Simulating Agent-Based Systems with HLA: The case of SIM_AGENT

PART-II

Michael Lees, Dr Brian Logan
School of Computer Science & IT
University of Nottingham UK

Ton Oguara, Dr Georgios Theodoropoulos
School of Computer Science
University of Birmingham UK

European Simulation Interoperability Workshop 2003
City Conference Centre, Stockholm, Sweden
16-19 June 2003
Motivation

- Agent-Based systems offer advantages when independently developed components must interoperate in a heterogeneous environment.
- Agents are increasingly being used in a range of different areas (computer games, mobile robotics and military simulation).
- Multi-Agent systems although extremely useful are difficult to design and test and so simulation is important.
- However,
  - No one testbed is, or can be, appropriate to all agents and environments
  - Computational requirements far exceed conventional von Neumann systems
Distributed Simulation of Agent-Based Systems

EPSRC Project No. GR/R45338/01

http://www.cs.bham.ac.uk/Research/pdes-mas/
HLA

- HLA was developed to enable different simulations on different machines to interoperate.

- Using HLA for Agent-based systems would have two obvious benefits
  - Distribution, help performance (one agent to one machine)
  - Interoperability, help development (different testbeds & toolkits)
Outline

• **Motivation**
  – Kinds of agents we want to simulate
  – The role of HLA

• **SIM_AGENT toolkit**

• **HLA_AGENT**—an HLA compliant SIM_AGENT

• **Preliminary results**
Inside an agent

- Sensors and motors are connected to a variety of internal processing modules (ovals) and
- internal short term and long term databases (rectangles)
- all performing various sub-tasks concurrently
- with information flowing in different directions (arrows).
An example architecture

- **reactive layer**: automatic processes triggered by changes in the agent or its environment

- **deliberative layer**: knowledge-based processes which explicitly consider alternative solutions

- **meta-management layer**: controls the activities of the deliberative layer
Architectures for agents

• For different kinds of agents, different architectures are appropriate:
  – in the case of an insect we would only have a reactive layer;
  – some birds and mammals seem to have a deliberative layer;
  – perhaps only humans or higher primates have a meta-management layer in its full form.
Developing and testing agents

- Multi-Agent systems are difficult to design, develop and test
- Formal verification is difficult
- Design remains largely experimental and simulation plays a key role
- A large number of simulators and testbeds have been developed
Tools for building agents

– We know from several decades of work in AI that different types of mechanisms are likely to be required

– Agents and components within individual agents act concurrently and asynchronously

– Individual agents need to be able to modify their own architectures
The SIM_AGENT toolkit

• SIM_AGENT is an architecture neutral framework for building simulated interactive agents and their environments

• Simulations consist of agents and objects, both of which can have complex behaviour(s)

• Agents may communicate with other programs or real machines via sensors and motor signals

• Example applications include: studies of affective and deliberative control in simple agent systems, simulations of tank commanders in military training simulations, controlling computer game characters
SIM_AGENT agents

• An agent consists of a collection of *modules* representing the capabilities of the agent (perception, problem-solving, communication etc.)
• Groups of modules can execute either sequentially or concurrently and with different resource limits
• Each module is implemented as a collection of rules in a high-level rule based language called POPRULEBASE
• The rules communicate via private *databases* in each agent–linked modules share databases
• Rule conditions and actions can access arbitrary procedural code, including code for neural nets and other ‘subsymbolic’ mechanisms.
Defining a SIM_AGENT Simulation

- Toolkit is object oriented with two main classes which are extended to make simulations
- Defining a SIM_AGENT simulation consists of:
  - defining one or more subclasses of the classes `sim_object` and/or `sim_agent` to represent agents and their environment
  - defining a `rulesystem` for each class or instance which specifies the internal architecture of the agent.
The SIM_AGENT scheduler

- At each timestep, the SIM_AGENT scheduler runs each agent, allowing it to:
  - sense its environment, including messages from other agents;
  - run internal processes that interpret sensory data and incoming messages and modify the agent's internal state; and
  - perform actions and send messages to other agents.
The \textbf{SIM\_AGENT} Implementation

- The SIM\_AGENT toolkit is implemented in the Poplog multi-language programming environment as a number of Pop-11 libraries:
  - PRB: a flexible forward chaining pattern-driven production system interpreter
  - SIM\_AGENT: a scheduler and some default classes and methods
An Example: SIM_TILEWORLD

- Tileworld is a well established testbed for agent architectures
- The environment consists of tiles holes and obstacles
- Agent’s goal is to fill the holes with tiles to gain points
- Environment is dynamic: tiles and holes appear and disappear at rates determined by the user
- In SIM_TILEWORLD both agent and environment are implemented in SIM_AGENT
Problems of agent simulation

• No one testbed is, or can be, appropriate to all agents and environments
• Demonstrating that a particular result holds across a range of agent architectures and environments often requires using a number of different simulators
• Simulating multi-agent systems often requires significant computational resources
• Each agent is typically a complex system and many agents may be required to investigate the behaviour of the system as a whole or even of a single agent
Agent simulation with HLA

• An HLA compliant agent simulator would have two main benefits
  
  – interoperability with different other (HLA compliant) simulators, e.g., agent testbeds & toolkits, facilitating development
  
  – distribution of agents and their environment across multiple processors, improving performance
Object Management

- Entities in the simulation are represented by *classes* of objects
- Each class of object has a set of *attributes*
- Communication between federates is achieved by creating and deleting objects and updating their attributes
- There is no global state—each federate is responsible for maintaining its own local information about objects simulated by other federates.

```
ObjectRoot
  privilegeToDeleteObject: string

TileWorld
  position: position

Agent
  CarriedTiles: TilesList

Object
  life: Integer

Obstacle

Tile
  Type: TypeEnum

Hole
  Type: TypeEnum
  Depth: Integer
```
Declaration Management

- Federates declare their interests via subscription and publication.
- In SIM_TILEWORLD, the environment creates all tiles, holes, and obstacles and publishes all attributes.
- The Agent then moves tiles and puts them in holes.

<table>
<thead>
<tr>
<th>Object</th>
<th>Federate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agent</td>
<td>Environment</td>
</tr>
<tr>
<td>privilegeToDelete</td>
<td>Publish</td>
</tr>
<tr>
<td>position</td>
<td>Subscribe</td>
</tr>
<tr>
<td>carriedTiles</td>
<td>Subscribe</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Tile</td>
<td>Publish</td>
</tr>
<tr>
<td>privilegeToDelete</td>
<td>Publish</td>
</tr>
<tr>
<td>position</td>
<td>Publish</td>
</tr>
<tr>
<td>life</td>
<td>Publish</td>
</tr>
<tr>
<td>type</td>
<td>Publish</td>
</tr>
<tr>
<td>depth</td>
<td>Publish</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Hole</td>
<td>Publish</td>
</tr>
<tr>
<td>privilegeToDelete</td>
<td>Publish</td>
</tr>
<tr>
<td>position</td>
<td>Publish</td>
</tr>
<tr>
<td>life</td>
<td>Publish</td>
</tr>
<tr>
<td>type</td>
<td>Publish</td>
</tr>
<tr>
<td>depth</td>
<td>Publish</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Obstacle</td>
<td>Publish</td>
</tr>
<tr>
<td>privilegeToDelete</td>
<td>Publish</td>
</tr>
<tr>
<td>position</td>
<td>Publish</td>
</tr>
<tr>
<td>life</td>
<td>Publish</td>
</tr>
</tbody>
</table>


Ownership Management

- HLA requires that federates own attribute instances before they can update their value.
  - for example, before an agent can move a tile, the federate on which the agent is running must obtain ownership of the tile’s *position* attribute
  - any subsequent attempt to move the tile by another agent should fail, as the tile is no longer at the position the agent perceived it
- To ensure that attributes are mutually exclusive, we only allow transfer of ownership once per simulation cycle for any given attribute
Time Management

- SIM_AGENT is a centralised time driven system where simulation advances in timesteps called cycles.

- All SIM_AGENT federates are time regulating and time constrained, so as to synchronise at the end of each cycle.

- Other federates may use different policies, e.g., a monitoring federate may be time regulating but not time constrained.
HLA_AGENT

• HLA_AGENT allows an existing SIM_AGENT simulation to be distributed using the HLA
• User simulation and SIM_AGENT library code runs unchanged
• User provides additional information specifying
  – the number of federates in the simulation; and
  – how the objects and agents are to be assigned to federates
so as to make best use of the available computing resources
• Distribution is symmetric—no additional management federates are required
HLA_AGENT federates

– Each HLA_AGENT federate corresponds to a single SIM_AGENT process
– Simulates *local* objects and agents which form its part of the global simulation
– Maintains *proxy objects* and agents representing objects of interest being simulated by other federates
– Each federate may be initialised with part of the global model or all federates can use the same simulation code with the user specifying which objects are to be simulated locally
HLA_AGENT implementation

- Uses the C++ reference implementation of RTI from DMSO
- Federate Ambassador queues RTI callbacks until the end the simulation timestep
- Add some initialisation code to connect to RTI
- Extend SIM_AGENT to hold additional data about the federation and federate (e.g. FOM)
- Wrap classes so updates to external slots are propagated via RTI
- Modify scheduler to distinguish between local and proxy objects
HLA_AGENT simulation loop

- Wait for synchronisation with other federates
- For each non-proxy object (and agent) simulated by this federate
  - run the agent’s sensors on local objects and proxies
  - transfer messages from other agents to this agent
  - run the agent’s rulesystem to determine which actions to perform
- Process each agent’s message and action queues (triggers calls to RTI to create/delete objects and propagate values)
- Process object discovery and object deletion callbacks, creating and deleting proxies as required
- Process attribute update callbacks, updating the slots of the local objects and proxies simulated at this federate
Experimental results

- We developed a HLA_AGENT Federation using SIM_TILEWORLD as a test case.
- Experiments were run on a Linux cluster, comprising 44 1.6GHz AthlonMP 1900+ processors interconnected by a standard 100Mbps fast Ethernet switch.
- Test environment was a Tileworld 20 units by 20 units in size containing 1, 2, 4, 8, 16, 32, or 64 agents:
  - the environment was simulated by a single *environment federate*.
  - the agents were simulated by one or more *agent federates* distributed over the nodes of the cluster.
SIM_AGENT on a single node

![Bar chart showing average cumulative time (seconds) for different numbers of agents, comparing elapsed time and CPU time. The chart indicates increased time with increasing number of agents.]
HLA_AGENT on a single node
Distributing the agent federates

![Graph showing time in seconds against number of agent nodes. The graph indicates a decreasing trend as the number of agent nodes increases.](image_url)
The agent federate
Summary

– Even with relatively lightweight agents like the Tileworld agents, we can get speedup by distributing agent federates across multiple cluster nodes

– However processor utilisation is low due to the high elapsed time overhead

– We would expect to see more favourable results with heavyweight agents which intrinsically require more CPU time
Further Work

• Investigate the performance of HLA_AGENT using other agent simulations

• Extend the RTI interface to take advantage of Time Management (logical time simulation) and Data Distribution Management (reducing the data transferred between federates)

• Investigate interoperation using HLA to integrate SIM_AGENT with other types of simulator