycle then repeats.

## 3 Distributing a SIM_AGENT Simulation with HLA

There are two distinct ways in which SIM_AGENT might use the facilities offered by the HLA. The first, which we call the *distribution* of SIM_AGENT, involves using HLA to distribute the agents and objects comprising a SIM_AGENT simulation across a number of federates. The second, which we call *inter-operation*, involves using HLA to integrate SIM_AGENT with other simulators. In this paper we concentrate on the former, namely distributing an existing SIM_AGENT simulation using SIM_TILEWORLD as an example. Based on the SIM_TILEWORLD implementation outlined in section 2, we chose to split the simulation into $n+1$ federates, corresponding to $n$ agent federates and the environment federate respectively.

In the distributed implementation of SIM_TILEWORLD, the communication between the agent and environment federates is performed via the objects in the FOM, through the creation, deletion and updating of attributes.[2] The FOM consists of two main subclasses: *Agent* and *Object*, with the *Object* class having *Tiles*, *Holes* and *Obstacles* as subclasses. The classes and attributes in the FOM are mapped in a straightforward way onto the classes and slots used by SIM_AGENT. For example, the *depth* attribute of the *Tile* class in the FOM maps to the `depth` slot of the `sim_tile` class in SIM_AGENT.

HLA requires federates to own attribute instances before they can update their value. In HLA_AGENT we use ownership management to manage conflicts between actions proposed by agents simulated by different federates. For example, two (or more) agents may try to push the same tile at the same timestep. Once the tile has been moved by one agent, subsequent moves should become invalid, as the tile is no longer at the position at which it was initially perceived. If the agents are simulated by the same agent federate such action conflicts can be handled in the normal way, e.g., we can arrange for each action to check that its preconditions still hold before performing the update and otherwise abort the action. However, this is not feasible in a distributed setting, since external actions are queued by SIM_AGENT for execution at the end of the current cycle. We therefore extend current practice in SIM_AGENT and require that attribute updates be *mutually exclusive*. Ownership of a mutually exclusive attribute can only transferred at most once per simulation cycle, and a federate relinquishes ownership of an attribute only if it has not already been updated at the current cycle. (If multiple attributes are updated by the same agent action, we require that federates acquire ownership of the attributes in a fixed order to avoid deadlock.) For example, if two agents running on different federates try to move a given tile at the same cycle, whichever agent's action is processed first will acquire ownership of the tile and succeed, while the other will be denied ownership and fail.

[2] In this example, SIM_AGENT's inter-agent message based communication is not used. Message passing also handled by the RTI, using interactions.

SIM_AGENT is a centralised, time-driven system in which the simulation advances in timesteps or cycles. We therefore synchronise the federation at the beginning of each cycle, by making all federates both time-regulating and time-constrained. This ensures that the federates proceed in a timestep fashion, alternating between performing their external actions and perceiving changes.

## 4 Extending SIM_AGENT

In this section we briefly sketch the extensions necessary to the SIM_AGENT toolkit to allow an existing SIM_AGENT simulation to be distributed using the HLA. Together, the extensions constitute a new library which we call HLA_AGENT.

In what follows, we assume that we have an existing SIM_AGENT simulation (e.g., SIM_TILEWORLD) that we want to distribute by placing disjoint subsets of the objects and agents comprising the simulation on different federates. Each federate corresponds to a single SIM_AGENT process and is responsible both for simulating the local objects forming its own part of the global simulation, and for maintaining *proxy* objects which represent objects of interest being simulated by other federates. Each federate may be initialised with part of the total model or all federates can run the same basic simulation code and use additional information supplied by the user to determine which objects are to be simulated locally. For example, in SIM_TILEWORLD we may wish to simulate the agent(s) on one federate and the environment on another.

The overall organisation of HLA_AGENT is similar to other HLA simulators. Each SIM_AGENT federate requires two ambassadors: an RTI Ambassador which handles calls to the RTI and a Federate Ambassador that handles callbacks from the RTI. Calls to the RTI are processed asynchronously in a separate thread. However, for simplicity, we have chosen to queue callbacks from the RTI to the Federate Ambassador for processing at the end of each simulation cycle. SIM_AGENT has the ability to call external C functions. We have therefore adopted the reference implementation of the RTI written in C++ developed by DMSO, and defined C wrappers for the RTI and Federate Ambassador methods needed for the implementation. We use Pop-11's simple serialisation mechanism to handle translation of SIM_AGENT data structures to and from the byte strings required by the RTI. All RTI calls and processing of Federate Ambassador callbacks can therefore be handled from HLA_AGENT as though we have an implementation of the RTI written in Pop-11.

To distribute a simulation, the user must define the classes and attributes that constitute the Federation Object Model and, for each federate, provide a mapping between the classes and attributes in the FOM and the SIM_AGENT classes and slots to be simulated on that federate. If the user simulation is partitioned so that each federate only creates instances of those objects and agents it is responsible for simulating, then no additional user-level code is required. In the case in which all federates use the same simulation code, the user must define a procedure which is used to determine whether an object should be simulated on the current federate. The user therefore has the option of partitioning the simulation into appropriate subsets for each federate, thereby minimising the number of proxy objects created by each federate at simulation startup, or allowing all federates to create a proxy for all non-local objects in the simulation. For very large

simulations, the latter approach may entail an unacceptable performance penalty, but has the advantage that distributed and non-distributed simulations can use identical code.

## 5 Experimental Results

To evaluate the robustness and performance of HLA_AGENT we implemented a distributed version of SIM_TILEWORLD using HLA_AGENT and compared its performance with the original, non-distributed SIM_AGENT version.

The hardware platform used for our experiments is a Linux cluster, comprising 64 2.6 GHz Xeon processors each with 512KB cache (32 dual nodes) interconnected by a standard 100Mbps fast Ethernet switch. Our test environment is a Tileworld 50 units by 50 units in size with an object creation probability (for tiles, holes and obstacles) of 1.0 and an average object lifetime of 100 cycles. The Tileworld initially contains 100 tiles, 100 holes and 100 obstacles and the number of agents in the Tileworld ranges from 1 to 64. In the current SIM_TILEWORLD federation, the environment is simulated by a single federate while the agents are distributed in one or more federates over the nodes of the cluster[3]. The results obtained represent averages over 5 runs of 100 SIM_AGENT cycles.

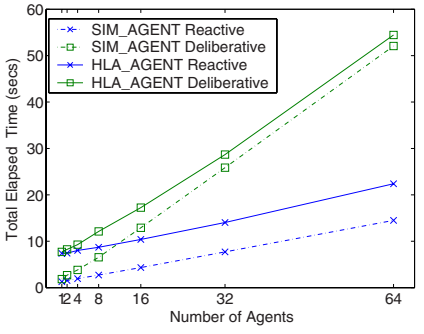
We would expect to see speedup from distribution in cases where the CPU load dominates the communication overhead entailed by distribution. We therefore investigated two scenarios: simple reactive Tileworld agents with minimal CPU requirements and deliberative Tileworld agents which use an $A^*$ based planner to plan optimal routes to tiles and holes in their environment. The planner was modified to incorporate a variable "deliberation penalty" for each plan generated. In the experiments reported below this was arbitrarily set at 10ms per plan.

For comparison, Figure 1(a) shows the total elapsed time when executing 1, 2, 4, 8, 16, 32 and 64 reactive and deliberative SIM_TILEWORLD agents and their environment on a single cluster node using SIM_AGENT and a single HLA_AGENT federate. This gives an indication of the overhead inherent in the HLA_AGENT library itself independent of any communication overhead entailed by distribution. As can be seen, the curves for SIM_AGENT and HLA_AGENT are quite similar, with the HLA overhead diminishing with increasing CPU load. For example, with 64 reactive agents the HLA introduces a significant overhead. In SIM_AGENT, the average elapsed time per cycle is 0.145 seconds compared to 0.216 seconds with HLA_AGENT, giving a total overhead for the HLA of approximately 54%. For agents which intrinsically require more CPU and/or larger numbers of agents, the overhead is proportionately smaller. With 64 deliberative agents, the average elapsed time per cycle is 0.522 seconds with SIM_AGENT and 0.524 seconds with HLA_AGENT, giving a total overhead for the HLA of just 0.4%.

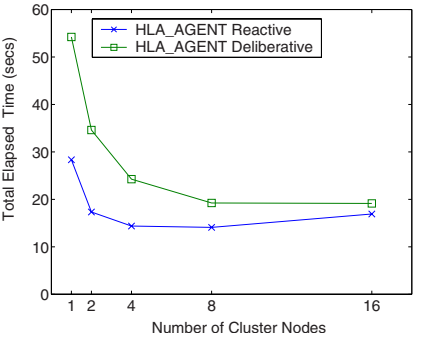
We also investigated the effect of distributing the Tileworld agents across varying numbers of federates. Figure 1(b) shows a breakdown of the total elapsed time for an agent federate when distributing 64 reactive and deliberative agents over 1, 2, 4, 8 and 16 nodes of the cluster.[4] In each case, the environment was simulated by a single environment federate running on its own cluster node. As expected, the elapsed time

[3] For our experiments, only one processor was used in each node.

[4] Unfortunately, it was not possible to obtain exclusive access to all the nodes in the cluster for our experiments.




(a) Total elapsed times for 1-64 Reactive and Deliberative agents in SIM_AGENT and HLA_AGENT (single federate).




(b) Total elapsed times for an Agent Federate (64 Reactive and Deliberative Agents distributed over 1-16 nodes).

**Fig. 1.** Elapsed Times

drops with increasing distribution, and with 4 nodes the elapsed time is comparable to the non-distributed case for the reactive agents.[5] For the more computation-bound deliberative agents a greater speedup is achieved, and with four nodes the elapsed time is approximately half that of the non-distributed case. However in both the reactive and deliberative cases, as the number of nodes (and hence the communication overhead) increases, the gain from each additional node declines. For example, with a single agent federate we would expect to see at least 128 attribute updates per cycle (since agents update their `x` and `y` positions every cycle). With 16 agent federates, the environment federate still receives 128 attribute updates per cycle, but in addition each agent federate receives at least 120 updates from the 60 agents on the other agent federates. As a result, the number of callbacks processed by the RTI in each cycle grows from 128 with 1 agent federate to 2048 with 16 agent federates.

In addition, without load balancing, the speedup that can be obtained is limited by the elapsed time for the slowest federate. An analysis of the the total cycle elapsed times for the simulation phase of HLA_AGENT (i.e., the time required to run the user simulation plus object registration and deletion, attribute ownership transfer requests and queueing attribute updates for propagation at the end of the user simulation cycle) shows that with more than 4 agent federates, the simulation phase time for the environment federate is greater than that for any single agent federate. Prior to this point, the environment federate spends part of each cycle waiting for the agent federate(s) to complete their simulation phase, and after this point agent federates spend part of each cycle waiting for the environment federate. With 8 agent federates, the elapsed time of the environment federate forms a lower bound on the elapsed time for the federation as a whole, and further speedup can only be obtained by distributing the environment across multiple federates. Without

[5] This represents a significant improvement on results reported previously [10], where 16 nodes were required to obtain speedup with 64 reactive agents. We believe the increase in performance is largely attributable to a change in the ticking strategy adopted. In the experiments reported in this paper, we used the no argument version of `tick`.

the communication overhead of distribution, we would therefore expect the total elapsed time to reach a minimum between 4 and 8 agent federates and thereafter remain constant.

Although preliminary, our experiments show that, even with relatively lightweight agents like the Tileworld agents, we can get speedup by distributing agent federates across multiple cluster nodes. However with increasing distribution the broadcast communication overhead starts to offset the reduction in simulation elapsed time, limiting the speedup which can be achieved. Together with the lower bound on elapsed time set by the environment federate, this means that for reactive agents the elapsed time with 16 nodes is actually greater than that for 8 nodes. With 64 deliberative agents, which intrinsically require greater (though still fairly modest) CPU, we continue to see small improvements in overall elapsed time up to 16 nodes.

## 6 Summary

In this paper, we presented HLA_AGENT, an HLA-compliant version of the SIM_AGENT agent toolkit. We showed how HLA_AGENT can be used to distribute an existing SIM_AGENT simulation with different agents being simulated by different federates and briefly outlined the changes necessary to the SIM_AGENT toolkit to allow integration with the HLA. The integration of SIM_AGENT and HLA is transparent in the sense that an existing SIM_AGENT user simulation runs unmodified, and symmetric in the sense that no additional management federates are required. In addition, the allocation of agents to federates can be easily configured to make best use of available computing resources.

Preliminary results from a simple Tileworld simulation show that we can obtain speedup by distributing agents and federates across multiple nodes in a cluster. While further work is required to analyse the RTI overhead and characterise the performance of HLA_AGENT with different kinds of agents and environments, it is already clear that the speedup obtained depends on the initial allocation of agents to federates. If this results in unbalanced loads, the slowest federate will constrain the overall rate of federation execution. It should be relatively straightforward to implement a simple form of code migration to support coarse grain load balancing by swapping the execution of a locally simulated object with its proxy on another, less heavily loaded, federate. We also plan to investigate the performance implications of distributing the simulation across multiple (geographically dispersed) clusters. Together, these extensions will form the first step towards a GRID-enabled HLA_AGENT.

Another area for future work is *inter-operation*, using HLA to integrate SIM_AGENT with other simulators. This would allow the investigation of different agent architectures and environments using different simulators in a straightforward way. In a related project, we are currently developing an HLA-compliant version of the RePast agent simulator [11], which will form part of a combined HLA_AGENT /RePast federation.

**Acknowledgements.** We would like to thank the members of the Centre for Scientific Computing at the University of Warwick, and in particular Matt Ismail, for facilitating access to the Task Farm cluster. This work is part of the PDES-MAS project[6] and is supported by EPSRC research grant No. GR/R45338/01.

[6] http://www.cs.bham.ac.uk/research/pdesmas

## References


1. Durfee, E.H., Montgomery, T.A.: MICE: A flexible testbed for intelligent coordination experiements. In: Proceedings of the Ninth Distributed Artificial Intelligence Workshop. (1989) 25–40
2. Pollack, M.E., Ringuette, M.: Introducing the Tileworld: Experimentally evaluating agent architectures. In: National Conference on Artificial Intelligence. (1990) 183–189
3. Atkin, S.M., Westbrook, D.L., Cohen, P.R., Jorstad., G.D.: AFS and HAC: Domain general agent simulation and control. In Baxter, J., Logan, B., eds.: Software Tools for Developing Agents: Papers from the 1998 Workshop, AAAI Press (1998) 89–96 Technical Report WS–98–10.
4. Sloman, A., Poli, R.: SIM_AGENT: A toolkit for exploring agent designs. In Wooldridge, M., Mueller, J., Tambe, M., eds.: Intelligent Agents II: Agent Theories Architectures and Languages (ATAL-95). Springer–Verlag (1996) 392–407
5. Anderson, J.: A generic distributed simulation system for intelligent agent design and evaluation. In Sarjoughian, H.S., Cellier, F.E., Marefat, M.M., Rozenblit, J.W., eds.: Proceedings of the Tenth Conference on AI, Simulation and Planning, AIS-2000, Society for Computer Simulation International (2000) 36–44
6. Schattenberg, B., Uhrmacher, A.M.: Planning agents in JAMES. Proceedings of the IEEE **89** (2001) 158–173
7. Kuhl, F., Weatherly, R., Dahmann, J.: Creating Computer Simulation Systems: An Introduction to the High Level Architecture. Prentice Hall (1999)
8. Sloman, A., Logan, B.: Building cognitively rich agents using the SIM_AGENT toolkit. Communications of the ACM **42** (1999) 71–77
9. Ephrati, E., Pollack, M., Ur, S.: Deriving multi-agent coordination through filtering strategies. In Mellish, C., ed.: Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, San Francisco, Morgan Kaufmann (1995) 679–685
10. Lees, M., Logan, B., Oguara, T., Theodoropoulos, G.: Simulating agent-based systems with HLA: The case of SIM_AGENT – Part II. In: Proceedings of the 2003 European Simulation Interoperability Workshop, European Office of Aerospace R&D, Simulation Interoperability Standards Organisation and Society for Computer Simulation International (2003)
11. Minson, R., Theodoropoulos, G.: Distributing RePast agent-based simulations with HLA. In: Proceedings of the 2004 European Simulation Interoperability Workshop, Edinburgh, Simulation Interoperability Standards Organisation and Society for Computer Simulation International (2004) (to appear).