Distributed Artificial Intelligence And The HLA: Bridging The Gap

Lance Burns, PhD Student; Dr Phil Smith, Senior Lecturer
School of ITMS, University of Ballarat
l.burns@ballarat.edu.au

Abstract. The fields of Artificial Intelligence (AI) and simulation have shared close ties since their inception. Simulations have often served as theoretical test-beds for evaluating various AI algorithms, many of which have later served to enable “intelligent” control of autonomous entities within various simulations. However, as AI and simulation have extended into the domain of parallel processing, yielding distributed AI (DAI) and distributed simulation respectively, this close relationship seems to have faltered. My PhD research seeks to identify and ideally overcome the factors constraining the use of DAI within the High Level Architecture (HLA), laying the ground-work for the practical use of DAI as a decoupled, flexible service to HLA simulations. This paper provides a brief summary of the research project; examining the issues and constraining factors discovered, the proposed solution, and the findings and results gathered thus far.

1. INTRODUCTION

This research project is seeking to bridge a gap. On either side of this gap we have two fields at the forefront of computer science: Distributed Artificial Intelligence (DAI) and distributed simulation.

Traditional non-distributed AI technologies have already been harnessed by many simulations, distributed or otherwise, which have required intelligent autonomous entities or components. The introduction of organic human intelligence into a non-real-time analytic simulation is obviously not feasible, but even in the case of real-time simulation it is often undesirable that all “intelligent” entities should be controlled by humans. It may be the case that only a very limited degree of intelligence is required, for example to coarsely simulate a flock of birds; perhaps as a hazard within a flight simulator. Simulations also often require repeatability and/or predictability, persistence (saving/restoring of simulation state), or may simply require the simulation of very large numbers of entities simultaneously. Each of these scenarios are difficult or impossible to achieve using purely human-controlled entities. In order to synthesize more complicated behaviour and/or larger numbers of entities, a simulation could make use of Distributed AI technologies, which can potentially draw upon massive computational resources. This can serve to expedite results, increase data processing capacity, or both.

The High Level Architecture (HLA) simulation framework was developed to facilitate the introduction of self-contained components which handle a particular aspect of a simulation, such as weather, physics, or AI. This is made possible due to the HLA’s adherence to the fundamental software engineering concepts of low coupling and high cohesion. As such, it is expected that a HLA-compliant DAI component would be simply added to a simulation, presenting a high-level interface for use by other components, and having an otherwise minimal impact on existing hardware/software configurations. Unfortunately no such component currently exists – a situation this research project is seeking to remedy.

2. DISTRIBUTED AI

DAI is a subfield of AI which has been in existence since the mid to late 1970s [11], and concerns itself with investigating knowledge models, communication and reasoning techniques suitable for problem solving in a distributed environment [7].

DAI is a multi-tier research area, with low-level concerns being separated from high-level concerns, as suggested by Bond & Gasser [1]. They state that “DAI is concerned with issues of coordination among concurrent processes at the problem-solving and representation levels. That is, we are not concerned with parallel processing for AI for reasons of improved efficiency in AI computations per se.” Research into this high-level coordination problem has culminated in a concept known as the Multi-Agent System [4, 11].

2.1 Multi-Agent Systems

Research into Multi-Agent Systems (MAS) is collectively defined as being: “the study, construction, and application of multi-agent systems, that is, systems in which several interacting, intelligent agents pursue some set of goals or perform some set of tasks.” [11] The “agents” in DAI are autonomous problem solving components, and may be artificial (software) or human (users).

O’Hare & Jennings [7] states that a MAS has significant advantages over a single, monolithic, centralized problem solver: Faster problem solving by exploiting parallelism; decreased communication by transmitting partial solutions to other agents rather than raw data to a central site; and the potential for increased flexibility by having heterogeneous agents with different abilities dynamically team up to solve current problems; and increased reliability by allowing agents to take on responsibilities of agents that fail.
Agents in a MAS may be identical (homogenous), or exhibit one of three differing levels of heterogeneity [7]. “High heterogeneity” is characterised by agents that only share a common interaction language, their other characteristics being potentially quite different. “Medium heterogeneity” is characterised by agents which only differ in problem solving methods and knowledge they utilise. Agents which exhibit “low heterogeneity” differ only in the resources available to them.

The actual functionality of an agent varies greatly between individual implementations, as agents are essentially an abstract concept. For example, an agent may be AI-based computer code, a human, a static mathematical function or script, or even a combination of these elements. Within the scope of this research project, an agent refers to an automated software component which embodies a set of Distributed Problem-Solving (DPS) methodologies (otherwise known as a reactive agent). The eventual solution will exhibit low heterogeneity, due to the HLA’s nature as a largely hardware/software resource independent platform.

2.2 Distributed Problem-Solving Methodologies

DPS methodologies underlie many DAI systems, and are commonly parallel extensions of existing non-distributed techniques such as expert systems, neural networks and genetic algorithms. These three examples were not chosen at random; they represent a cross-section of machine learning paradigms: symbolic, connectionist and social/emergent respectively. In this section, we will briefly examine the three DPS techniques which form the low-level mechanics of the agents in the proposed MAS-based solution.

2.2.1 Parallel Expert Systems

Parallel expert systems, such as the one described in Noyes [6], implement a slightly more advanced inference engine than their non-parallel siblings. Instead of a single linear search through inference rules, this engine instead divides the rules between available processing nodes. This results in a theoretically linear reduction in rulebase search time, assuming nodes of identical processing capacity.

2.2.2 Parallel Artificial Neural Networks

A Parallel Artificial Neural Network (PANN) benefits from the inherently parallel structure of an ANN [12], whereby the artificial neurons and layers operate independently of one another. Hidden layers, which perform the majority of processing in most large neural networks [5], may be divided amongst multiple processing nodes [10]. The output/activation function(s) and weights associated with the hidden-layer neural connections may also be stored and processed by each of these nodes.

2.2.3 Parallel Genetic Algorithms

In “Parallel Genetic Algorithms”, Shonkwiler [8] gives a universal method for parallelizing a genetic algorithm. This method simply executes the same algorithm on each processing unit independently of one another, but with a different set of initial candidate solutions. Communication between nodes occurs only to gather solutions. Analysis showed that the technique achieved super-linear parallel speedup using $m$ processors; given by $ms^{m-1}$ where the acceleration factor $s$ is a parameter depending on the details of the genetic algorithm, but typically $s > 1$ [8].

2.3 HLA_AGENT

Before delving into our own solution, it is necessary to examine the only known previous attempt at a solution to this problem; and indeed the only known example of DAI technology being used in conjunction with the HLA at the present time: HLA_AGENT [3]. HLA_AGENT is a tool which distributes the SimAgent toolkit (an architecture for DAI – discussed in the next section) using the HLA as its low-level distribution mechanism.

The HLA_AGENT system divides up its constituent agents and objects across a number of federates. The SimAgent agents themselves are not network-distribution capable, and rely on the presence of multi-processor hardware/external software mechanisms for distribution – in this case, a Linux cluster of 32 dual-2.6GHz Xeon-based nodes. The RTI (Run Time Infrastructure) used for experimentation was the DMSO C++ reference RTI. The federation consisted of a federate which simulates a simple “Tile World”, consisting of “tiles”, “holes” and “obstacles”, and one or more HLA_AGENT federates containing a number of agents which each individually attempt to push the tiles into holes while avoiding obstacles.

Using a single HLA_AGENT federate with 64 reactive, standalone agents running on a single cluster node, the communication overhead induced by communicating via the HLA RTI was measured as high as 54% when compared with a standalone SimAgent reference implementation. While this already represents a significant reduction in performance, this problem would be massively compounded in the case of a distributed set of cooperating agents, which would have to communicate low-level problem-related messages via the RTI under this arrangement.

HLA_AGENT was created with the aim of enabling inter-operability between a MAS and an HLA simulation. While the project achieved this aim to some degree, it has done so at the expense of the HLA ideals of reusable, self-contained components, and its large degree of hardware/software independence. The nature of the simulation must be programmatically accounted for in the HLA_AGENT federate, resulting in tight coupling with the simulation, and a marked reduction in cohesion within the MAS. In contrast to HLA_AGENT, this research project seeks application independence in
accordance with HLA ideals. In addition, the requirement of specialist hardware/software configurations to achieve distribution would also constrain the use of a DAI service, so this must be avoided.

2.3.1 The SimAgent Toolkit

There are a number of existing architectures for DAI, both academic (such as SimAgent) and industrial (such as ARCHON; Architecture for cooperative heterogeneous on-line systems [7]), but most are unrelated to simulation. An exception to this rule is the SimAgent toolkit, which is essentially a self-contained simulation framework in itself (not HLA related). Like most modern DAI architectures, SimAgent focuses on the development and application of Multi-Agent Systems and their constituent agents.

The SimAgent toolkit (formerly known as SIM_AGENT) provides a range of resources for research and teaching related to the development of simulations containing interacting agents. SimAgent is based on the currently Linux-exclusive Poplog programming environment, implemented in its core language: POP-11. SimAgent (like ACT-R, COGENT, and the original SOAR) is primarily designed to support design and implementation of very complex agents, each composed of very different interacting components (like a human mind) where the whole thing is embedded in an environment that could be a mixture of physical objects and other agents of many sorts [9].

In contrast to the goals of this research project, SimAgent is not a service, but rather a self-contained simulation environment for research and teaching with Multi-Agent Systems. As mentioned in the previous section SimAgent is also not natively network distributed, instead relying on external layers of software/hardware to perform distribution.

2.4 Significance Of The Problem

Review of the literature indicates that no system currently exists which offers Distributed Artificial Intelligence technology as a flexible, decoupled service to HLA simulations. This means that simulation developers wanting to harness AI will likely avoid DAI altogether, limiting not only simulation accuracy and/or capacity, but also further research in this area. If needed, they are currently forced to dedicate time, money and resources to developing solutions which are likely to be bespoke (simulation specific solutions), tightly coupled with the low-level simulation code, bound to a problem-specific DAI methodology and/or require specialised hardware/software configurations.

3. PROPOSED SOLUTION

The proposed architecture provides the means of making available a decoupled, flexible DAI service to an HLA simulation. The underlying DAI component model is a simple Multi-Agent System whose agents exhibit low heterogeneity (due to potentially differing software/hardware resources). The agents themselves will embody a simple set of core Distributed Problem-Solving techniques; namely a parallel expert system, a parallel artificial neural network, and a parallel genetic algorithm. These three techniques were chosen on the basis that they represent a cross-section of the three fundamental categories of machine learning: Symbolic, connectionist and social/emergent, respectively.

In reviewing the literature, it was found that the overhead of passing low-level DAI messages through the RTI has an acceptably large impact on performance. Passing DAI messages through the RTI would also reduce its processing capacity for handling messages from non-DAI federates, resulting in an overall drop in simulation performance.

![Figure 1: HLA simulation with federate using DAI](image)

To combat the issue of RTI communication overhead, the low-level mechanics of the proposed architecture are completely abstracted from the HLA simulation, and present a single HLA-compliant high-level interface to the simulation via a DAI federate, as seen in Figure 1. This is an application of the façade design pattern [2], which aids in the development of decoupled, cohesive software subsystems (i.e. a service). Because the low-level DAI implementation is invisible to the simulation, it may also be maintained, modified, or replaced entirely – providing that the interface is not changed – without any impact on the remainder of the simulation.

In an effort to minimise the impact upon simulation code, a local middleware component abstracts all extra functionality required for integration with the DAI federate. The middleware abstracts the functionality by exposing a simplified, code-level API (Application Programming Interface) to its parent federate. Note that Federate 2 shown in Figure 1 does not require the DAI service, and is unaffected by its introduction into the simulation; highlighting the decoupled nature of the solution.

3.1 Adherence To Requirements

This section examines the design as it stands against the three principal requirements which were derived from the literature review and subsequent research. For the
sake of clarity each has been divided into its own subsection.

3.1.1 Requirement 1: Flexible

*The solution must provide a suitable platform which can support modern DAI techniques, without violating the other requirements.*

Due to its agent-based design, this architecture is capable of supporting a massive variety of Distributed Problem-Solving techniques. Middleware forms a core part of the design, allowing the architecture to present a decoupled, consistent API to the HLA simulation, yet with the potential for flexibility in the underlying methodologies used through abstraction.

3.1.2 Decoupled

*The solution must be highly cohesive and must be minimally coupled with any HLA federate intending to utilise its services.*

The use of a high-level API via middleware means that the architecture has a large degree of application independence via abstraction, giving extensive scope for reuse, easier maintenance, and generally reduced development costs. By avoiding RTI communication altogether for the low-level transactions between agents, the solution is vastly simpler to retarget to a different RTI (many of which are proprietary and differ slightly from one another). Retargeting to completely different distributed simulation architectures is also far more feasible, as theoretically, only the middleware API layer and simulation-side sections of the DAI federate would need to be redeveloped.

This solution is also independent of the existing object model, only requiring the incorporation of its own internal object model for HLA compliance, and to allow other federates to send and receive interactions and objects related to the DAI service via the middleware. Because the middleware is responsible for handling the DAI-related objects/interactions, the object model itself is abstracted away from the high-level middleware API (and hence the client federate code).

3.1.3 Requirement 3: Efficient

*The system should not require specialist hardware to operate, and should strive for efficiency wherever it does not contravene the other requirements.*

The design work has been performed with efficiency in mind. Separation of low-level DAI communication from the simulation results in a minimisation of communication overhead, while decoupled design serves to enhance internal cohesion, resulting in minimal amounts of superfluous “glue” code. In supporting existing modern Distributed Problem-Solving techniques, the MAS automatically benefits from the increased performance offered by these methodologies.

In addition to this inherent efficiency, *load balancing* is also performed within the MAS. Lees, et al. [3] states that “without load balancing, the speedup that can be obtained is limited by the elapsed time for the slowest component”. As the HLA was designed to accommodate distribution across heterogeneous hardware platforms, load balancing becomes especially important. For example, one DAI node may be running on a 200MHz processor, while another is running on a computer with four 2GHz processors. Clearly the former should not be required to process as much data as the latter, which conversely should be assigned a much larger portion of the processing.

Within the context of this research project, load balancing may be achieved in several ways. For instance, a node at maximum processing loading could reject incoming solution requests. As such the DAI federate would queue requests until one or more nodes became available for new processing tasks. In the case of a Parallel Expert System, the number of inferences performed by a node could be directly proportional to the available computational resources. PANNs would be similarly distributed, with a dynamic number of layers/neurons being allocated to each node. The method for Parallel Genetic Algorithms described in Shonkwiler [8] seems to provide load-balancing simply by its nature, with a slower node simply contributing less potential solutions during the evolutionary process.

4. CONCLUSION

A flexible, decoupled architecture for Distributed Artificial Intelligence with a high-level HLA-compliant API would cut costs and reduce development time for simulation developers requiring such a service, and would allow for further research within the nexus between DAI and distributed simulation. It would allow developers to focus on development of the simulation logic, and enhance their ability to design and develop highly cohesive, decoupled simulation components.

A solution has been proposed which satisfies the requirements previously stated: That the architecture must be flexible enough to support modern DAI technologies, that it must be as independent of the client application/simulation as possible, and finally that it should strive for efficiency; which it achieves not only through its flexibility, but also through its decoupled design (which allows for greater degrees of internal cohesion), and through application of proven software engineering techniques such as design patterns and middleware.

4.1 Project Progress / Future Work

As per the project methodology, a reference implementation of the design is undergoing development – entitled “Encephalon” (Greek for mind or brain) – and is being used as a platform for experimentation, observation and evaluation. At the time of writing, extensive testing of the Parallel Artificial Neural Network capabilities of the agents is currently underway. Figure 2 below shows an
Encephalon agent in the process of training a PANN to perform Optical Character Recognition (OCR). The bar graph at the top is displaying the decreasing mean square error being produced by the PANN over time.

Figure 2: An Encephalon agent training a PANN.

THE QUICK BROWN FOX

Figure 3: Small sample of an example query.

In this particular experiment the Encephalon system was able to recognize large sentences which exhaustively utilized the training data (the alphabet) with 100% accuracy (consistent over 10,000 testing cycles). A small sample of one of these sentences can be seen in Figure 3.

THE QUICK BROWN FOX

Figure 4: 100% (relative) 8-bit noise.

THE QUICK BROWN FOX

Figure 5: 25% maximum-severity additive noise.

THE QUICK BROWN FOX

Figure 6: 50% maximum-severity additive noise.

The PANN was also very noise tolerant, with 100% accuracy against very noisy data such as the example shown in Figure 4. Figure 5 shows the same data subjected to random additions of maximum-value pixels. In the case of 25% coverage of such pixels, the average accuracy fell to 97%. With 50% coverage as shown in Figure 6 (which renders the text very difficult to read even for a human) average accuracy dropped further to 75%.

At the time of writing the project is at a stage where the implementation has just reached a level of maturity at which meaningful experimentation can be performed. Preliminary results seem to validate the real-world operational capabilities of the system, but extensive/concrete results are expected to be attained within the first half of 2008.

REFERENCES