Lucid Platform: Applying HLA DDM to Multiplayer Online Game Middleware

Elvis S. Liu  Milo K. Yip  Gino Yu
Multimedia Innovation Centre
School of Design
The Hong Kong Polytechnic University
{mcelvis, mcmilo, mcgino}@polyu.edu.hk
http://www.lucidplatform.com

Figure 1: Screenshots of Virtual Three Kingdoms - a demo game using Lucid Platform 1.0

ABSTRACT
Multiplayer Online Game (MOG) middleware is a set of software toolkits that assists and simplifies the MOG development process. Using the middleware enables the developers to focus more resources on improving game design while spending less time and money on 3D rendering and server infrastructure issues. As the scale of MOGs grows rapidly, providing scalable data sharing service, or interest management, becomes one of the major requirements for MOG middleware. In this paper we present the details of interest management system of our MOG middleware - Lucid Platform. We employ the Data Distribution Management (DDM) service of the High Level Architecture (HLA) as the main concept of our design. Since the message filtering mechanisms of DDM are very complex and may impose significant computational overheads, we use an efficient interest matching algorithm for the implementation. Experimental results, based on the demo games and the emulator programs, show that our approach works well in practice.

Categories and Subject Descriptors
C.2.4 [Computer-Communication Networks]: Distributed System—Client/Server, Distributed Applications; I.6.7 [Simulation and Modeling]: Simulation Support Systems

General Terms
Algorithms, Performance

Keywords
Computer Games, Collaborative Virtual Environments, Middleware, Interest Management, High Level Architecture, Data Distribution Management

1. INTRODUCTION
Multiplayer Online Games (MOGs) tend to be the most enjoyable type of computer games which enable players to simultaneously interact in a shared virtual world even though they are located differently around the world. The games like Lineage II [1], World of Warcraft [2], Final Fantasy XI [3] and Everquest II [4] all attract hundreds of thousands, if not millions, of players around the world and demonstrate that there is a viable market for the gaming industry. The huge rewards for success are attractive; however, the technology requirements are not easy for game developers to accomplish. Creating MOGs is expensive, technically difficult and time consuming. 3D rendering, AI, Physics, data communications, data persistence and network security all present potential technical barriers. Although some third-party 3D game middleware (game engines) such as Unreal
The remainder of this paper is organized as follows. In section 2, we present the background and related work of HLA and interest management. In section 3, we give a brief description of Lucid Platform 1.0 and its interest management module. In section 4, we discuss the details of the scalable interest management algorithm. In section 5, we compare the experimental results of brute force approach and our approach. Finally, in section 6 we give the conclusions of this paper and discuss briefly the future plan of our work.

2. backgrounds and related works

In this section we briefly review the existing approaches of interest management and data distribution management service of the HLA.

The interest management technique has been studied extensively in many fields such as military simulations, collaborative virtual environments and MOG development. Various interest management schemes have been proposed throughout the years which can be mainly divided into two categories: zone-based and aura-based. NPSNET [9, 10], SPLINE [11] and DIVE [12] are the systems that use zone-based approach. They divided the virtual world into a number of zones; participants within the same zone are considered to have common interests that they receive events or updates only from that zone. Choosing a proper resolution is one of the major considerations for this approach. If the zone size is large, each zone would contain a large number of simulation entities. As a consequence, participants might receive data from the simulation entities that they are not interested. However, using small zone size implies that the number of multicast groups would be large thus increases the management overheads. The MASSIVE systems [13, 14, 15] adopted the aura-based interest management approach. The basic idea is using auras to represent the interests for each participant. When the auras overlap, a connection between the aura owners is established and messages are exchanged through the connection. This approach provides a more precise message filtering mechanism than the zone-based schemes; however, more computational efforts are required to test the overlapping statuses for the auras.

Although the zone-based and aura-based approaches are common for message filtering based on spatial information, they do not concern other aspects such as class and attribute of simulation entities. For examples: in a fantasy role-playing game, a knight may not detect a specter using his aura-based interest management approach and receive its data; or in military simulation game, a low rank soldier may not receive information that sends to his nearby general. These can be supported by the HLA services as class-based/attribute-level filtering and value-based filtering through multidimensional routing space.

The HLA is developed by the Defense Modeling and Simulation Office (DMSO) of U. S. Department of Defense which subsequently became IEEE standard of computer simulation (IEEE 1516) [16] in 2000 and adopted by the Object Management Group (OMG) as Facility for Distributed Simulation Systems in 2001. The HLA specification provides a general infrastructure and services for distributed simulations which consists of three core components including federation rules, interface specification and the object model template (OMT). A Run Time Infrastructure (RTI) is a soft-
ware implementation that meets the HLA interface specification and executes the federation execution (the simulation system). Six services, namely: federation management, object management, declaration management, ownership management, time management and DDM, are supported by the RTI. The DDM provides flexible mechanism for publishing and subscribing interests through multidimensional routing space. The basic structure of routing space is as following:

Routing space: A routing space is a collection of dimensions.

Dimension: Dimensions are used to define regions.

Extent: An extent is a bounded range defined along each dimension of a routing space.

Region: Each region is defined in terms of a set of extents.

The DDM interest matching process begins by specifying subscription and update regions. An object is said to be interested by a federate (an individual simulation) if and only if the following two conditions are satisfied:

1) At least one of the object’s attributes is subscribed by the federate (through declaration management); and
2) At least one update region associating with the object overlaps the subscription region(s) of the federate.

The usage of region in DDM is very flexible: it can be used for zone-based as well as aura-based interest management. For the latter case, the subscription and update regions may need to be modified as the states of objects change over time. Such modifications may lead to re-matching of the interests. Similar to the client/object interest matching as discussed in Section 1, the complexity of using the brute force approach for this process is \( O(nm) \), where \( n \) is the number of update regions and \( m \) is the number of the subscription regions. This places high demands on computational resources especially for remapping dynamic multicast groups. One approach to dealing with this problem is to use infrequent region updates with lookahead time [18] prediction. Specifically, the regions are only modified after a certain period of time (not at every time step) and thus the demand of remapping operations can be reduced. One disadvantage of this approach is that we must specify sufficiently large regions so that the simulation entity would not move out of scope before the regions are updated. As a consequence, federates may receive irrelevant data with larger subscription regions.

3. LUCID PLATFORM 1.0

In this section we briefly describe the architecture of Lucid Platform 1.0 and highlight the implementation of HLA DDM service in the interest management module.

3.1 Overview

The Lucid Platform [8] is a MOG middleware developed by the Multimedia Innovation Centre (MIC) of The Hong Kong Polytechnic University which provides complete solution for MOG and single player game development. Since the commercialization in May 2005, several demo games, VR applications and commercial MOGs were developed based on the middleware including our latest demo game Virtual Three Kingdoms (as shown in Figure 1) and the award-winning Xbox game Auroral Snare [19] (as shown in Figure 2) developed by Treasure Box.

Two software development kits are included in the Lucid Platform 1.0, namely, Lucid3D [20] and LucidNet [21]. Lucid3D aims to provide software libraries for front-end functionalities such as 3D graphics, audio and AI, where LucidNet provides back-end software libraries such as synchronization, interest management and transaction. Figure 3 shows the architecture of Lucid Platform 1.0.

The logical communication model of LucidNet is based on the client/server architecture. Each client communicates to
other clients via the server. Despite being likely bottlenecks, this model provides best control and coordination of message delivery in a MOG. The server can reduce message traffic to individual clients by not sending packets to those clients if the packet in question is not interested by the potential recipients. This is handled by the interest management module of the LucidNet.

In addition to the software libraries, Lucid Platform 1.0 also provides a set of tools that assist MOG development. Among which the schema compiler is used to compile the object schemas that contain the class-based metadata that are used by the interest management system. We describe the details of object schema in the following section.

3.2 Object Schema and Schema Compiler

The user-supplied object schema (in XML format) is used to describe the properties of object class, attribute, methods, dimensions and routing space. These metadata are essential for the whole development process, since they can be used for interest management module and remote method invocation (RMI) module 3. Actually, one can treat the object schema as a combination of the functionalities of HLA OMT [7] and CORBA IDL [22]. Figure 4 illustrates the format of the schema.

Figure 4: The Format of XML Object Schema

Currently 12 primitive types are supported by the Lucid Platform 1.0 object schema, including integer, floating point number, character, string, etc. Similar to CORBA IDL, the users can define their object classes by any combination of these types. The details of the supported types and the schema format are presented in [21].

The schema compiler is one of the tools provided by the Lucid Platform 1.0 which can be used to compile the XML object schema and produce the stub and skeleton source files for RMI as well as the source files that contain the meta information for interest management (e.g., classes, attributes and routing space). At the initialization stage of the game application, the Lucid Platform provides means that automatically import these source files and generate the handles (IDs) for classes, attributes and dimensions according to the meta information. As we discuss in the following section, after the clients obtain these handles from the RTI, they can use them to specify the class-based interests during runtime.

3.3 The Interest Management Module

The interest management module of LucidNet provides class-based and value-based filtering mechanisms which are based on the concepts of the HLA DDM and part of the declaration management and object management services. If the developers want to build their MOGs without interest management, they can simply apply the RMI module which makes direct data communication between clients and server. Such approach only allows message broadcasting without any filtering process. On the other hand, if interest management module is applied, each game client would communicate with the server through a proxy program. The proxies are client agents that reside on the server and are used to invoke RTI functionalities. Each of them contains two main ambassador interfaces, namely, the RTIambassador and the FederateAmbassador:

RTIambassador: The client proxy requests service (e.g., subscribing interests, modifying regions, etc.) of the RTI by calling member functions of the interface RTIambassador.

FederateAmbassador: The RTI sends messages and responses (e.g., receiving updates) to the proxies by calling functions implemented in the FederateAmbassador.

The details of the member functions of these two interfaces are presented in [21]. It is very important to understand that FederateAmbassador is abstract. The users of Lucid Platform must implement the functionalities declared in FederateAmbassador which creates a derived class that contains the actual implementation for each of the callback functions. An instance of this user supplied class is required to register to the RTI. Hence the RTI can invoke the callbacks according to the reference of the corresponding instance. Figure 5 illustrates the architecture of interest management module.

4. SCALABLE INTEREST MATCHING ALGORITHM

Since runtime performance is an important factor for real-time applications such as MOGs, we applied a scalable interest matching algorithm which borrows the ideas from the classic collision detection system I-COLLIDE [23]. The problem of real-time collision detection is very similar to the interest matching problem. Since there are $C^n_2$ potentially collided object pairs for a scene of $n$ objects, using the brute force approach to detect collision is an $O(n^2)$ process. Many prior works [24, 25, 26, 27, 28] were developed to reduce the $O(n^2)$ computational complexity. Although they are common for fast object culling, they are not suitable to our system because they concern only the 3-dimensional space. The concept of dimension reduction of the I-COLLIDE system, however, does not have this limitation. We use this concept to define multidimensional region intersection as following:

3The details of the RMI module are beyond the scope of this paper but are discussed in [21]
Two regions overlap in n-dimensional space if and only if their extents on the 1\textsuperscript{st}, 2\textsuperscript{nd}, ..., and n\textsuperscript{th} dimension overlap.

A region in Lucid Platform can be defined as a subscription region, an update region, or both. Each region contains a set of class-based subscriptions \( S \) and a set of object updates \( U \). When two regions overlap, further investigation of the class-based matching of the corresponding \( S \) and \( U \) is necessary. The class-based subscription and object update are defined as following:

Class-based subscription: \( S=(F,C_S,A_S)\in S \), where \( F \) is the federate, \( C_S \) is the object class and \( A_S \) is the attribute handle set.

Object update: \( U=(O,C_U,A_U)\in U \), where \( O \) is the object instance, \( C_U \) is the object class and \( A_U \) is the attribute handle set.

\( O \) is said to be interested by \( F \) if and only if \( C_S=C_U \) and \( A_S\cap A_U\neq\emptyset \). When \( O \) is updated, the values of the result of \( A_S\cap A_U \) will be sent to \( F \).

\( R_1, R_2 \): Two overlapped regions
\( S_1 \): A set of class-based subscriptions associated with \( R_1 \)
\( U_1 \): A set of object updates associated with \( R_1 \)
\( S_2 \): A set of class-based subscriptions associated with \( R_2 \)
\( U_2 \): A set of object updates associated with \( R_2 \)

BEGIN
  FOR each subscription \( S=(F,C_S,A_S)\in S_1 \)
    FOR each object update \( U=(O,C_U,A_U)\in U_2 \)
      IF \( C_S=C_U \) and \( A_S\cap A_U\neq\emptyset \)
        Establish connection between \( O \) and \( F \)
        END IF
  END FOR
END FOR

FOR each subscription \( S=(F,C_S,A_S)\in S_2 \)
  FOR each object update \( U=(O,C_U,A_U)\in U_1 \)
    IF \( C_S=C_U \) and \( A_S\cap A_U\neq\emptyset \)
      Establish connection between \( O \) and \( F \)
    END IF
  END FOR
END FOR

END

The scalable matching algorithm is developed based on the concept of the Sweep and Prune algorithm of I-COLLIDE [23]. We first construct a list of endpoints of the extents for each dimension. By sorting these lists, we can determine which extents overlap. Quick sort would be a good choice for the initialization stage, the complexity for this process is \( O((m+n) \log(m+n)) \) where \( m \) is the number of subscription regions and \( n \) is the number of update regions. If the extents of two regions, say \( R_1 \) and \( R_2 \), overlap in all dimensions, we then carry out the class-based interest matching process (as shown in Figure 6).

After the initialization stage, we may use insertion sort for the sorting process during runtime. Since temporal coherence is exploited, we can cache the matching results of the previous time steps. The overlap status and the class-based matching results are only modified when the insertion sort performs a swap. In environments where the objects make relatively small movements between consecutive time steps, the lists would remain almost sorted. The complexity of re-sorting the lists using insertion sort would be \( O(n + m + s) \) for each dimension, where \( s \) is the number of swaps. In the general case, the value of \( s \) could be extremely small.

5. EXPERIMENTAL RESULTS

In this session we describe the evaluation of the Lucid Platform 1.0 interest management module. There are two primary concerns of our evaluation: 1) the filtering strength of the interest management system; 2) the runtime efficiency of the scalable interest matching algorithm. We carried out several experiments, based on the emulator program of Lucid Platform 1.0, to test the performance of our system. The emulator was used to generate synthetic client connections and data traffics on the network. By doing so we are able to simulate different MOG scenarios (from small-scale to large-scale) and investigate the system performance in these scenarios. All of the tests were run on a Pentium 4 2.8GHz with 1GB main memory and used the following experimental set-ups:

Occupation Density: We define the occupation density as the proportion of the scene volume occupied by the subscription and update regions. Greater occupation density implies greater probability of matched interest (for the number of objects is constant). In our experiments, the occupation density is set to 1%.

Object Movements: Average speed of a moving object equals to 50% of its radius per time step.
Object Distribution: The objects are distributed randomly to the scene.

Number of Clients: The number of clients varied from 20 to 200 in order to analyze the efficiency for each algorithm.

Number of Objects: One object is created for each connected client.

Size of Data Packet: 32 Bytes per object.

Number of Dimensions: We tested the algorithms in 3-dimensional routing space.

Number of Regions: An update region and a subscription region are associated with each moving object.

Figure 7: Performance Comparison of State Broadcasting and Interest Management System of Lucid Platform 1.0

We first investigate the filtering strength of the interest management system. In order to do so, we need to measure the total packet size that the server sends to the clients at each time step. Figure 7 compares such measurements regarding object state broadcasting and our interest management scheme. For the state broadcasting approach, the server sends all object states to all clients without any filtering process. The experimental results show that as the number of clients (as well as the number of objects) increases, this approach consumes significant network bandwidth. Our interest management scheme, however, performed very well on filtering irrelevant data. The total size of packets that sends by the server does not have significant increase when the number of clients (and objects) becomes large. This indicates that our interest management system provides a more scalable approach for data sharing.

Figure 8: Performance Comparison of Brute Force Approach and the Scalable Interest Matching Algorithm

The second set of experiments focused on comparing the runtime efficiency of the brute force approach and our scalable matching algorithm. The brute force approach, as discussed in the previous sections, begins by comparing all updates with all subscription regions. If collision occurs, we further investigate the class-based subscriptions and object updates that are associated with the collided regions. This process is performed at every time step, no matching result would be cached. Figure 8 shows the experimental results of the two approaches. It is not difficult to notice that the scalable interest matching algorithm requires less computational effort for interest matching. This is particularly significant when we gradually increase the number of objects. In summary, our approach imposes less computational overheads and scales much better than the brute force approach.

6. CONCLUSIONS AND FUTURE WORKS

In this paper, we have reviewed the detail design of the interest management system of our MOG middleware - Lucid Platform. Our system makes use of the HLA DDM service which provides a generic framework for class-based and value-based message filtering. Since the filtering mechanisms are complex and may impose significant computational overheads, we have proposed an approach of efficient interest matching based on caching. Experimental evidence demonstrates that our system performs very well on filtering irrelevant data as well as reducing computational overheads.

Using our interest management system can assist the developers in building scalable MOGs, however, the client/server communication model itself may eventually become a bottleneck. In Lucid Platform 2.0, we plan to develop a middleware for multiple-server architecture. We will revise our interest management system to allow distributed processing that the workload can be shared among different server and further enhances runtime efficiency.

7. ACKNOWLEDGEMENTS

Our work was supported by SME Development Fund (SDF) of the Trade and Industry Department, HKSAR Government, under the grant D02002077: "Mini-Scale Multiplayer Online Game Development Platform". We would like to thank Treasure Box for the permission of using the screenshots of Auroral Snare.

8. REFERENCES


