Proceedings of the 2004 Winter Simulation Conference
R .G. Ingalls, M. D. Rossetti, J. S. Smith, and B. A. Peters, eds.

RESOLVING MUTUALLY EXCLUSIVE INTERACTIONS IN AGENT BASED DISTRIBUTED SIMULATIONS

Lihua Wang
Stephen John Turner
Fang Wang

School of Computer Engineering
Nanyang Technological University
639798 SINGAPORE

ABSTRACT

With the properties of autonomy, social ability, reactivity and pro-activeness, agents can be used to represent entities in distributed simulations, where fast and accurate decision making is a determining factor of the whole environment. Resolving concurrent interactions is a key problem of this kind of system, as the shared environment needs to allow agents to interact with the environment in a causally consistent way. There will usually be either mutually exclusive or collaborative interactions. This paper presents our research in designing a middleware component called Interaction Resolver (IR) to resolve the effect of concurrent interactions and still guarantee the consistency and causality of the system. The ownership management services provided by the High Level Architecture (HLA) are compared with IRs in resolving mutually exclusive interactions in our prototype, a minesweeping game. Conclusions are drawn based on the experimental results.

1 INTRODUCTION

Recently, there is a trend of using agents in distributed simulations (Uhrmacher, Fishwick, and Zeigler 2001). An *agent* can be regarded as an encapsulated computer system that is situated in some environment and is capable of flexible, autonomous action in that environment in order to meet its design objectives (Jennings 2000). Agents can communicate with each other via some form of communication language and they 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. The novelty of our project is to use agents to represent some of the entities in distributed simulations.

A distributed simulation is executed on a computing system with multiple possibly geographically distributed processors interconnected via a communication network (Fujimoto 2000). Using agents in distributed simulations means some entities in the simulation can automatically update and act according to the latest information about the environment in which they participate, thus no decisions from the outside world need to be made for these entities.

The High Level Architecture (HLA) (DMSO 1998) 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 (Andersson and Löf 1999). 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 Run-Time Infrastructure (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 (Bellifemine, Poggi, and Rimassa 1999) is a software framework fully 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 (FIPA 2002) and a set of tools that support debugging and deployment. There are different approaches to constructing the overall architecture for integrating agents into an HLA simulation. In our model (Wang, Turner, and Wang 2003), a middleware is composed of JADE and a gateway federate. The gateway federate is developed to take charge of

agents. JADE agent containers where agents reside are constructed upon it and the gateway federate can still access the RTI directly. There can be more than one agent residing in the same JADE container. Every 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*. Using the object-to-agent (O2A) communication channel provided by the JADE toolkit, a gateway federate can communicate with its agents using the sensor and effector objects.

This paper focuses on how to resolve concurrent interactions, especially mutually exclusive interactions, in agent based distributed simulations. When more than one agent joins the simulation, management of the shared environment is a key problem: the shared environment needs to be structured to allow agents to interact with the environment in a causally consistent way. There will usually be either mutually exclusive or collaborative interactions in agent based distributed simulations. To resolve the effect of these interactions and still guarantee the consistency and causality of the system, we developed a middleware component called Interaction Resolver (IR).

In the rest of this paper, section 2 gives a brief introduction to concurrent interactions. Two solutions to the problems of mutually exclusive interactions in our prototype system are illustrated in section 3. The two solutions are ownership management (OM) services provided by the RTI and our Interaction Resolver (IR). OM and IR are compared in depth in section 4 based on tests of two different mutually exclusive scenarios in our prototype. Benchmarking results and a summary are also presented in this section. Finally, section 5 concludes the paper together with future work.

2 CONCURRENT INTERACTIONS

An *interaction* can be regarded as the way entities in a system communicate with or influence one another. The results of these interactions normally change behaviors of these entities. In distributed simulations, concurrent interactions can be defined as interactions that happen at the same simulation time or during the same time step. As agents have the properties of autonomy, social ability, reactivity and pro-activeness, which offer great convenience in various situations, it is necessary to support concurrent interactions in agent systems where agents can collaborate to achieve their goals.

Concurrent interactions can be either mutually exclusive or collaborative. It depends largely on the intentions of these interactions. Mutually exclusive interactions are interactions that try to access a shared object concurrently, while the object only allows a single operation on it at one time. On the other hand, there are situations where a shared object permits concurrent accesses to be combined in order to change properties of the object. These interactions are collaborative interactions. A famous example is that if two different persons lift a table one after another, they are not able to lift it up individually. The table can only be lifted when both of them collaborate at the same time as shown in Figure 1.

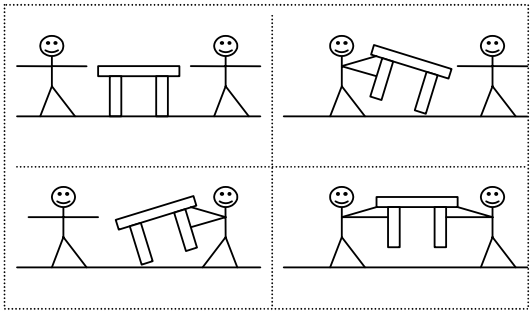

Figure 1: Collaborative Concurrent Interactions

According to (Broll 1995), distributed virtual environments providing concurrent interactions have to deal with two different kinds of problems: the detection of those concurrent interaction requests and a "good" mechanism to resolve these requests. Unresolved concurrent interactions may lead to unexpected errors of the whole system. Beside other specialized methods, Broll mentions four possible alternatives: *priority based interaction request resolving*, *request time dependent interaction sequencing*, *constraint based interaction request resolving* and *combining interaction request*.

3 RESOLVING MUTUALLY EXCLUSIVE INTERACTIONS

A prototype system named a minesweeping game was developed for our project to investigate mutually exclusive interactions. The game is implemented using the JADE agent toolkit version 2.5 and DMSO RTI1.3NG-V6. Figure 2 gives a snapshot of this game.

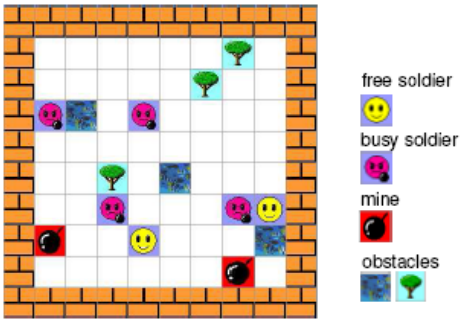

Figure 2: Snapshot of the Minesweeping Game

In this game, soldiers are roaming in a minefield to find and clear mines to make the environment safe. The environment has an *n*n* grid. It consists of static obstacles for soldiers to avoid and dynamic mines for soldiers to pick

up. Mines are dynamic as they can explode within a certain time if they are not picked up and cleared from the environment by soldiers. Each soldier is represented by an autonomous agent. It can only have one mine in hand, so if a soldier already has a mine in its hand (a busy soldier), it has to walk purposely toward the border of the minefield to release it. After that, it will roam in the environment again as a free soldier to find a mine.

The sensor region of each agent is just the eight grid cells around its current position. We use Data Distribution Management (DDM) services provided by the RTI to construct the agent sensors (Wang et al. 2003). Each agent federate's subscription region (in DDM, each federate expresses its interest in receiving data via subscription regions) is mainly based on the sensor regions of agents in it.

3.1 Problems of Mutually Exclusive Interactions

As far as our prototype is concerned, there are both mutually exclusive interactions and collaborative interactions. With more than one agent participant in the game, collision problems will occur. This is mainly because of the limited sensor region of each agent. Figure 3 illustrates this problem in our prototype.

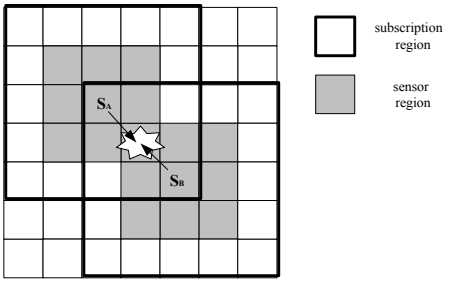

Figure 3: An Example of Collision

In this figure, shadowed areas are the sensor regions for agents. It is clear that agent SA and agent SB may step to the same empty grid cell without awareness of the existence of each other. Another example of mutually exclusive interactions in our present system is the problem of picking up the same mine. It is almost the same as the example above.

It may be thought that enlarged subscription regions (we use this method for DDM services as they are not time stamped) may be one solution, as it allows each agent to get sufficient information of agents around it as shown in Figure 3. However, the intentions of other agents, their future actions (stepping to the empty cell or not, picking up the mine or not) are unknown at that time step. When the next time step begins, the collision may have already happened. Advanced methods have to be adopted to solve these problems. The ownership management services provided by the RTI can be one solution, but this approach has its limitations. A new method using Interaction Resolvers has been developed for agent based distributed simulations. Interaction Resolvers can also be used to solve collaborative interactions, which is part of our future work.

3.2 Solution 1: Ownership Management (OM)

Ownership management is used by federates and the RTI to transfer ownership of instance attributes among federates (DMSO 1998). It is possible for an object instance to be wholly owned by a single federate. But only one federate can have update responsibility for an individual attribute of an object instance at any given time. This mutually exclusive property of ownership transfer can be used to solve the problems of mutually exclusive interactions.

For example, we can declare mines as HLA objects. Suppose there are two agents that want to pick up the same mine, they must first request the ownership of the mine from the original owner federate of the mine. If one of them must obtain the ownership of that mine instance before taking any action, the collision of concurrent picking up will not occur: when one agent gets the ownership of the mine instance, the other one will be notified of its failure in competition for the ownership. So when the next time step begins, only the winner picks up the mine. The loser will adopt other actions: the surrounding information for it has already changed as the mine is cleared from its original position, so the loser will not try to pick up the mine.

As the HLA interface specification does not relate time stamps to ownership transfer, it is possible to have unexpected situations happen. For example, two agent federates may both request the ownership of one object attribute within the same time step, but due to the time delay and other factors, it is possible for the later one to get ownership because the RTI receives it earlier. This receive-order (RO) based priority in getting the ownership can be inconvenient in agent based simulations, where users may prefer to specify that certain agents with defined priority or attributes should win in a competition. Also, if two federates try to acquire and then release ownership of the same object instance within the same time step, it is highly possible that one of them may get the ownership just after the other has released it. Thus the ownership can be acquired twice in one time step, which is illogical.

In the RTI, it makes no sense for each federate to acquire ownership of specific set of instance-attributes more than once. So when there are more than one agent in one agent federate, these agents will have to compete with each other mainly based on the time they submit their request. Who will obtain the ownership for this federate is unpredictable, as only one agent will "represent" this federate. This is unfair if we want to realize more functions in the competition.

Based on all these considerations, a mechanism using a middleware interaction manager for each federate called *Interaction Resolver* is investigated in our project.

3.3 Solution 2: Interaction Resolvers (IR)

Interaction Resolvers are deliberately designed to resolve concurrent interactions in our prototype and similar distributed simulations. Each federate will have a local IR,

which is used to work out partial collisions that happen within this federate's subscription region. All the IRs will follow the same working mechanism, thus the result reached in each federate is the same for every shared object. In our game, for each time step, as each agent that resides in the federate only needs to know the objects within its sensor region to make a decision, as long as this information is consistent, accurate and up-to-date, there will be no errors for the decisions of agents in the whole simulation. IRs are distributed across the whole simulation, this greatly reduces the network load compared to the centralized ownership management method. Figure 4 shows the overall architecture of the agent based simulation with IRs.

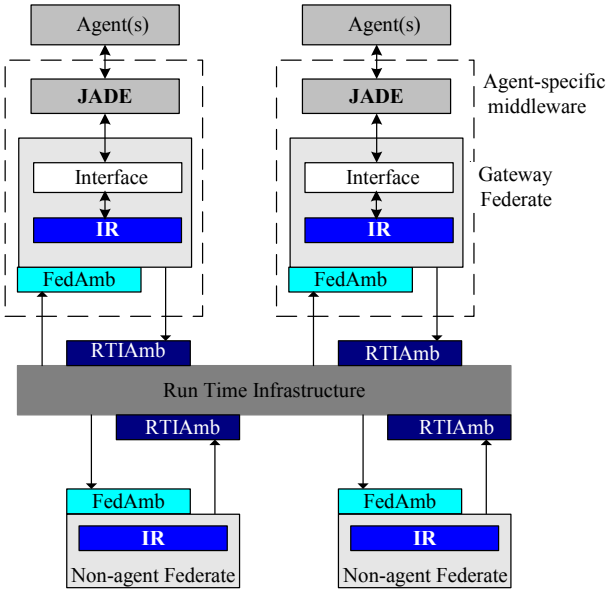

Figure 4: The Middleware with Interaction Resolvers

In each time step, all interactions (*interaction* has the general meaning instead of the *interaction* used in the HLA) within a federate's subscription region will be received and stored in a list in its local IR. Then the IR will group them according to the objects and objects' properties of these interactions. For example, in our system, within the interactions received within one time step, some of them may be concerned with stepping to cells, while others may be concerned with picking up mines. The objects of these interactions here are the empty grid cells and the mines. How to determine if these interactions have collisions depends on the definition of the system, or the properties of these objects. In our game, if agents step to grid cells with the same position *(x,y)*, these interactions are considered as having a collision. Likewise, interactions of picking up mines with the same position *(x,y)* are also treated as having a collision.

Concurrent interactions will be put in different collision lists and then sorted. The sorting of these interactions is mainly based on how the collision is defined. For mutually exclusive interactions, each local IR will compare interactions in each collision list according to the agents' priorities to decide who can be the winner in the collision. If two interactions have the same priority, further properties of each interaction will be used for the decision. These properties must have unique values throughout the whole distributed simulation, thus it can quickly bring an end to the winner deciding procedure. We use object instance IDs in our prototype, as these IDs are assigned to each object instance by the RTI, and are unique throughout the federation. So even if two agents have the same priority, they definitely have different instance IDs, and we can then select the winner.

Only the winner can have exclusive privilege to access the shared object, while all losers in this list will require correction. In our game, we roll back losing stepping agents to their previous positions. These corrections are adopted because all agents assume that they have already accessed the shared object and then their owner federate updates this information to other federates. That is why collisions are detected in each relevant IR in the beginning of the next time step. Because IRs are used before constructing sensors that are sent to corresponding agents, as long as all IRs use the same method of correction, and the results of the correction are consistent, the whole system is free of mistakes as each agent will get corrected status of itself and updated nearby information for its next action. IRs will not function if there are no collisions in one time step. Figure 5 illustrates the working of IRs in our system.

```
for each collisionList L[]
    MaxPriority = 0;
    i = 0;
    while i <= L[].length do
        if L[MaxPriority]. priority < L[i]. priority then
            rollback (L[MaxPriority])
            MaxPriority = i
        else if L[MaxPriority]. priority > L[i]. priority then
            rollback (L[i])
        else if L[MaxPriority]. priority = = L[i]. priority then
                if L[MaxPriority].mInstanceID > L[i]. mInstanceID then
                        rollback (L[i])
                else
                        rollback (L [MaxPriority]);
                        MaxPriority = i
                end if
        end if
        i++
    end while
    winner = L [MaxPriority];
    accept (winner)
end for
```

Figure 5: Algorithm of IRs in the Prototype System

Admittedly, using IRs does bring temporary inconsistency across a time step in the whole federation, but as far as all the "pure" agents are concerned, in each time step, all IRs immediately correct this inconsistency, and agents will get corrected information and make right decisions. There is no inconsistency in the information received by agents. So in this sense, the whole system is still consistent. Figure 6 shows the keeping of consistency in our game. Here TAR means Time Advance Request. In this example, we

can clearly see that for federate 1 and federate 2, there is inconsistency of data among different federates at the same simulation time: each federate may not have the right agent positions for agents of other federates within its subscription region. But as far as all the pure agents are concerned, before they make deliberations in every time step, the use of IRs ensures that all relevant data is consistent.

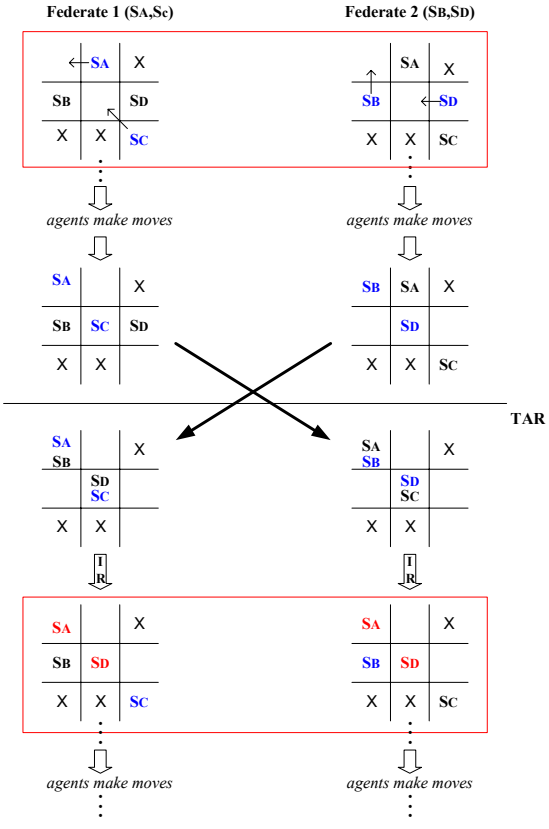

Figure 6: How Consistency is Ensured Using IRs

4 COMPARISON OF SOLUTIONS

In order to compare the efficiency of IR with OM, two different test environments are set up. In the first group, all agents are trying to step into the same grid cell in a 5*5 minesweeper environment, while in the second group, all agents are picking up the same mine. Figure 7 shows the test environments.

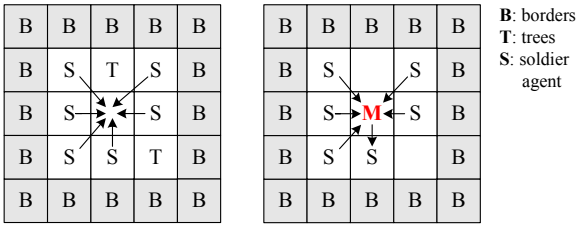

Figure 7: Test Environments

4.1 Stepping to the Same Grid Cell

In this test, all agents in the environment need to step to an empty cell. We add some obstacles in order to increase the number of agents trying to step to the same grid cell. Once the winner gets the right to step to the empty cell in the middle, it will step out the next time step, and all losers will stand still at that time. This is to make the whole test cycle repeat and give the maximum number of agents in competitions for cells.

In order to resolve the conflict, both OM and IR can be used. A synchronization point is used to start the simulation at exactly the same time, otherwise agents in early joined federates will move around before all agents have joined, which will influence the test results.

4.1.1 Using OM

The specific grid cell in the middle of the minefield is declared as an HLA object in the federation; once the environment federate is initialized, it automatically has the ownership of that grid cell. All agents interested in stepping to this grid cell should first request its ownership, and only when the federate in which the agent resides gets the ownership, can the agent step to the cell. The winner agent will then step out of the cell, and release the ownership of the grid cell for the next possible ownership competition. Agents failing to step to the grid cell will be notified by the RTI and will stand still until there is another new empty grid cell within their sensor region. Figure 8 illustrates the test cycle of this process.

There are two time steps in one test cycle. We have two main types of agent federates: the winner federate (the agent federate who gets ownership of the cell) and the loser federates (other agent federates who are denied ownership). The test cycles for both types are almost the same. They mainly differ in the agent actions: the winner agent will step to and step out of the grid cell, while all loser agents will stand still at their original positions. It is important to note that loser agents can reside in a winner federate.

A test cycle of a winner federate is described as follows. When a test cycles starts, the federate will receive soldier updates within its subscription region (S step 1). It will use this information to construct the sensors for each agent in it (S step 2). The federate will then send both sensors and effectors to respective agents using the *O2A communication channel* (S step 3). When the pure agent gets its new sensor, it will deliberate its action and decide to step to an empty cell within its sensor region (S step 4). The federate will get the decision from the effectors (S step 5), and request ownership of the empty grid cell (S step 6). In order to ensure subsequent ownership transfer, the agent federate will *tick* and wait for ownership callbacks from the RTI (S step 7). This ensures every federate in the competition know the result of their ownership requests. The

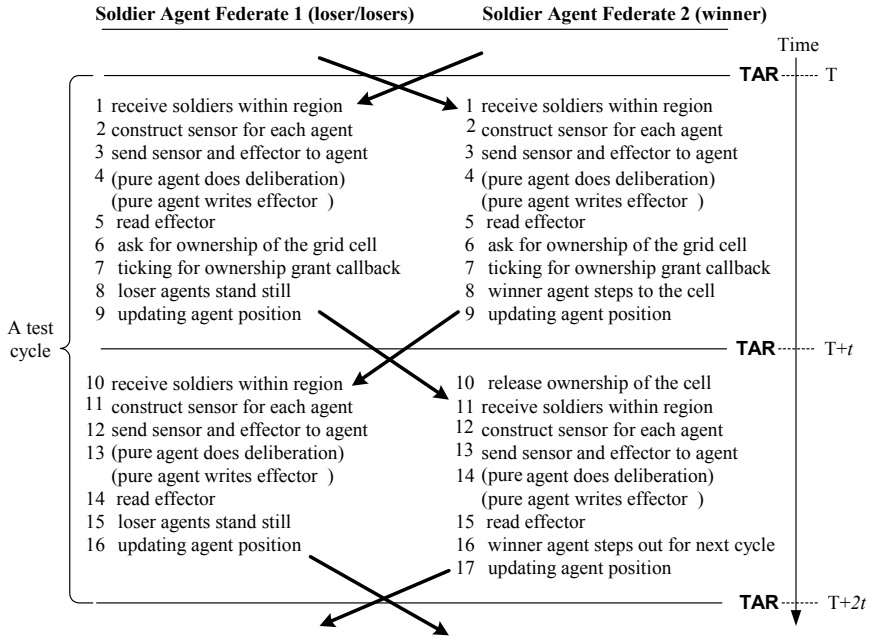

Figure 8: The Test Cycle of Using OM in Stepping to the Same Grid Cell

winner agent will step to the cell (S step 8). After this, the agent federate will update agent positions within it (S step 9). In the next time step, the winner federate will first release the ownership of the grid cell instance (S step 10). S step 11 ~ S step 15 are the same as S step 1 ~ S step 5. After the federate has read the effectors, for the winner agent, it will step out of the grid cell (S step 16). Then the federate will update agent positions again (S step 17).

4.1.2 Using IR

Using IRs, each federate can locally decide who can step to the cell due to the rules of the game. In this test, the one with the highest priority has the privilege. If two agents have the same priority, their unique instance ID in this federation will help in further decision. All losers will be rolled back by the local IR. The same information in each IR ensures the consistency of the view of each agent.

All agents will step to the empty grid cell if there is one within their sensor regions, and only the winner will stay in that cell, while all losers will be rolled back. This is one of the main differences from the method of using OM.

Also, as grid cells need not be HLA objects when using IR, there is no ownership associated with each grid cell. Thus there is no need for agent federates to request its ownership before they step to the cell, and release ownership when they step out.

As with the OM services method, this method also has two time steps in one test cycle. The test cycle of this method is illustrated in Figure 9. The first time steps are the same for both types of agent federates. They all try to step to the empty grid cell. The federate will first get all the soldier information within its subscription region (S step 1). Then it will construct sensors (S step 2) according to the information and send both sensors and effectors to respective agents using the *O2A communication channel* (S step 3). When the pure agents get the new sensors, they will deliberate their actions and write effectors (S step 4). The federate gets each agent's action information from its effector (S step 5), and they know that agents are stepping to the empty grid cell (S step 6). This information is updated by these federates (S step 7). When the next time step begins, each federate will get relevant updated soldier information (S step 8). All the agents with the same position will be grouped and the local IR will decide who can be the winner (S step 9). Only the winner agent can stay in the position (S step 10), while all losers will be rolled back to their old positions. Because the positions of soldier agents change after IR is used, sensors and effectors for every agent need to be refreshed to notify the pure agents of their current positions (S step 11). S step 12 ~ S step 15 are the same as S step 2 ~ S step 5. For the winner agent, it will then step back to its original place for the next test cycle (S step 16). Other loser agents will stand still as they have already been rolled back by IRs. At this stage, the environment will become exactly the same as in the initialization, that is, all agents stand centered around an empty grid cell. The federate will update agents' information before it advances the simulation time (S step 17). Then the next cycle begins.

4.1.3 Experimental Results

Timings for different numbers of agents/federates are carried out. The platform for our experiments is six DELL 2.2GHz Pentium 4 computers connected via 100MB Ethernet running Windows XP. JADE agent toolkit version

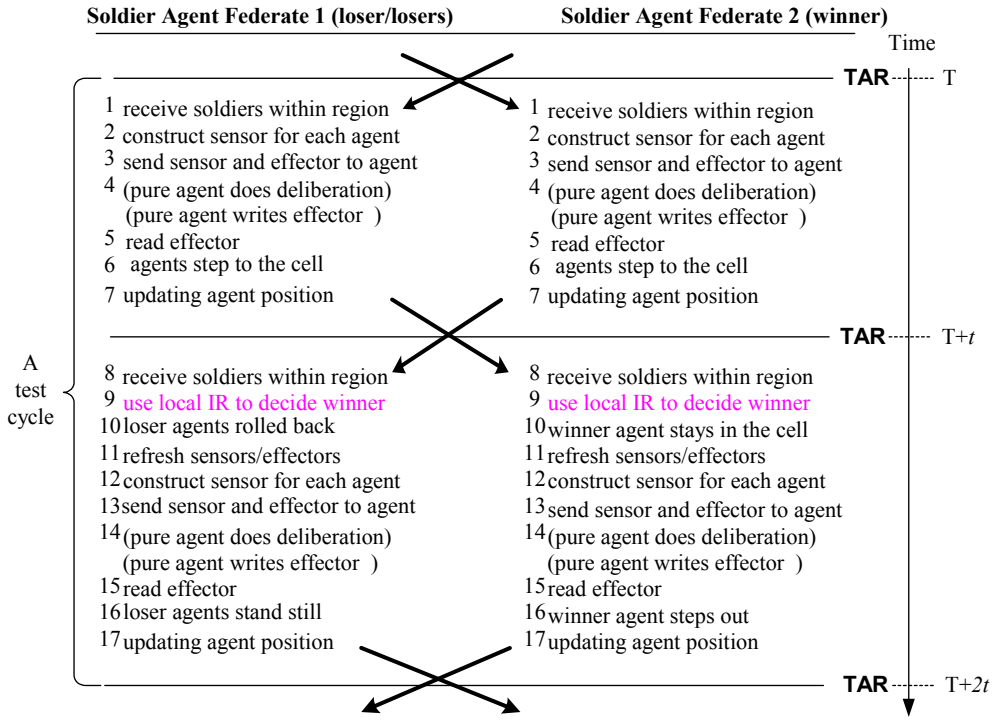

Figure 9: The Test Cycle of Using IR in Stepping to the Same Grid Cell

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 from the federates and all unnecessary outputs were deleted from the source code. The execution times for the agent federates to run 2000 test cycles (4000 time steps in this test) using both OM and IR are recorded. For each method, the test is divided into ten different groups, varying in agent and federate numbers. The distribution of agents in varying numbers of federates (see Table 1) is carefully designed to get even load balancing and thus achieve the best result.

Table 1: Distribution of Agents in Federates

Number of Federates	Total Number of Agents	Distribution
2	2	1+1
	4	2+2
	6	3+3
	8	4+4
3	4	1+1+2
	6	2+2+2
	8	3+3+2
4	4	1+1+1+1
	6	1+1+2+2
	8	2+2+2+2

Figure 10 shows the complete results according to the number of federates in the federation. It is clear that using IR is more efficient than using OM for this test.

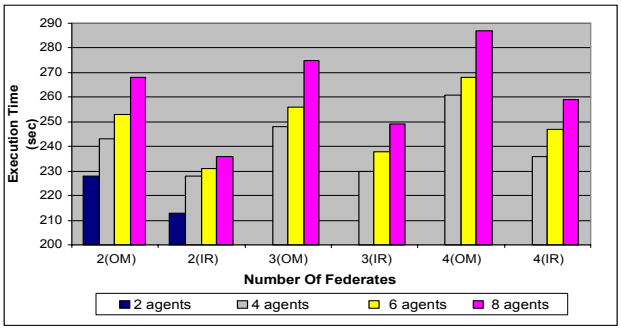

Figure 10: Execution Time of OM and IR in Test One

4.2 Picking up the Same Mine

In this test, soldiers are deliberately set as static agents, that is, they stand still in their initial positions and their only action is to pick up a mine within their sensor regions. Once there is a mine in the minefield, all agents will try to pick it up. The federate that gets the ownership will then delete the mine from the federation instead of releasing the ownership as in the previous test. The environment federate will keep on generating mines in the middle once the old mine is picked up and deleted from the federation.

4.2.1 Using OM

The mine is declared as an HLA object in the federation. Once a mine object is initialized, the environment federate automatically has the ownership of it. All agents interested

in picking up this mine should first request ownership of the mine, and only when the federate the agent resides in gets the ownership can the agent pick that mine up, and delete it from the federation. Agents failing to pick up the mine will be notified by the RTI and will wait until there is another new mine within their sensor regions.

This method uses three time steps in one test cycle. This is mainly because the Environment federate needs to update the mine object to other federates, and the discovery of the mine object needs one time step. The test cycle is almost the same as the previous one except for above differences.

4.2.2 Using IR

IRs in each federate will resolve the collision of picking up the same mine. Only the winner of the collision will have the chance to acquire the ownership of the mine, and it will get the ownership without ownership competition in the RTI. This is different from using IR in stepping to a grid cell where no ownership transfer is involved.

For each federate, its local IR will group agents with the same purpose-pick up the mine, then the IR will determine which agent wins in the competition due to its priority. All losers in the IR will be notified of their failure in picking up the mine. So in this test, ownership transfer is still used, but in a less competitive way than in using OM to resolve mine picking conflicts. Some extra time spent in ownership transfer is inevitable here. There are also three time steps in each test cycle. The cycle is similar to the one used in competition for empty cells.

4.2.3 Experimental Results

The platform for this test is exactly the same as the previous one. The execution times for the agent federates to run 2000 test cycles (6000 time steps in this test) using both OM and IR are recorded. This test uses the same distribution of agents in federates as in Table 1. Figure 11 shows the complete results according to the number of federates in the federation. These results demonstrate that in this test, although extra time is spent in ownership transfer when IR is used, IR is still more efficient than OM.

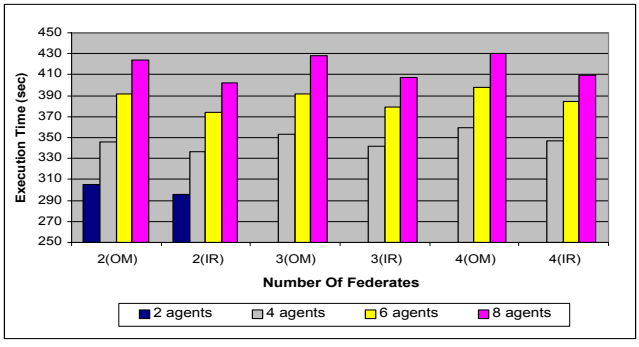

Figure 11: Execution Time of OM and IR of Test Two

4.3 Summary

From the two tests, we can see clearly that the IR method is more efficient than the OM method.

Due to the limitation of the environment in our game, there are at most eight agents in a collision. In a more general distributed simulation, where there are many more agents in a collision, the original OM method may not scale well. With the increase of the requests from federates for the same set of instance-attributes, the network load will be greatly increased due to the centralized server. The IR method solves the conflict locally instead. Moreover, there is no bottleneck in the system as all IRs are distributed across the simulation. All these factors reduce the bandwidth requirement in the simulation. This method also avoids using *tick()* to wait for ownership callbacks from the RTI. Suppose there are many federates in the federation, all ticking for the callbacks, it brings a large synchronization burden for the whole federation.

Moreover, there is a disadvantage of the OM method that all losers in the competition will not know who wins. The way to select the winner is decided by the RTI due to the request time and is unpredictable. This is very inconvenient for agent based simulations where we may want a specific agent with certain advantages to win over others. The IR method enables users to design different winning mechanisms for various implementations. In addition, all the losers in the competition will know the winner, and this is an advantage for practical implementations where further decisions may be based on this knowledge.

We have mentioned that the OM method allows only one agent in each agent federate to request ownership. The IR method definitely does not have this trouble as all agents within each IR are the same regardless of whether or not they belong to the same federate. It ensures free-competition throughout the federation. Also the rules for the competition can be freely modified by the users.

When we change to an event-based distributed simulation, resolving concurrent interactions is quite different, as there will not be any time-step in the simulation. For the OM method, it is almost impossible as the services are not time stamped. Additional work is needed if OM is to be used in event-based distributed simulations. However, for IR we may just need to set a period of time acting as some kind of time threshold.

Based on all these comparisons, we can safely draw the conclusion that the IR method is better than the OM method for solving mutually exclusive interactions in agent based distributed simulations.

5 CONCLUSIONS AND FUTURE WORK

This paper shows how to resolve concurrent, mutually exclusive interactions in agent based distributed simulations. Using our prototype system, a minesweeping game, own-

ership management services are compared with interaction resolvers. Two different situations of mutually exclusive interactions: stepping to the same grid cell and picking up the same mine are investigated using these two solutions. Besides the flexibility and convenience IR provides, experimental results also show that the distributed IR saves more execution time than the centralized OM.

Current IRs in our system can only deal with mutually exclusive interactions. In the near future, a more complete IR component needs to be devised to satisfy the requirements of collaborative agents. We need to set up certain rules for local IRs to divide different interactions according to certain properties, and act upon each of them to achieve a combined interaction. This is not difficult based on what we have achieved so far. Moreover, the current IR algorithm needs to be generalized, that is, IRs can analyze the common features of interactions and divide them into different categories (Natrajan and Reynolds 1999).

REFERENCES

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

Bellifemine, F., A. Poggi and G. Rimassa. 1999. JADE - A FIPA-compliant agent framework. In *Proceedings of 4th International Conference on Practical Applications of Agents and Multi-Agent Systems*, 97-108.

Broll, W. 1995. Interacting in Distributed Collaborative Virtual Environments. In *Proceedings of Virtual Reality Annual International Symposium*, 148-155.

DMSO. 1998. High Level Architecture RTI Interface Specification, Version 1.3, Defense Modeling and Simulation Office.

FIPA. 2002. Agent Management Specification Technical Report SC00023J, Foundation for Intelligent Physical Agents. Available online via <www.fipa.org/> [accessed April 12, 2004].

Fujimoto, R.M. 2000. Parallel and Distributed Simulation Systems, Wiley Interscience.

Jennings, N.R. 2000. On agent-based software engineering, *Artificial Intelligence* 117: 277-296. Elsevier.

Natrajan, A., and P.F. Reynolds, Jr. 1999. Resolving Concurrent Interactions. In *Proceedings of 3rd International Workshop on Distributed Interactive Simulation and Real Time Applications*, 85-92.

Uhrmacher, A.M., P.A. Fishwick and B.P. Zeigler (eds). 2001. Special issue on agents in modeling and simulation: exploiting the metaphor, In *Proceedings of the IEEE,* 89 (2): 127-129.

Wang, L., S.J. Turner and F. Wang. 2003. Interest Management in Agent-Based Distributed Simulations, In *7th IEEE International Workshop on Distributed Simulation and Real-Time Application*, 20-27.

AUTHOR BIOGRAPHIES

LIHUA WANG is a Master student of Nanyang Technological University, Singapore. She received her B.Eng in Computer Science and Engineering from Xi'an Jiaotong university, China in 2001. Her research interests include agents and distributed simulations. Her email address is <wlh@pmail.ntu.edu.sg>.

STEPHEN JOHN TURNER is Director of the Parallel and Distributed Computing Centre in the School of Computer Engineering, Nanyang Technological University (Singapore). 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, distributed virtual environments, grid computing and multi-agent systems. His email address is <assjturner@ntu.edu.sg>.

FANG WANG is a PhD student of Nanyang Technological University (Singapore). She received her Bachelor degree from Huazhong University of Science and Technology (China). Her current project title is autonomous agents for distributed virtual worlds. Her email address is <wfirene@pmail.ntu.edu.sg>.