Interest Management in Agent-based Distributed Simulations

Lihua Wang Stephen John Turner Fang Wang
Parallel & Distributed Computing Centre, School of Computer Engineering
Nanyang Technological University
Singapore 639798
E-mail: {PG00960276, ASSJTurner, PG01538896}@ntu.edu.sg

Abstract

Distributed simulation enables participants situated in different geographical locations to share a common virtual world, which is called a Distributed Virtual Environment (DVE). Among the different research topics concerned with DVEs, there is a current trend of linking Multi-Agent systems and DVEs together. With the properties of autonomy, social ability, reactivity and pro-activeness, agents can be used to represent entities in DVEs, where fast and accurate decision making is a determining factor of the whole environment.

This paper provides a description of integrating agents into an HLA-based distributed simulation. It focuses on how to construct the sensor of an agent with different interest management schemes. Using the JADE (Java Agent DEvelopment Framework) agent toolkit and the High Level Architecture (HLA) in our prototype, a minesweeping game, we outline two different implementations of this game. Due to the dynamic characteristics of agents, a problem of overdue information from the environment is discussed, and we propose an enlarged subscription region method to solve this problem. Moreover, advisories provided by the HLA are adopted to reduce the overheads. Conclusions are drawn based on the experimental results of these implementations.

1. Introduction

Agents and multi-agent systems are one of the most prominent and attractive technologies in computer science at the beginning of this new century. An *agent* can be regarded as an encapsulated computer system that is situated in some environment and capable of flexible, autonomous action in that environment in order to meet its design objectives [1]. Agents may also communicate with each other via some form of communication language and typically have the ability to engage in cooperative problem solving.

The autonomy, social ability, reactivity and pro-activeness of agents offer great flexibility in various situations, thus agents and multi-agent systems are being used increasingly in a wide range of application areas, including information retrieval, telecommunications, business process modeling, education, military simulations, social simulations, games etc. Recently, there is a trend of using agents in distributed simulations. There are various research issues focused on this topic [2]. Most commonly, due to the limitations of current development tools and methodologies of agent systems, simulation is used to help agent system developers learn more about agents' interactive behaviors and investigate the implications of alternative architectures and coordination strategies. Also some researchers use agents to control simulations or provide advanced simulation services.

The novelty of our project is to use agents in distributed simulations, representing some of the entities in Distributed Virtual Environments (DVEs). A networked virtual environment is a distributed simulation of a virtual world in which multiple users interact with each other in real-time, even though these users (*Avatars*) may be physically located in different geographical places [3]. Using agents in the DVEs means some of the avatars can automatically update and act according to the latest information about the environment they participate in, thus no decisions from the outside world need to be made for these entities all through the simulation.

The High Level Architecture (HLA) [4] is a current U.S. Department of Defense (DoD) and IEEE standard for modeling and simulation. Some of the advantages of using the HLA as a multi-agent environment have been studied in [5]. HLA provides a standard that can reduce the cost and development time of simulation systems and increase their capabilities by facilitating the *reusability* and *interoperability* of component simulators. In the HLA, a distributed simulation is called a *federation,* and each individual simulator is referred to as a *federate*, with one point of attachment to the RTI. A federate can be a computer simulation, an instrumented physical device or a passive data viewer. The Interface Specification of the HLA describes six service classes to support federations: *federation management*, *declaration management*, *object management, ownership management, time management* and *data distribution management*.

The benefits of the HLA and the JADE (Java Agent DEvelopment Framework) agent platform were utilized in this project. JADE [6] is a software framework fully

Proceedings of the Seventh IEEE International Symposium on Distributed Simulation and Real-Time Applications (DS-RT'03)
1530-1990/03 $17.00 © 2003 **IEEE**

implemented in the Java language. It simplifies the implementation of multi-agent systems with both a middleware that complies with the Foundation for Intelligent Physical Agents (FIPA) specifications [7] and a set of tools that support debugging and deployment.

Using various interest management schemes to construct the *sensor* of an agent is the main focus of this paper. An agent uses its *sensor* to perceive the environment it is embedded in. Modeling agents within federates in the federation, the necessary data transmitted are the data useful for each sensor of a specific agent. As interest management in large scale distributed simulations is used to alleviate the network traffic by reducing or eliminating unnecessary data transferred during the simulation, it can be used to meet the demands of this kind of agent-based distributed simulation.

In the rest of this paper, section 2 gives a brief introduction to interest management. Our approach to integrate agents into the HLA simulation is illustrated in section 3. A minesweeping game as a prototype system and its different implementations using various interest management schemes are described in section 4. Section 5 gives discussions and suggestions for improvement of these implementations. Benchmarking results and comparisons are presented in section 6. Finally, section 7 concludes the paper together with future work.

2. Interest Management

Interest management (IM) is used in distributed simulations to reduce communication requirements. IM ensures the simulated entities only receive information they need during the simulation execution. To use IM, entities have to declare their preferences of specific data first, then the infrastructure will match their interests and the needed data will be finally transmitted between matching entities. Various IM schemes have been devised over the years, utilizing different communication models and filtering methods [8]. In many existing systems, IM is realized via the use of IP multicast addressing, where data is sent to a selected subnet of all potential receivers.

The IM services specified for the HLA are the latest in a succession of mechanisms for large scale distributed simulations. The HLA supports two types of filtering:

- *Class-based filtering.* This uses Declaration Management (DM) services. DM services allow a federate to update and receive updates to object attributes based solely on object class.
- *Value-based filtering.* This uses Data Distribution Management (DDM) services. DDM services extend DM services using routing spaces and regions.

The fundamental concept used in HLA to support value-based DDM is the *routing space.* A routing space is a normalised multidimensional coordinate system, where federates indicate interest in receiving or providing updates via *subscription regions* or *update regions*. By calculating the intersection of update and subscription regions, the Run-Time Infrastructure (RTI) establishes connectivity and thus provides efficient data transfer in the federation. The DMSO RTI version 1.3 statically breaks up a routing space into grid cells, and assigns a channel to each cell [4]. The update and subscription regions are mapped into the grid cells. An update is sent out on all channels corresponding to cells that overlap the update region. Entities subscribe to the channels matching the cells that overlap the subscription region. As entities change interests, they change channels correspondingly.

3. Overall Architecture

There are different approaches to constructing the overall architecture for integrating agents into an HLA simulation. A fundamental concern is to construct a feasible middleware between the agents and the RTI.

One approach is to develop object models for agent federates and for the whole federation when developing agents. In [5], a *KQML-layer* is added to every federate which hosts an agent, in order to let the agents communicate with each other successfully. Other functionality is needed to keep track of remote agents' capabilities in their architecture. The authors concluded that if the HLA/RTI were extended with an agent specific services middleware, it would provide a suitable environment for intelligent agents. In their model, non-agent federates access the RTI directly, while agent federates do so indirectly via the agent middleware.

We have advanced this proposal in an alternative way and obtained a flexible architecture in our prototype system. In our model, a *gateway federate* is developed to take charge of agents. Agent containers where agents reside are constructed upon it and the *gateway federate* can still access the RTI directly. Moreover, JADE has been selected to provide the agent-specific services, and there can be more than one agent residing in the same JADE platform. Figure 1 shows the overall architecture of our approach, the middleware shown is composed of JADE and the gateway federate.

In the simulation, the *gateway federate* is in the same JVM (Java Virtual Machine) as the agent and its platform. Using the *object-to-agent communication channel* provided by the JADE toolkit, a gateway federate can communicate with its agent by sending the sensor and effector objects alternately. This avoids the use of Java RMI which would be necessary if the JADE platform and *gateway federate are* executed in different JVMs. Details of the implementation of the agent side of this prototype can be found in our previous paper [9].

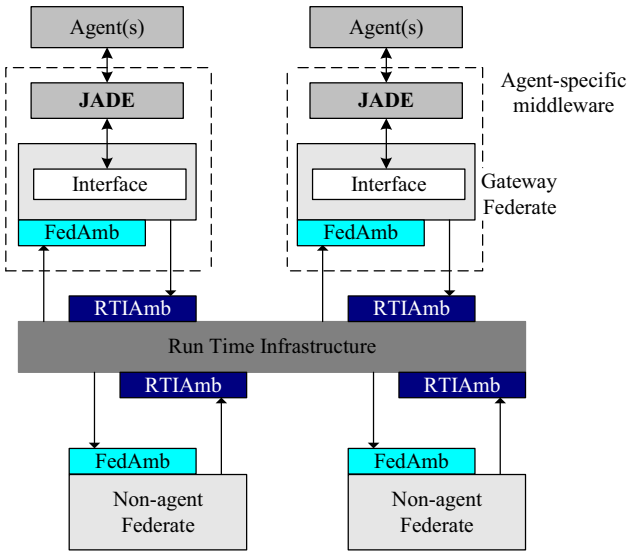

Figure 1. Middleware composed of JADE and the gateway federates

4. Prototype: a Minesweeping Game

In order to investigate and implement new ideas into the research, we examined various simulation systems used by other research groups. The systems varied from battlefield systems, air traffic control systems, stone-picking robot [10] to maze and other games, e-commerce models and simulations of natural ecological phenomena. Based on the considerations of multi-agent systems and distributed simulations, a prototype system named minesweeping game was eventually proposed. This prototype is intended to be a test bed to provide a preliminary exploration of different issues to support the distributed simulation of multi-agent systems and their environments. It is implemented using the JADE agent toolkit version 2.5 and DMSO RTI1.3NG-V6.

4.1. Description of the game

The minesweeping game contains a certain number of soldiers represented by autonomous agents and an environment shared by the soldiers. The soldiers aim to make the environment area safe by clearing all the timing mines in it before they explode. The environment has an *n*n* grid, where *n* can be set by the user. There are a number of randomly distributed obstacles (borders, trees and rivers) and mines in the environment. The obstacles are static as they exist through out the simulation and cannot be moved by soldiers. The mines are dynamic in that they can be picked up and cleared from the environment by soldiers. If a mine has not been cleared when its life expires, it will explode and the soldiers will fail their mission. The game is over when all the mines in the environment have been cleared by the soldiers.

In this game, each soldier is represented by an agent. In order to clear the mines in the environment, a soldier has to roam about the environment and detect if there is any object within its sensor range: if there are obstacles, it will step in another direction to avoid the obstacles, if there are mines, it will select a mine to pick up, add the mine to its package and then walk purposely towards the border of the environment to release the mine. After this, it will roam in the area again, trying to find another mine until all the mines have been cleared from the environment. Currently, it is presumed that when a soldier has a mine in hand, it cannot pick up another mine until it releases the mine in its hands at the border. So when a soldier detects a mine while its hands are full, it can only inform other soldiers nearby to come and pick it up.

Every soldier agent has a limited knowledge of the environment; they get the information via their own sensors, do deliberations and then act upon the environment using their effectors. This process is illustrated in Figure 2.

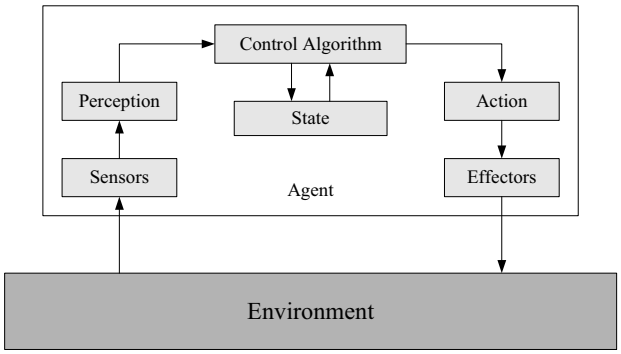

Figure 2. Soldier agent architecture of the prototype system

According to this architecture, the simulation cycle of a soldier agent in this prototype system is divided into three logical phases: *Sensing*, *Deliberation* and *Action*. In the first phase of sensing, the agent gets data from the environment, then it will decide its corresponding actions according to the rules in the deliberation phase, and finally in the third phase, the agent will perform actions.

4.2. Implementations

In order to simplify the system, two federates are initially used in this prototype: the environment federate and the soldier agent federate, representing the minesweeping environment and the soldier agent respectively. Furthermore, currently there is only one

soldier agent residing in the JADE platform. A more complex system with more agents will be investigated in the near future.

The main problem of this prototype is how to construct the sensor of an agent. What an agent can get from the environment wholly depends on the configuration of its sensors. Sensors of agents can have different ranges, which allow the agent to sense objects and other agents within a certain area centered on its current position. In this prototype, one sensor is used by an agent, and it can detect information about the nine cells centered on the agent's current cell in the environment. Figure 3 depicts the sensor range of the agent. The agent can choose to move into one of the eight cells around it if there is no obstacle around.

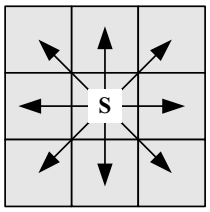

Figure 3. Sensor range

Therefore in the HLA federation, what a soldier agent federate needs is just the information of the eight grid cells around its current position instead of all the objects in the environment. This requires some mechanisms to filter out all the unnecessary information to reduce the network load and improve the efficiency of the system.

Currently, we have developed two different versions of this game. Version 1 is a simplified version, which was built at the initial stage of our research to test the feasibility of the prototype system. This version just uses the object updates and interactions provided by the RTI to avoid information broadcast of the environment, it is quite like a request and reply system. Instead, in version 2, advanced RTI services such as data distribution management (DDM) and ownership management (OM) are used to do data exchange and filtering. Currently, these two versions are both time-stepped simulations, and the agent will make one action within one simulation cycle. As a conservatively synchronized simulation, both the two federates are time-constrained and time-regulating, so their advances of logical time regulate the other and at the same time, are constrained by the other.

4.2.1. Version 1. In this version, the *Soldier* object represents the agent and has attributes for its position and the altered grid cells around it. *Mine* and *Obstacle* classes are not considered in the FOM (Federation Object Model), as the information about the mines and obstacles are passed from the environment federate to the soldier federate using interactions instead of objects. The Simulation cycle of this version is illustrated in Figure 4. TAR denotes a Time Advance Request, *t* is the time step interval and the *lookahea*d is also set to *t*. The logical time corresponding to each TAR is the time each TAR requests.

When the simulation starts up, the environment federate initializes the obstacles and mines in it, such as the position and the life time for the mines as they are set as dynamic objects. After the initialization period finishes, the environment waits for any information that comes from the soldier agent federate. When the soldier federate code is activated, it will first launch a JADE agent platform and initialize a new agent object instance. When the detailed initialization for a new agent is completed in the pure agent code, the soldier federate will resume and initialize its own federate.

In each simulation cycle, when the environment federate get the reflected attribute values of the soldier agent (step 1), it will do some work on the position to match what is around the agent in the environment (step 2), and then send out that information via an interaction (step 3). The soldier federate receives the interaction, directly uses the information to construct the sensor (step 4) and then sends both the sensor and effector to the agent using the object-to-agent communication channel provided by the JADE toolkit (step 5). After the agent gets the new sensor, it will resume and deliberate its corresponding action using its rules and make a move (step 6), this information will be written to the effector. After this move, the agent will wait again for the next sensor sent by the soldier federate. The soldier federate gets the new position of the soldier from the effector (step 7), updates this position to the environment federate (step 8) and then the next simulation cycle will begin.

The success of this version verified the feasibility of the proposed architecture. However, a drawback of this version is that there are actually two time steps in one simulation cycle, one for the soldier federate to update its new position to environment federate, the other one for the soldier federate to receive the related information via interactions from the environment federate. This can incur low efficiency of federation execution as the RTI will spend time in synchronization between federates for each time step. Moreover when more agents join the federation, this version does not scale well, because the environment federate has to calculate the match for the position of every agent, and send out the corresponding interaction to every agent.

Based on those considerations, we reinvestigated this system and developed our version 2. Ownership management and data distribution management provided by the RTI are used, which can greatly increase the efficiency of performance and reduce the network load.

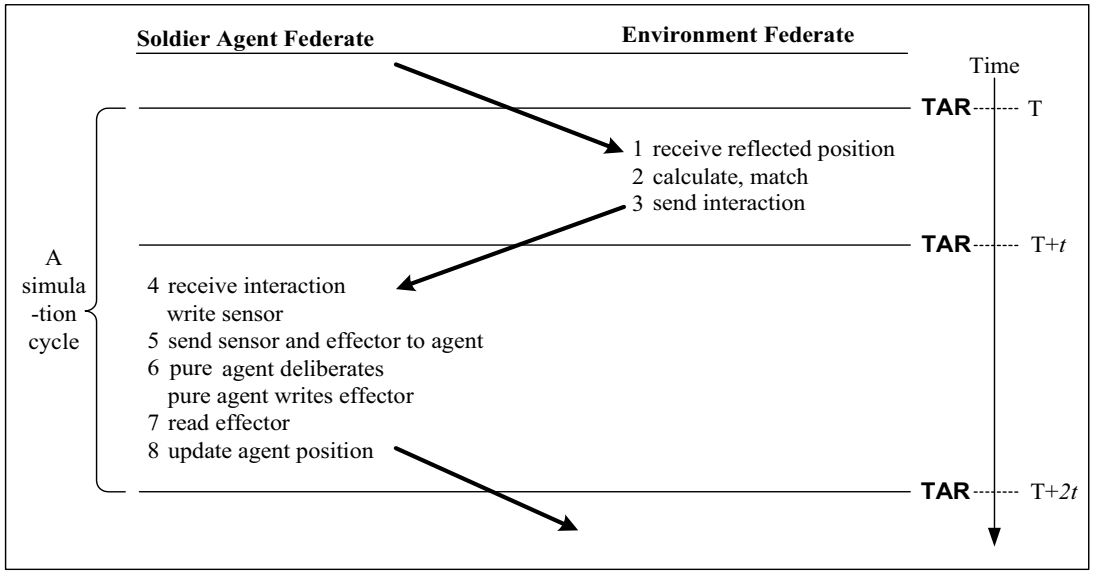

Figure 4. The simulation cycle of version 1

4.2.2. Version 2. This version treats obstacles and mines as objects in the federation, and it uses ownership transfer for management of dynamic mines. Once a mine has been picked up by a soldier agent, the ownership of that mine will be transferred from the environment to this agent, thus the agent is responsible for updating the new position of that mine, or even deleting this mine from the federation. DDM services are used to transfer information about the sensor and control the unnecessary data placed on the network. The routing space is statically partitioned into cells, with exactly the same size as the grid cells of environment. No interactions are used in this version.

Based on the previous object classes, in this FOM, *Obstacle* and *Mine* classes are added together with their own attributes. Mines are set as dynamic objects in the game, when the time expires, the mine will explode and clear itself from the environment.

Using DDM services, the environment federate will update all the objects within it for every simulation cycle, while the soldier federate will only discover[1] the objects within its current subscription region. Table 1 depicts the update regions and subscription regions for each federate in this version. The simulation cycle of version 2 is shown in Figure 5. It is a little different from the first version. Thus we describe it based on each federate.

When the simulation starts up, the environment federate initializes the obstacle objects and mine objects in it, and sets update regions and a subscription region. Using the ownership push scheme, all mine objects will also express their wish for possible ownership transfer.

[1] Here, *discover* has the general meaning of "being known or visible", instead of the term *discover* used in the HLA.

Table 1. The DDM regions of version 2

	Soldier Agent Federate	**Environment Federate**
Subscription Regions	Nine cells centered on agent's position	All cells in environment
Update Regions	Single cell containing Agent	Single cell containing Obstacle/Mine

In each simulation cycle, the environment federate will see if it receives new positions of agents (E step 1), if so, it will place these agents in its grid cells (E step 2) and unconditionally updates all the objects in the environment (E step 3). Otherwise, the federate will only update all the objects (E step 4). In both conditions, the federate will then wait for the next time step.

For the soldier agent federate, it will launch a JADE agent platform and initialize a new agent instance. When the detailed initialization for a new agent is completed in the pure agent code, the soldier federate will resume and initialize its own federate. The soldier federate gets the initial position of the soldier from the effector, sets the subscription/update regions and updates this position to the environment federate. In each simulation cycle, if there are discovered objects (S step 1), which means these objects are within the subscription region of the soldier, the soldier federate will process this information and construct a sensor (S step 2). Using the *object-to-agent communication channel*, the federate will send both sensor and effector to the agent (S step 3). When the pure agent gets the new sensor, it will resume and deliberate its

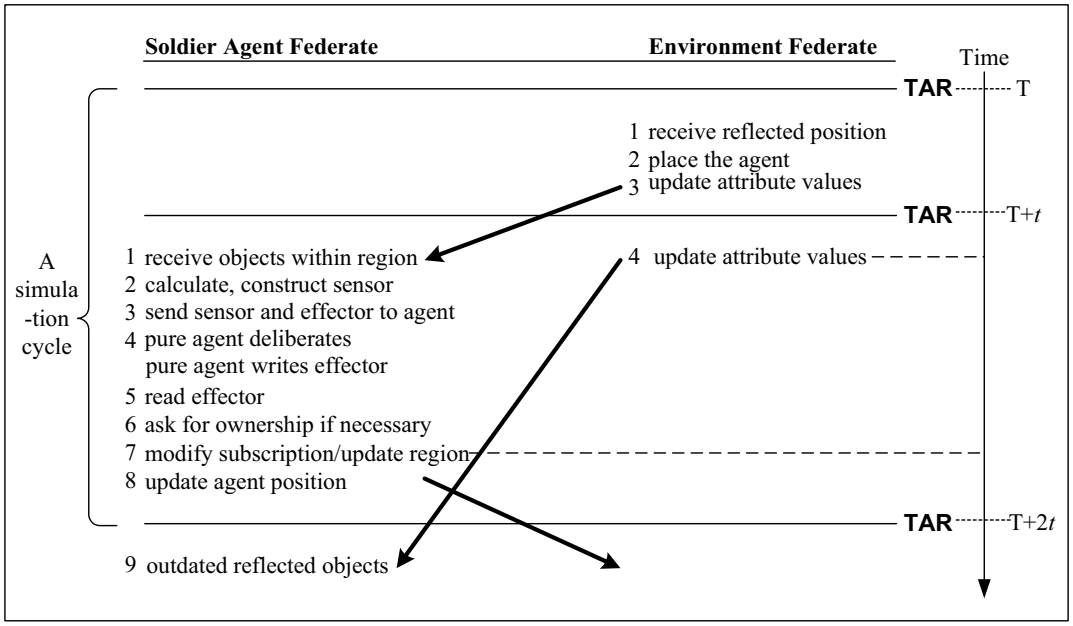

Figure 5. The simulation cycle of version 2

corresponding action using its rules and make a move (S step 4). Furthermore, if the agent decides to pick up a mine, the position of that mine will be also specified. This information will be written to the effector and read by the soldier federate (S step 5). If the federate gets the position of a mine, it will request ownership of that mine from the environment federate (S step 6). After that, the federate will immediately modify the subscription/update regions according to the new position of the soldier agent (S step 7), and update this position (S step 8). However, one problem with this version is that when the next time step begins, the federate will receive outdated objects, which will be discarded (S step 9).

Version 2 therefore still has two time steps in one simulation cycle. These problems and possible solutions will be presented in detail in the next section.

5. Dynamic Regions: Problems and Solutions

We have indicated that although version 2 uses DDM, it still has two time steps in each simulation cycle. This reduces the efficiency of the federation as the RTI will spend time in synchronization among federates for every time step. In the description of this game, every time a soldier agent stands on a specific position in the environment, it needs the sensor to do deliberation before moving to another position. Because the soldier agent can move every cycle, its fast changing subscription/update regions make the connectivity established by DDM change dynamically too. This results in the possibility that information provided by the connectivity is outdated for an agent, therefore the agent will make wrong decisions based on the stale information.

For example in version 2 (see figure 5), after deliberation, at a logical time $T+t$, the soldier decides to move from (X_0,Y_0) to position (X_1,Y_1) and immediately changes its subscription/update regions according to this new position. It is supposed to discover the objects around (X_1,Y_1) when it advances its logical time to T+2t, so that at the next time step, it can construct a new sensor based on these objects, and the agent can thus deliberate according to the new sensor and make another move. However, unfortunately when the environment federate sends out updates with timestamp T+2t, the soldier agent federate has not modified its regions yet. So the connectivity in the communication infrastructure used by the RTI was the one set up based on the subscription/update regions at logical time T. Thus the soldier agent federate will discover "old" objects around (X_0,Y_0) when it advances its time to T+2t. The desired objects will only be discovered at the next time step. So for every action the agent takes, it has to waste an extra time step in the federation to ensure the soldier agent federate will get the right information to construct its sensor and thus move in an appropriate direction.

5.1. Enlarged Subscription Region

A new algorithm is proposed to solve the dilemma about the causality problems in this prototype. This algorithm enlarges the subscription region of the soldier agent, thus when the soldier moves to any position at the next time

step, this region will still cover the desired objects within the agent's new sensor range. It is illustrated in Figure 6. The soldier agent federate only needs to locally filter the objects discovered based on its current sensor range, and discard the redundant information. So even if the DDM connectivity remains unchanged after the soldier federate's modification of its update/subscription regions, we can still make the simulation cycle conform to one time step. A successful version, version 2.1, has been implemented using this algorithm. Its simulation cycle is depicted in Figure 7.

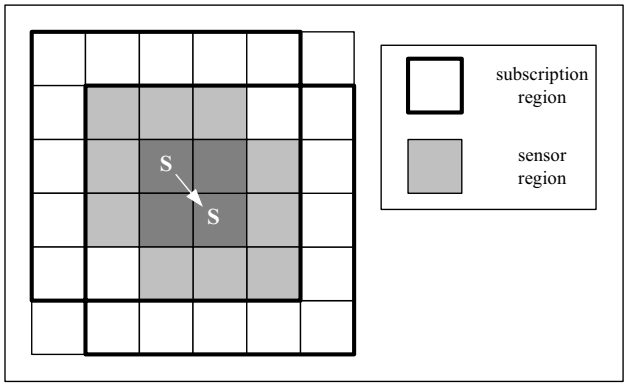

Figure 6. Enlarged subscription region of the soldier federate

5.2. Using Advisories

Although the enlarged region algorithm reduces the simulation cycle to one time step, the environment still has to blindly update all the objects in it every time step. This increases the overhead and reduces the scalability of the whole system. It seems ideal if the environment federate can update just the objects the agent needs at each time step. This can be realized using the appropriate methods and advisories provided by DDM, namely *requestClassAttributeValueUpdateWithRegion( )* and *enableAttributeScopeAdvisorySwitch( )*. After the soldier agent decides to change its position and modify the corresponding update/subscription regions, it will notify the environment federate of this change by *requestClassAttributeValueUpdateWithRegion( )*. This causes the environment federate to receive the callback *ProvideAttributeValueUpdate()* for the desired objects whose update regions overlap the soldier federate's subscription region. The environment federate will then update only these objects. As the enlarged region algorithm is already used here, the problem of outdated connectivity in the communication infrastructure is avoided. We developed version 2.2 using both the enlarged region algorithm and advisories, this version reduces overheads by sending out data only when necessary. The simulation cycle of this version is very similar to version 2.1.

6. Experimental results

Experiments were conducted to evaluate the different approaches for the minesweeping game. The platform for our experiments is three DELL 2.2GHz Pentium 4 computers connected via 100MB Ethernet running Windows XP. JADE agent toolkit version 2.5, DMSO RTI1.3NG-V6 and JAVA jdk1.4.0 are used. In our simulation test, each machine runs a federate. To ensure better results, one computer is used to run the *rtiexec* and *fedexec* separately and all unnecessary outputs were deleted from the source code. The execution time for the

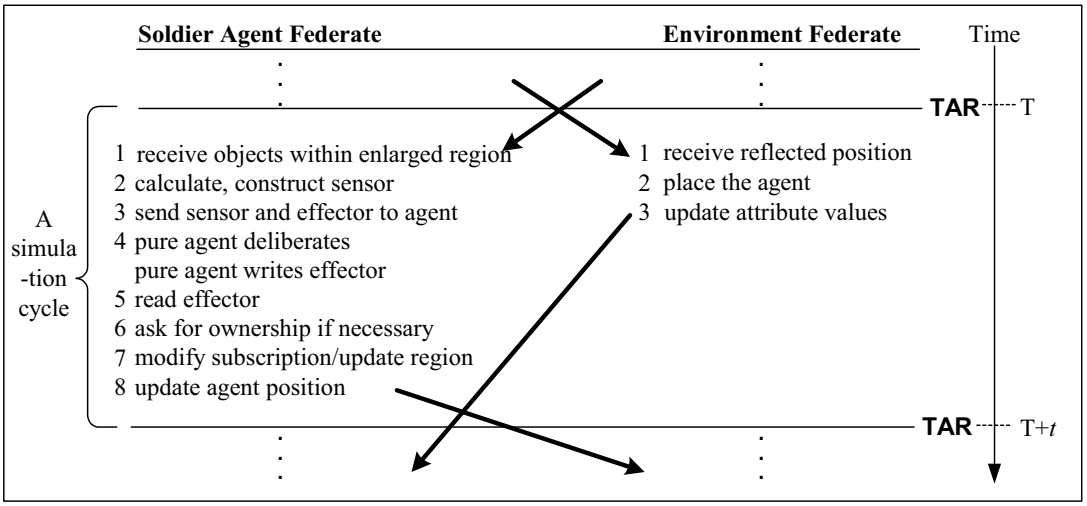

Figure 7. The simulation cycle of version 2.1

agent federate to run 5000 simulation cycles using the four different implementations is recorded. The execution results of the different versions are shown in Figure 8.

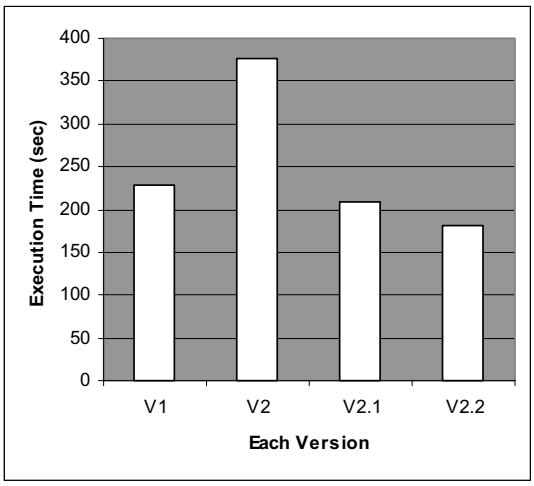

Figure 8. Execution time of different versions

Comparison of the simulation results shows that the proposed enlarged subscription region algorithm, although it increases the network load slightly and introduces some latency in processing the unnecessary information, is still more efficient when the execution time is concerned. Furthermore, the results show version 2.2, which uses both the enlarged subscription region algorithm and the advisories, not only reduces the overhead, but also decreases the federate execution time. Thus it is the most efficient implementation.

7. Conclusions and Future Work

This paper has discussed how to integrate JADE agents into an HLA-based distributed simulation with the focus on interest management. A minesweeping game has been set up to verify the feasibility of combining these two technologies together, and different implementations of this prototype have been discussed. As agents are dynamic in the environment, we also proposed an algorithm for this game to circumvent the problem resulting from the synchronization of services in logical time. Moreover, advisories provided by DDM are used to improve the performance of these implementations. The experimental results show that our efforts to reduce the simulation execution time are successful.

There are other solutions to the problem mentioned in section 5, for example modifying the RTI to maintain a log of messages as discussed in [11]. As our research goes on, a more general algorithm to synchronize services between agents and the environment will be required. This may be realized using time stamps for some currently unsynchronized RTI services and/or exploring a middleware between the RTI and federates. Furthermore, when more than one agent participates in the environment, the game scenario will be more complicated. This will result in numerous issues both in agent technology and distributed simulation. Suppose two agents try to step to the same place without knowing the existence of each other in their previous sensor, or two agents try to pick up the same mine. Mutual exclusion mechanisms are required to solve these kinds of problem. Specifically, as agents are collaborative in nature, they are different from previous work in DVEs where entities seldom have concurrent interactions. Concurrent interactions [12] are everywhere in the real world. If we want to emulate the real world more precisely, managing concurrent interactions in DVEs is a demanding need.

References

[1] N.R. Jennings, "On agent-based software engineering", *Artificial Intelligence* 117, Elsevier, 2000, pp. 277-296.

[2] A.M. Uhrmacher, P.A. Fishwick and B.P. Zeigler (eds), "Special issue on agents in modeling and simulation: exploiting the metaphor", *Proceedings of the IEEE*, 89 (2), February 2001.

[3] R.M. Fujimoto, *Parallel and Distributed Simulation Systems*, Wiley Interscience, 2000.

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

[5] J. Andersson and S. Löf. "HLA as Conceptual Basis for a Multi-Agent Environment", *Technical Report 8TH-CGF-033*, Pitch Kunskapsutveckling AB, 1999.

[6] F. Bellifemine, A. Poggi, and G. Rimassa, "JADE - A FIPA-compliant agent framework", *Proceedings of PAAM'99*, April 1999, pp. 97-108.

[7] FIPA Agent Management Specification. *Technical Report SC00023J*, http://www.fipa.org/, December 2002.

[8] K. L. Morse, "Interest management in large scale distributed simulations", *Tech. Rep. 96-27*, Department of Information and Computer Science, University of California, Irvine, 1996.

[9] F. Wang, S.J. Turner and L. Wang, "Integrating Agents into HLA-Based Distributed Virtual Environments", *Proceedings of the 4th Workshop on Agent-Based Simulation*, Montpellier, France, April 2003, pp. 9-14.

[10] F. Chantemargue and B. Hirsbrunner, "A collective robotics application based on emergence and self-organization", *Technical report*, PAI group, Fribourg, Switzerland, 1999.

[11] I. Tacic and R.M. Fujimoto, "Synchronized Data Distribution Management in Distributed Simulations", *Proceedings of the 12th Workshop on Parallel and Distributed Simulation* (PADS '98), May 1998, pp. 108-115.

[12] A. Natrajan, P.F. Reynolds, Jr. "Resolving Concurrent Interactions", *Proceedings of 3rd International Workshop on*

Distributed Interactive Simulation and Real Time Applications,
October 1999, pp. 85-92.

Proceedings of the Seventh IEEE International Symposium on Distributed Simulation and Real-Time Applications (DS-RT'03)
1530-1990/03 $17.00 © 2003 **IEEE**