Interfacing RePast with HLA Using a Generic Architecture for COTS Simulation Package Interoperability

Yuanxi Liang  
Wentong Cai  
Stephen J. Turner  
Parallel & Distributed Computing Centre  
School of Computer Engineering  
Nanyang Technological University  
Singapore 639798  
65-6790-4110, 65-6790-4600, 65-6790-5045  
{yxliang | aswtcai | assjturner}@ntu.edu.sg  

Georgios K. Theodoropoulos  
Rob Minson  
School of Computer Science  
University of Birmingham  
Edgebaston, Birmingham  
B152TT, UK  
44-121-414-4780, 44-121-414-3710  
{g.k.theodoropoulos | r.minson}@cs.bham.ac.uk  

Keywords: HLA, RePast, COTS simulation package, interoperability

ABSTRACT: A Distributed Multi-Agent System (MAS) is often appropriate in solving large-scale experimental problems. RePast is a well-known simulation toolkit for agent-based modeling. In order to extend the scalability of RePast, one way is to partition the model into a distributed simulation. Interfacing RePast with the High Level Architecture (HLA) is a way to build the distributed system. This has previously been achieved in the HLA_RePast system by using a traditional interfacing approach which is time-consuming in the development process. In this paper, a new approach to interfacing RePast with the HLA is introduced. It is a pioneering attempt to adopt a generic architecture for Commercial Off-The-Shelf (COTS) simulation package interoperability to interface an agent-based simulation package with the HLA. A middleware called DSManager originally designed to support COTS simulation package interoperability, is proven to be able to support interoperability of RePast models in a fast and easy manner. The advantages and tradeoffs, and the performance of the new distributed RePast system are discussed and compared to the HLA_RePast system.

1. Introduction

There has been much interest and research done in the area of multi-agent systems in recent years. A multi-agent system is a system composed of several agents. An agent describes a software abstraction capable of acting by itself in order to accomplish tasks. Agents are autonomous and are capable of making decisions by perceiving the context in which they are operating without human intervention. In a multi-agent system, the agents normally communicate with each other to accomplish a greater goal than that accomplished by each agent individually. Multi-agent simulations have a wide-range of applications in various areas of engineering, investigation of human and social dynamics, and evaluation of military operations. RePast [1] is a popular Java-based multi-agent simulation toolkit.

The multi-agent paradigm allows the simulation to be scalable in terms of complexity by changing the behavior of the agents for more complex situations. However, increased complexity of behavior as well as an increased number of agents will lead to more processing which slows down the simulation. A distributed system approach is then desirable to distribute the simulation so that the workload on each of the computers is reduced. To distribute the
Simulation among many computers, the possibility of interfacing RePast with the High Level Architecture (HLA), an IEEE standard for distributed simulation, has been explored [2].

**DSManager** [3] is a generic interface consisting of a set of functions to be invoked by a Commercial Off-The-Shelf (COTS) Simulation Package (CSP) for interfacing CSPs with the HLA. It is designed to hide the HLA concept from both the CSP and the modeler and provides an easy and fast approach to building up a distributed simulation. The Simulation Interoperability Standards Organization’s (SISO) CSP Interoperability Product Development Group (CSPI-PDG) has developed and standardized a set of Interoperability Reference Models (IRMs) [4], aimed to clearly identify the interoperability capabilities of CSP-based distributed simulations. The **DSManager** has been used to support Type A.1 and Type A.2 IRMs [5, 6]. It has also been applied to support the interoperability of simulation models developed using Autosched AP [7].

In this paper, we will demonstrate how the **DSManager**, originally designed as a generic architecture to support COTS simulation package interoperability, can also be used to support interoperability of RePast models so that a distributed simulation execution can be formed via the HLA Runtime Infrastructure (RTI). Section 2 gives an overview of related work in terms of current achievements, focusing in particular on the HLA RePast system. Section 3 describes the system we developed using the **DSManager** – the RePast_JDSM system. In particular, the scheduling system and time management issues are discussed. In section 4, the RePast_JDSM system is compared and contrasted with the HLA_RePast, while the experiments and results are discussed in Section 5. Finally, Section 6 gives conclusions and future work.

2. Related Work

2.1 The RePast Simulation Toolkit

The RePast simulation toolkit is a Java-based platform for the development of agent-based simulation models [1]. It has been adopted for the development of multi-agent simulation experiments for social phenomena research.

On one hand, RePast provides a set of pre-defined library classes for the development of different components in an agent-based simulation model, such as the simulation engine, the event scheduler, the display and GUI, and so on. On the other hand, it is a flexible development platform that allows the modelers to modify and extend the functionality of the library classes purely based on Java programming language.

In RePast, the **Schedule** class is one of the most important library classes which is an implementation of a discrete event scheduling engine [2]. Events are associated with instances of the **BasicAction** class where a single method, **execute()**, is defined. This method contains the actual actions to perform the event, and the invocation of this method represents the occurrence of the event.

Figure 1 shows the standard RePast Model-Executive interface. The model contains an event schedule queue. In runtime, the executive extracts the schedule from model, executes events and then modifies the model states accordingly.

![Figure 1: The Standard RePast Model-Executive Interface](image)

2.2 The High Level Architecture and the Runtime Infrastructure

The High Level Architecture (HLA) is an architecture for reuse and interoperability of simulations. The initial intent of the HLA is to form a complex simulation system by reusing the existing simulation components, which can reduce the cost and time required to develop a new simulation application. The term federate refers to a simulation component, and a federation is a collection of federates interacting with each other for a common purpose. The Runtime Infrastructure (RTI) is the software used to support a federation execution conforming to the HLA standard [8]. The HLA can also be used to support a distributed simulation system where a large-scale model is partitioned into a number of federates so as to achieve an execution speedup.
2.3 The HLA_RePast System

The HLA_RePast system is an extension of the RePast simulation toolkit which enables the execution of a distributed simulation with multiple interacting instances of RePast models using the HLA [2]. With the HLA_RePast system, the modelers can partition a large-scale model into a collection of small components. Figure 2 shows an illustration of an HLA_RePast federation.

The system provides a middleware layer to detect the occurrence of events in the RePast model and transmits them via the RTI. The PublicObject scheme used in the HLA_RePast system provides mechanisms for mapping RePast state-transitions to the HLA events.

To ensure a maximum of transparency in scheduling, a new algorithm replaced the normal RePast scheduling algorithm for stepping through the scheduler. The system does not execute any event in the local schedule until all external events with timestamp < T are received (T is the time of the next event in the local schedule). An external buffer is used to store events received from the RTI. Once T is granted by the RTI, the events in the external buffer with timestamp < T will be executed. After that, the system executes the events in the local schedule [9].

2.4 A Generic Architecture for COTS Simulation Package Interoperability

In order to integrate different CSPs based on the HLA, a generic architecture (Figure 3) has been defined conforming to the CSP Interoperability Product Development Group (CSPI-PDG) standard [10]. Based on this generic architecture, the time management and data exchange mechanisms between different components are enclosed in the middleware and hidden from the modelers. It provides an easy approach for integrating different CSPs with the HLA/RTI.

There are five requirements summarized in [11] for using this architecture:

- (R1) The ability to initialize the distributed simulation prior to simulation execution
- (R2) The ability to suspend the simulation execution
- (R3) Access to the time of the next event to be simulated
- (R4) The ability to introduce new events/entities from an external source into the event list
- (R5) Access to information of simulation objects/entities that are shared among federates
A library, DSManager, written in C++ was developed by Wang to act as the middleware layer in the architecture [12]. This library has been used to support Type A.1 General Entity Transfer and Type A.2 Bounded Receiving Element Interoperability Reference Models (IRMs) [4, 5, 6]. In CSPs, an entity is represented by a message and is the basis of information exchange between models. A set of basic methods are available for the modelers to create a federation, to register incoming and outgoing entities, and to transfer and receive entities. Figure 4 shows the basic interactions between CSP, DSManager and RTI [5]. With the DSManager library, this generic architecture has been applied to interoperate different CSPs like AutoSched [11] and AutoMod [13].

3. Interfacing RePast with HLA Using the Generic Architecture Approach

3.1 The Middleware – JDSManager Library

In order to integrate the RePast simulation toolkit with the HLA, a Java binding version of the DSManager library known as JDSManager was developed. The Java Native Interface (JNI) [14] was used to create the wrapper library classes in Java. When a call to JDSManager is made, the DSManager library will be invoked and the call will be transferred from JDSManager to DSManager to perform the actions. Associated with the DSManager library, JDSManager provides four basic groups of methods for federation management, entity declaration management, object transfer management and time management (Table 1).

<table>
<thead>
<tr>
<th>Function</th>
<th>Name of method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Federation management</td>
<td>RegisterController()</td>
</tr>
<tr>
<td></td>
<td>RegisterDistributedSimulation()</td>
</tr>
<tr>
<td></td>
<td>SetSynMethod()</td>
</tr>
<tr>
<td></td>
<td>SetLookahead()</td>
</tr>
<tr>
<td></td>
<td>TerminateDistributedSimulation()</td>
</tr>
<tr>
<td>Entity declaration management</td>
<td>RegisterIn/OutEntity()</td>
</tr>
<tr>
<td></td>
<td>GetIn/OutEntityID()</td>
</tr>
<tr>
<td></td>
<td>GetIn/OutEntityName()</td>
</tr>
<tr>
<td>Object transfer management</td>
<td>Set/GetAttributeValue()</td>
</tr>
<tr>
<td></td>
<td>TransferEntity()</td>
</tr>
<tr>
<td></td>
<td>ReceiveEntity()</td>
</tr>
<tr>
<td></td>
<td>GetReceivedEntityTime()</td>
</tr>
<tr>
<td></td>
<td>GetValueSize()</td>
</tr>
<tr>
<td>Time management</td>
<td>AdvanceTime()</td>
</tr>
</tbody>
</table>

Table 1: Main Methods in the JDSManager Library

3.2 Adapting RePast Using the Interoperation Framework

To adapt RePast for interoperation using the defined architecture and the JDSManager library, we studied if RePast satisfies the five requirements stated in section 2.4. Since RePast is purely based on Java, it has a high extensibility and flexibility for model development, and the five requirements are easy to follow and are discussed as follows.

(R1) The ability to initialize the distributed simulation prior to simulation execution

In RePast, the setup() method is an abstract method in the SimModelImpl class that has to be implemented by the modelers for simulation initialization. It is called before the simulation execution, and hence the HLA/RTI initialization codes can be placed in this method.

(R2) The ability to suspend the simulation execution

Since RePast is based on Java, the simulation execution process is fully controlled by the modelers in coding. The simulation execution can be suspended having a specific class built for explicit simulation control.

(R3) Access to the time of the next event to be simulated

The future event list is invisible in RePast and we cannot access the time of the next event directly from the common methods provided by the Schedule class. In order to access this information, we extended the Schedule class by adding a future event queue, which will be discussed in section 3.4 in more detail.

(R4) The ability to introduce new events/entities from an external source into the event list

The Schedule class has a number of methods to schedule actions into the event list before and during the simulation execution. The modelers can define any responding reaction after receiving messages from external sources.

(R5) Access to information of simulation objects/entities that are shared among federates

The information of simulation objects/entities can be defined as any type of attributes in Java, which are accessible to the model. To share the objects/entities, the attribute values are passed to setAttributedValue() method in the JDSManager library, and then are transferred to other federates.
3.3 The RePast_JDSM System

By adopting the generic architecture and integrating the middleware libraries, RePast is capable of running a distributed simulation using the RTI. Figure 5 shows the overall architecture of the integrated system which is named RePast_JDSM. The ScheduleExt is an extended version of the Schedule class which has a queue storing the time of the future events (see section 3.4), and a time management mechanism (see section 3.5). The action of updating attribute values to other federates, known as the entity sending process in RePast_JDSM, is initiated in the model, and then the entities are passed to the RTI through the middleware JDSManager. On the other side, the external entities are transferred from the RTI through the middleware to the ScheduleExt class where the associated events will be executed.

3.4 Scheduling

Since there is no direct access to the time of the next event in the future event list in RePast, we extended the Schedule class by adding a future event queue and have an explicit mechanism to update this queue.

A queue data structure called nextEventQ is added into the extended Schedule class known as ScheduleExt. This queue stores the timestamp information of the future events in the simulation, and the timestamps are sorted in increasing order. The nextEventQ only serves as a role of collecting and providing the time information of the future events in the model, and it does not replace the original schedule queue of RePast or activate any action. Three methods are also added to manipulate the queue, which are addNextTime(), getNextTime(), and removeNextTime(). In RePast, there are two main groups of methods to schedule actions into the simulation engine. The first group of methods is used to schedule actions at a specific logical time, the other group of methods is used to schedule actions based on a specific logical time interval so the actions will be executed repeatedly.

The update of the nextEventQ is handled explicitly by the modelers. In the first case, when an event is scheduled at a specific logical time, the timestamp information of this event is put into the nextEventQ by calling the addNextTime() method. When the event happens, within the execute() method of the associated BasicAction class object, the associated timestamp information will be removed from the nextEventQ by calling the removeNextTime() method.

In the second case, when an event is scheduled based on a logical time interval, similarly, the first happening time of this event is put into the nextEventQ. However, when the event happens, the associated timestamp information will be first removed from the nextEventQ. After the current event finished, the next happening time of the same event is put into the nextEventQ again within the execute() method of the associated BasicAction class object. Hence, the timestamp information of events scheduled based on a time interval will be recorded by the nextEventQ repeatedly.

Based on the mechanism described above, the time of the next event to be simulated in the model is available for time management.

3.5 Time Management

In interfacing RePast and the JDSManager library with the HLA, one important step is to implement the time management mechanism. In general, two synchronization methods are usually used, which are Conservative synchronization and Optimistic synchronization [12]. Because optimistic synchronization requires a rollback mechanism at each federate and requires more effort to implement, conservative synchronization is preferred.

The conservative synchronization mechanism is placed in the preExecute() method in the ScheduleExt class. The basic algorithm is described in pseudo code in Figure 6. Before each event is executed, the preExecute() method will be invoked and it will first check whether the execution of the event is safe to proceed. If the time of the next event is greater than the last granted time from the RTI, it may invoke the advanceTime() method in the JDSManager library with the next event time. After the time is granted, the entities received from other federates will be handled according to their timestamps. The associated events will be executed. The execution of events may...
schedule future events, if any, into the original schedule queue of the RePast simulation engine and the `nextEventQ` may need to be updated accordingly.

1. NET = time of next event in `nextEventQ`
2. GT = current granted time by RTI
3. If (NET>GT) Then
4. GT = invoke `advanceTime()` method with NET to RTI
5. NER = number of entities received from RTI
6. For each of the entities received Loop
7. If (timestamp<=NET) Then
8. Execute the associated event
9. End If
10. End For_Loop
11. End If

**Figure 6: Time Advancing Algorithm**

### 4. Comparison between RePast_JDSM and HLA_RePast

#### 4.1 Agent State Update

In agent-based simulation, the states of each agent are updated according to other agents’ states and some common environment states. When an agent-based simulation model is distributed, the update of each agent’s states and the update of the common variables are critical to the correctness of the simulation.

In the HLA_RePast system, the update of agent’s states is handled by the `PublicObject` scheme [9]. The state update of each agent is monitored by the `LocalManager` class which will then broadcast the updates to all other federates through the RTI. In fact, the mechanism of publicizing the state update is encapsulated inside the `LocalManager` class and is hidden from the modelers. This reduces the work of dealing with explicit agent state update in the development process.

In contrast, for the agent state update, the RePast_JDSM system provides the fundamental mechanism of transferring and receiving messages between federates. The agent state update to other federates is explicitly handled by the entity transferring process, while getting the state updates from other agents is handled by the entity receiving process. Instead of broadcasting the state updates to other federates, RePast_JDSM actually explicitly sends the state updates to individual federate. Although it might require a greater effort in dealing with the agent state update explicitly, the RePast_JDSM system gives a better flexibility to the modelers in the design of different distributed agent-based simulation models.

#### 4.2 The Scheduling System

The scheduling system is another important component in agent-based simulation that schedules and triggers events and state updates at desired times. In distributed simulation, the scheduling system needs to handle external events from other federates and execute them at correct logical times.

Figure 7 shows the scheduling system of HLA_RePast and RePast_JDSM. In the HLA_RePast system, the scheduling system is represented by an extended distributed `Schedule` class shown as `DisSchedule` [9]. A local schedule queue replaces the original `schedule` queue and handles local events and state updates. It also triggers HLA updates to the RTI from local events. In addition, an external schedule queue is added to receive state updates from the RTI and then trigger updates to the states in the model.

The RePast_JDSM system extends the `Schedule` class using a different scheduling strategy from the HLA_RePast system. It does not replace the original schedule queue, but manipulates it according to the external entities and messages received from the RTI. The original schedule queue still triggers state updates in the local model, and hence triggers the action of transferring messages to other federates though `JDSManager`. The `ScheduleExt` class handles the process of receiving entities from other federates with a time management mechanism, and then executes or schedules the associated events in the original schedule queue according to the timestamps. This is also discussed in section 3.5 as a part of the time management algorithm. Based on the analysis above, HLA_RePast has a more complex scheduling system for agent state update than RePast_JDSM.
4.3 Preferred Partition Strategies

Three basic partition strategies are generally used for distributed multi-agent simulation systems [9] as listed below.

1. **Centralized Environment Partition**: one federate models the environment, and N federates model agents.

2. **Spatial Partition**: the environment is divided into N physical regions across federates, and agents move from federate to federate as they move through the environment.

3. **Component Partition**: for an agent architecture with N discrete cognitive components, each component is modeled by one federate. Here each federate houses M separate instances of its given component, where M is the number of agents in the model.

Particular to the first and third partition strategy, the variables in the environment are shared by all the agent federates, and hence, they require a conflict resolution mechanism to handle the situation that two agents in two different federates try to write the same variable simultaneously. In the spatial partition strategy, the conflict of shared variable update can be eliminated as the environment variables are owned by different spatial partition federates, and agents are only able to acquire write ownership of a shared variable within the same spatial partition federate. However, to adopt the second strategy requires an agent migration mechanism in the system. For the third partition strategy, both the conflict resolution for shared variables and an inter-component communication mechanism are required.

Based on the PublicObject scheme, the HLA_RePast system provides three types of shared variable for conflict resolution, which are Exclusive, Cumulative and Viewable [2]. The implementation of this conflict resolution mechanism is also encapsulated and invisible to the modelers. In summary, with the three types of shared variable in the models, HLA_RePast is able to use a centralized environment partition strategy without additional effort. However, without an explicit entity transferring and receiving mechanism, agent migration or inter-component communication is difficult to achieve.

For the RePast_JDSM system, it is obvious that the spatial partition strategy is preferred because it has an explicit entity transferring and receiving mechanism. Yet, the first and the third partition strategies are also possible to adopt if a conflict resolution mechanism is added. Since the fundamental mechanism for communication with the RTI is provided, RePast_JDSM can be extended to have a more complex mechanism for conflict resolution, but this requires the modelers to deal with the implementation explicitly and hence more effort in the model development process is required. In contrast, HLA_RePast is difficult to be extended, but it requires a minimum effort from the modelers in dealing with conflict resolution.
5. Performance

Because in an agent-based simulation model, the most frequent activity is to share the states with other federates, we decided to compare the efficiency of sharing agent’s states between federates, and designed a distributed model for the experiments.

The model consists of two federates and each federate has the same number of agents. Each agent has a variable that is shared to the associated agent at the other federate. In each time step, each agent performs the action of updating its owned variable with a certain probability, and performs the action of reading the shared variable owned by the associated agent at the other federate if the shared variable was updated. This process can be modeled straightforwardly in the HLA_RePast system by simply using the PublicObject and shared variable scheme. In the RePast_JDSM system, this is modeled using the entity transferring and receiving mechanism. In each time step, each agent updates its owned variable and sends an update message to the associated agent with a certain probability, and receives the update message from the other federate if any.

The experiments were run on three computers with two Intel Pentium D 3.2GHz CPUs, 2GB of RAM and Windows XP operation system. Each federate was run on one computer, and the NG_RTI 1.3 was run on the third computer. All the computers are connected by a 100Mbps network.

The first experiment was performed by varying the number of agents at each federate from 1 to 8192. The probability to perform state update activity is 1. Since the variations between each run were relatively small, the elapsed time of 500 iterations was collected by taking the average of the measurements from 3 separate runs.

Figure 8 shows the results from 1 to 4096 agents with a logarithmic X-axis. The elapsed times for the HLA_RePast system almost remain the same at about 150 seconds until the number of agents at each federate exceeds 256, while the elapsed times for the RePast_JDSM system are almost the same at about 20 seconds until it has more than 32 agents on each federate. In Figure 8, the elapsed time for RePast_JDSM seems to be getting closer and closer to HLA_RePast. However, Figure 9 shows the full set of results from 1 to 8192 agents with a linear X-axis, which clearly indicates that the elapsed time for HLA_RePast starts to grow faster than RePast_JDSM from 4096 agents.

In the first experiment, two costs are increasing as the number of agents increases, which are the cost of the scheduling system, and the communication cost via the RTI. When the number of agents is small, the overall cost mainly represents the overhead of the scheduling system, and RePast_JDSM has a scheduling system with less overhead than HLA_RePast. This reflects the fact that the scheduling system in HLA_RePast is more complex than the one in RePast_JDSM as discussed in section 4.2. However, when the number of agents increases, the communication cost becomes the major component in the overall cost. Because of the simpler scheduling mechanism, the communication cost becomes a dominating overhead factor sooner in RePast_JDSM than HLA_RePast. This explains the observation in Figure 8 that the elapsed time for RePast_JDSM seems to be closer and closer to HLA_RePast. When the number of agents increases further, the communication cost of HLA_RePast grows faster than RePast_JDSM. This explains the fact in
Figure 9 that the elapsed time of HLA_RePast is much higher than RePast_JDSM when the number of agents is 8192 on each federate.

Based on the discussion above, we can conclude that RePast_JDSM has a lower cost of the scheduling system, but the communication cost becomes a dominating overhead factor sooner than HLA_RePast.

The second experiment was performed by varying the probability of state update activity from 0.1 to 1 while the number of agents remained the same. The elapsed time of 500 iterations was collected by using the average of the measurements from 3 separate runs. Figures 10 and 11 show the results based on the conditions that the federates have 1024 and 4096 agents respectively. The elapsed time for RePast_JDSM grows linearly in a smooth manner. On the other hand, the one for HLA_RePast also tends to grow linearly but in a relatively irregular manner, in that it grows slower when the probability is low than when the probability is high. This observation may be explained by the fact that the state update activity in RePast_JDSM is handled explicitly by message passing without other overhead, so that the communication cost growth rate is close to a constant. In contrast, the state update in HLA_RePast is handled implicitly and hence it is possible that some internal process in the system may influence the outcomes.

In the second experiment, the communication cost via the RTI also varies with the probability of state update activity. The results give a similar conclusion as the one in the first experiment that the communication cost of HLA_RePast grows slower when the amount of state update activities is low, but grows faster when the amount of state update activities is high, comparing to the case of RePast_JDSM.

In summary, based on the simple model, the RePast_JDSM system has a higher efficiency of sharing agent’s states between federates, with a smaller overhead of the scheduling system than the HLA_RePast system. The communication cost of RePast_JDSM grows linearly when the amount of state update activities increases. This reflects the fact that RePast_JDSM uses the basic mechanism of transferring entity for state update and a lighter weight scheduling system than the one in HLA_RePast.

6. Conclusions and Future Work

In this paper, we describe our work on interfacing the RePast simulation toolkit with HLA using a generic architecture for COTS simulation package interoperability. In our work, the middleware JDSManager acts as the communication bridge between RePast and the RTI. Based on the discussion of the five requirements for adopting the generic framework, RePast has been proven to be suitable to adopt this generic approach. The RePast_JDSM system has an extended scheduling system to handle agent state update and external events dynamic scheduling with a time management mechanism based on conservative synchronization.

Compared to the HLA_RePast system, the RePast_JDSM system uses a simpler approach to interface with the HLA, and it provides an explicit entity transferring and receiving mechanism for sharing agent’s states. On one hand, RePast_JDSM has a greater flexibility and extensibility, because the open architecture allows more complex mechanisms to be added in. On the other hand, this also means that more
work is required from the modelers in the model development process, especially when a conflict resolution mechanism is needed. Although the spatial partition strategy is preferred for RePast JDSM models, the centralized environment partition and the component partition strategies are also adoptable if an additional conflict resolution mechanism is implemented. From the discussion of the experimental results, the overall overhead of RePast JDSM is less than the one of HLA RePast. The main advantage of using the DSManager and the generic approach is that the interfacing of RePast with the HLA can be realized easily. The ease of building and maintenance is apparent compared to the existing interoperating approach.

Currently RePast JDSM employs no conflict resolution mechanism for the problems of updating shared variables or objects simultaneously. Developing this conflict resolution mechanism is a priority for future development. Then the system will be compared with HLA RePast again based on a more complex simulation model, for example, the Tileworld [15]. Furthermore, since the middleware JDSManager and DSManager are generic, the approach of integrating RePast with the HLA discussed in this paper can also be applied to other agent-based simulation packages.

Acknowledgements

This research was carried out as a part of the project Crowd Simulations for Military Operations (COSMOS) supported under Defense Science and Technology Agency (DSTA) of Singapore grant POD0613456. The authors would also like to thank other research staff under the project for their valuable discussions.

References

Author Biographies

YUANXI LIANG is currently a Project Officer in the Parallel and Distributed Computing Centre at Nanyang Technological University (NTU) of Singapore. He received a Bachelor of Engineering in Computer Engineering with First-Class Honors from NTU in 2007. His current research interests include parallel and distributed simulation, multi-agent system, AI and image analysis and recognition.

WENTONG CAI is an Associate Professor with the School of Computer Engineering at Nanyang Technological University (NTU), Singapore, and head of the Computer Science Division. He received his Ph.D., in Computer Science, from University of Exeter (U.K.). He was a Post-doctoral Research Fellow at Queen's University (Canada) before joining NTU in February 1993. Dr. Cai's research interests include Parallel & Distributed Simulation, Grid & Cluster Computing, and Parallel & Distributed Programming Environments. His main areas of expertise are the design and analysis of scalable architecture, framework, and protocols to support parallel & distributed simulation, and the development of models and software tools for programming parallel/distributed systems. He has authored or co-authored over 180 journal and conference papers in the above areas. Dr. Cai is a member of the IEEE. He is currently an associate editor of ACM Transactions on Modeling and Computer Simulation (TOMACS), editorial board member of Multiagents and Grid Systems - An International Journal, and editorial board member of International Journal of Computers and Applications.

STEPHEN JOHN TURNER is Professor of Computer Science in the School of Computer Engineering at Nanyang Technological University (Singapore) and Director of the Parallel and Distributed Computing Centre. He received his MA in Mathematics and Computer Science from Cambridge University (UK) and his MSc and PhD in Computer Science from Manchester University (UK). His current research interests include: Parallel and Distributed Simulation, Grid Computing, High Performance Computing and Multi-Agent Systems.

GEORGIOS THEODOROPOULOS is a Reader in the School of Computer Science at the University of Birmingham (UK) where he has set up and leads the Distributed Systems Lab. He was also a founding Director of the Midlands e-Science Centre. He received his Diploma in Computer Engineering from the University of Patras (Greece) and his M.Sc and Ph.D. in Computer Science from the University of Manchester (UK). His research interests are in parallel and distributed simulation, distributed virtual environments, Grid computing and Peer-to-Peer systems.

ROB MINSON is a Research Fellow in the School of Computer Science at the University of Birmingham, UK. He received his M.Sc from the same institution in 2003. He currently works on large parallel simulation of Multi-Agent models as part of the MWGrid project at Birmingham. He is a member of the Distributed Systems Lab at Birmingham and has published previously in the fields of Parallel and Distributed Simulation, Multi-Agent Simulation, Interest Management in Large-Scale Virtual Environments, Massively Multi-player Computer Games and Peer-to-Peer Systems.