USING AN OBJECT EXCHANGE MODEL FOR DISTRIBUTED SIMULATION

ZHAO NA

(B.S., Beijing Normal University, P. R. China)

A THESIS SUBMITTED

FOR THE DEGREE OF MASTER OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE

NATIONAL UNIVERSITY OF SINGAPORE

2003

Acknowledgements

I sincerely appreciate people who have contributed to this thesis and to making my time in the Computer Science graduate program a great learning experience.

Firstly I would like to thank my supervisor, Dr. Gary Tan Soon Huat. He has been more than available throughout my research, implementation, and the writing of research papers. His input and direction was invaluable in the creation of this thesis, and the research it contains. Without his help, this thesis would not exist.

Appreciation also goes to Dr. Simon Taylor who gives me countless help in my research. His guidance and constructive criticism benefited me a lot.

I am grateful to School of Computing, National University of Singapore which offers me funding to finish my master study.

Words are never enough to express my deep love and thanks to my parents who give me life and encourage me to face difficulties on my way of growing up. Gratefulness and love also go to my dear sister and my friend, Chen, for their unwavering supporting, understanding and encouragement.

Finally, I would like to thank my lab fellows in the Modeling and Simulation Group, Hu Yu, Karthik Shenoy, Ng Wee Ngee… who create a harmonious atmosphere around me and share a happy time with me.

Table of Contents

List of Figures

List of Tables

Publications Arising from This Thesis

G. Tan, N. Zhao and S.J.E. Taylor, "*An Object Exchange Model Template for Distributed Simulation."* Proceedings of the European Simulation Interoperability Workshop, 03E-SIW-102, 2003.

G. Tan, N. Zhao and S.J.E. Taylor, *"Automobile Manufacture Supply Chain Simulation in GRIDS Environment."* Proceedings of the Winter Simulation Conference, 2003.

Abstract

Being in the "Internet Age", any application technology has to consider its innovative application in distributed environment, otherwise it will lose its vitality soon. Distributed simulation, which refers to the execution of simulations on loosely coupled systems (such as geographically distributed computers interconnected via the Internet) [2], is one of the fast growing distributed applications that leave a golden era before us.

Current research in distributed simulation is moving to connect existing simulation models together by the exchange of information so that they can constitute a complete simulation system. Research in distributed simulation middleware technology and standardization of the information exchange format has gained attention as distributed simulation becomes important.

In this thesis, we introduce the Generic Runtime Infrastructure for Distributed Simulation (GRIDS) and the GRIDS Object Exchange Model Template (OEMT). The main aim of this research is to investigate and evaluate the capability of the GRIDS middleware and the OEMT in supporting Distributed Supply Chain and other various types of distributed simulation applications through two case studies --- Automobile Manufacture Supply Chain Simulation and Singapore Mass Rapid Transit (MRT) System Simulation. Furthermore, from the experience gained from the case studies, we improve the OEMT into a more robust and advanced one.

Chapter 1 Introduction

The scientists who created ARPAnet in the 1960's never imagined that the Internet would become so powerful after 40 years' development. At the beginning of the new century, our society is in the transition from an industrial society to an information society. The Internet, with its magic is the main force in driving this transition. Today, the Internet has more than ten million domain names, hundred million connected computers and billions of customers. Even the most conservative man cannot deny that Internet and the World Wide Web are changing our life!

Being in the "Internet Age", any application technology has to consider its innovative application in distributed or web-based environment, otherwise it will lose its vitality soon. Distributed simulation is one of the fast growing distributed applications that leave a golden era before us.

The aim of this chapter is to provide the readers with a general understanding of the fundamental of this research. We will first review the basic background knowledge of distributed simulation and its application in supply chain management. Then we will give an overview of the distributed simulation middleware infrastructures. Furthermore, this chapter introduces the research motivation, objectives and the structure of this thesis.

1.1 Distributed Simulation

There are two ways to analyze the behaviour of a system scientifically. One way is to use mathematical methods (such as algebra, calculus or probability etc.) to obtain exact information on questions of interest [2]. This is called an analytic solution. However, most real-world systems are too complex to be evaluated analytically, and these models must be studied by another means --- **Simulation**. According to Robert E. Shannon [2], simulation is "the process of designing a model of real system and conducting experiments with this model for the purpose either of understanding the behavior of the system or of evaluating various strategies (within the limits imposed by a criterion or set of criteria) for the operation of the system." A simulation creates an abstract representation of a system, and then gains insight into the working of the system, or predicts the system's future performance, or tests out the results of changing some aspects of the system. The whole process is made on the model, without having to manipulate or observe the actual system.

Distributed simulation is concerned with the execution of simulations on geographically distributed computers interconnected via a local area and/or wide area network [22, 30, 31]. The primary goal of distributed simulation is to obtain higher performance via parallel execution, but its advantages are not restrained in high performance. Distributed simulation system also offers benefits such as responsiveness, resource sharing, information secrecy, reusability, fault tolerance and so an.

Due to these inherent fortes of distributed simulation, it has been a useful and powerful tool in numerous and diverse application areas [17] such as evaluating military weapons systems or their logistics requirements, determining hardware and software requirements for a computer system, analyzing financial or economic systems and so on. One significant application area of distributed simulation is designing and

analyzing supply chain.

1.2 Supply Chain Management

A **Supply Chain** is the series of activities that an organization uses to deliver value, either in the form of a product, service, or a combination of both, to its customers [9]. It also includes the flow of information and finances in addition to the material flow [35].

In today's competitive environment, the traditional integrated business in which a single enterprise acts alone seems to be a thing of the past. The trends of globalization of markets have forced even large organizations to rely on hundreds or even thousands of external firms or suppliers to deliver value to the ultimate customers. Competitive success of an organization is no longer a function of its own efforts, but depends on the efficiency of the entire supply chain. Therefore, building an effective supply chain/supply network [8] is fast becoming paramount in today's marketplace. The effort of managing and coordinating the activities between separate entities is often referred to as **Supply Chain Management (SCM)** [28]. It can be defined as achieving a sustainable competitive position and maximizing shareholder value by optimizing the relationship of process, information, and physical goods among internal and external trading partners [9]. Typically, SCM costs represent a majority of the operating expenses of most companies. These costs can range from as low as 30% to as high as 75% [9].

A supply chain is a complex system and there are sources of large uncertainties in the supply chain. Successful SCM requires carefully defined approaches to test and analyze the performance of the chain [24]. It is obviously not advisable to do the corresponding analysis on the real world system due to the high cost. What is needed is a tool that can give visibility of the entire supply chain that allows for the testing of

numerous "what if" scenarios such as outsourcing, consolidating vendors, collaborative planning, or implementing e-business [9].

Simulation has been identified as one of the best means to analyze supply chains. Commercial simulation tools for SCM have been released in recent years, such as the IBM Supply Chain Analyzer (SCA) [13], and the integrated tools of simulation and optimization by i2 [34]. These simulation tools are powerful in supply chain modeling and optimization capabilities. However, these tools are limited by their lack of capacity for parallel execution. The current emerging electronic commerce and dynamically changing business environment requires for a next-generation supply chain modeling and simulation environment which include scalable and efficient model execution and support for flexible future extensibility based on an open industry standard.

The **Distributed Supply Chain (DSC) Simulation** [3] is such a second-generation tool, which introduces the distributed simulation approach into Supply Chain Management. A DSC simulation models a supply chain across multiple businesses. It involves the organizational departments responsible for each activity and the external suppliers and customers who are part of the integration supply chain, and simulates the flow of materials and information through multiple stages of manufacturing, transportation and distribution. A DSC simulation offers analysts and decision-makers a means to replicate the behavior of complex systems as they operate over time. In addition, the distributed characteristic of the simulation models offers great benefits such as fast and efficient simulation execution, taking advantages of the functionality of various vendors' simulation products, allowing each organization hides its inner working information, reducing the costs and time of building a new simulation model and so on. Companies may react faster to global competition by using this approach to investigate efficiency and effectiveness improvement in their supply chains.

1.3 Distributed Simulation Middleware Infrastructures

Building a detailed model of the supply chain does not pose a problem when the chain involves only a single enterprise. However, since a DSC simulation often involves multiple companies across enterprise boundaries, each of these companies may already have its own simulation program and they may not like to share their models and internal data with other companies. In addition, the existing simulation models may be implemented using different languages and packages on different platforms which may even be located at different places. The lack of interoperation, lack of portability to multiple languages and lack of ability to execute over the Internet still obstruct the progress of DSC simulation.

Although there are many techniques involved in building simulation in a distributed or web-based environment [1, 23], the most common way to solve this problem is by the use of standard programming interfaces and protocols that provide a uniform means and style of access to various simulations which may be based on different platforms. Such standardized interfaces and protocols have come to be referred to as **middleware** infrastructure. Research in distributed simulation middleware technology has also gained attention as distributed simulation has become important. These middleware are developed based on several premises or assumptions. Firstly, no single, monolithic simulation can satisfy the needs of all the users for the differences in users' interests and requirements. No one can anticipate all the uses of simulation and all the ways of simulation could be combined in a monolithic system. And it is possible to decompose a large simulation problem into smaller parts which are easier to define, build and verify. Secondly, simulation builders vary in their knowledge background of domains to be simulated. This makes their products different in detail. Finally, future technologies and tools must be incorporated and future

requirements must be considered.

These observations led the middleware designers toward the following goals:

- It should be able to integrate the software from different sources.
- It should have the capability to insulate the components from differences in the implementation technology so that it can hide the complexities and disparities of different simulations.
- It is responsible for the communication between individual simulations. It should be able to marshal the information, pass it through network to the targeted simulations, and then de-marshal it into the format which is understandable by the simulations.
- It should provide certain services to facilitate interoperation of simulations. E.g. synchronization service to coordinate the time steps of cooperating simulations.

The High Level Architecture (HLA) [18, 29, 44], Generic Runtime Infrastructure for Distributed Simulations (GRIDS) [33, 37, 39] and Common Object Request Broker Architecture (CORBA) [6, 21, 41] are all such middleware which are developed by different organizations with distinct characteristics, but share the property to support the reuse and interoperation of simulations. This research concentrates on GRIDS which is a lightweight message-oriented middleware with extensible features and package interfaces capable of supporting the demands of distributed simulation. Similar to HLA, it is used to research interoperability and reuse by linking simulations together and was originally developed to support Distributed Interactive Simulation. The main purpose of the infrastructure is to coordinate the activities of distributed components with additional functionality via the use of a novel service distribution model known as **Thin Agents**, which are used to support the simulation by providing tasks such as optimization and assistance [32]. Thus, it is a very appropriate

architecture to be used in Distributed Supply Chain (DSC) Simulation and other simulation areas given the distributed nature of such simulations.

1.4 Motivation

Current research in distributed simulation is moving to connect existing simulation models (known as federates[1]) together by exchanging information, so that they can constitute a complete simulation system. The motivation of this research is that connecting existing models could be more cost effective than recoding the separate models into a single model. One of the research issues is the standardization of the information exchange format so that they may be understood by each model.

In a distributed simulation, information transfer is very important because federates realize interactions though exchange data. Federates keep sending and receiving information between one another during their lifetime. Information has various formats such as data, message, and file or sometimes the information is an object itself. For example, in an automobile supply chain simulation, the tyre factory must send its products together with its quantity and quality parameter to the car assembly factory. In this case the information takes the form of objects (representative of entities transferred between the models). However, a problem is the specification of these objects so that they may be described in a manner relevant to the end user community. The GRIDS Object Exchange Model Template (OEMT) defines the format and syntax for recording information in exchange object models. This common template provides a standardized way of specifying models to be used as input or output, and facilitates understanding between federates of a distributed simulation.

[1] In this thesis, as within common distributed simulation terminology, a single simulation component that participates as a part of the entire simulation is called a **federate**, while the entire simulation is called a **federation**.

During the past few years, the OEMT standard has been developed from theory to real application. The template has been set up, and is under improvement. The emphasis of the research is on using, evaluating and improving the OEMT by two case studies --- **Automobile Manufacture Supply Chain Simulation and Singapore Mass Rapid Transit (MRT) System Simulation.** Both case studies use the OEMT as the standard to specify the objects in full scale. These case studies therefore give practical instances of OEMT application. In addition, GRIDS will also be evaluated through the practical experience gained from the case studies.

1.5 Research Objectives

This work makes several contributions to research areas related to distributed simulation and middleware. The focus of this research is **to investigate the standardization of object exchange models and the GRIDS middleware in distributed simulation to support the DSC and other various types of simulation applications.** The overall research effort has been broken down to a set of research objectives to be achieved:

1) Investigate distributed simulation environment and existing middleware. This is to gain knowledge that is useful in fulfilling subsequent objectives.
2) Evaluate the suitability of GRIDS as a middleware to facilitate distributed simulation; provide useful experience on implementing simulation application in GRIDS environment and suggest how GRIDS might be improved and utilized in future distributed simulation applications.
3) Identify the significance of object passing in distributed simulation; specify the problem to standardize object exchange model; evaluate the capability of the OEMT to standardize object exchange model and offer a new version of the OEMT based on the case study experience.

4) Develop an OEMT application tool kit to automatize the generation of the object exchange model. In addition, implement the Object Exchange Model Repository (OEMR) on-line services and other assistant services in this kit.

1.6 Thesis Structure

This thesis is structured in six chapters based on the research strategy described above. Each chapter addresses a distinct point in carrying out this research. The first chapter sets out to present a brief account of the research background. It addresses some key components of conceptual research which will be discussed in detail in the following chapters. Furthermore, this chapter introduces the motivation, objectives and scope of this research to inform the readers of the contents and structure of this thesis.

Chapter 2 concentrates on the distributed simulation middleware, GRIDS. This chapter first presents the motivation of middleware development. Then it further goes on into detailing the structures, functionalities and characteristics of the GRIDS architecture. This chapter also gives a brief overview of the other two popular middleware HLA and CORBA. Based on the in-depth understanding of these middleware, a comparison among them is made.

Chapter 3 describes the advantages of object passing and the requirement to standardize objects as the motivation of the GRIDS OEMT. It explicates the original OEMT standard and the Object Exchange Model Repository (OEMR) conception in particular.

The two case studies are given in chapter 4. They are used to illustrate how the features of the OEMT and GRIDS contribute to distributed simulation application. The first case study applies OEMT and GRIDS in a DSC simulation. Their application is extended to another area in the second case study. The experience gained from the case studies is summarized and analyzed at the end of this chapter. GRIDS and OEMT are

evaluated. The suggestions on how they might be improved and utilized in future distributed simulation applications are stated in this chapter.

In chapter 5 a new version of the OEMT that is more complete and more powerful in the standardization of object models is provided. An OEMT application tool kit is developed to facilitate the simulation builders in the creation of object models and explains the OEMR online services provided by this kit. This chapter also introduces a novel concept—the Object Exchange Model Dictionary (OEMD) and its functionality. The OEMD services are also implemented in the OEMT kit.

The thesis ends with the conclusion in Chapter 6. This chapter summarizes the work in this thesis, presenting the objectives achieved, the contributions made, and highlights possible avenues for further research.

Chapter 2 Generic Runtime Infrastructure for Distributed Simulation (GRIDS)

As we have mentioned, one of the most important problems of distributed simulation is that the development and deployment of different types of simulation products has far outstripped efforts to standardize all aspects of distributed computing, from the physical layer up to the application layer. This lack of standards makes it difficult to implement an integrated, multi-vendor, enterprise-wide distributed simulation configuration. The middleware, which cuts across all simulations, has the capability to hide the complexities and disparities of different simulations. It is used to overcome incompatibilities of various simulation products and is responsible for the communication between individual simulations.

GRIDS is such middleware for reuse and interoperation of simulations. It supports the reuse of capabilities available in different simulations and the possibility of distributed collaborative development of a complex simulation application.

This chapter introduces the three main middleware used in distributed systems. Since this research is based on the DSC simulation which is a more focused field of distributed simulation, the discussion about the middleware will cater to the needs of this field.

Section 2.1 and 2.2 details the GRIDS structure and execution process. Section 2.3

introduces the other two popular middleware architectures, the High Level Architecture (HLA) and the Common Object Request Broker Architecture (CORBA). This section also compares GRIDS with HLA and CORBA.

2.1 GRIDS Basic Architecture

The Generic Runtime Infrastructure for Distributed Simulation (GRIDS) project was initiated in 1997 with the goal to develop an extensible component-based runtime infrastructure that could be used to coordinate the activities of distributed simulation components. GRIDS is best described as an execution environment capable of supporting a broad range of simulation types [32]. It originally catered to the needs of DSC simulation and is rapidly growing to be widely applicable across a full range of simulation application areas, including education and training, manufacture and transportation.

Instead of the static and fixed functionality advocated by the HLA RTI specification, GRIDS provides the basic simulation services (communications, simulation interface and data services) to connect simulation models and a mechanism to add extra functions (thin agents services) where appropriate [33]. GRIDS's functionality is enhanced by the use of a novel service distribution model known as **thin agents**. These agents may be used to support the simulation by providing tasks such as optimizations and assistance. The extensibility is the principal difference between GRIDS and other approaches to distributed simulation middleware.

The middleware is composed of the following major elements:

- **Boot Server:** a single process used to coordinate the initialization, execution and termination of a distributed simulation. It is responsible only in the initialization of the federation to provide service for federates to register into a federation, then it compiles information from all the federates and sends the relevant information

to each federate, before excluding itself from the federation execution. It is regarded as the Central Runtime Component (CRC) of the middleware.

- **Client:** used by the federate to interact with the rest of the federation. It is regarded as the Local Runtime Component (LRC). GRIDS client comprises four primary services: communications, simulation interface, data services and thin agent services.
- **Thin Agent:** GRIDS term for a component service. Thin agents are used to support the federates by providing tasks such as performance optimization, Time Management [42], Data Distribution Management [12, 36] and other special simulation services. Their specific function is entirely dependent on the requirements of the application and therefore can provide general services and more specific services.
- **Metadatabase:** the general data structure in GRIDS used to store information.

Figure 2-1 illustrates graphically the middleware's setup in a typical GRIDS federation. It is composed of a single boot server and several clients. Each simulation federate is connected to a GRIDS client via an interface. Thin agents are distributed to participating clients and instantiated to provide the required services.

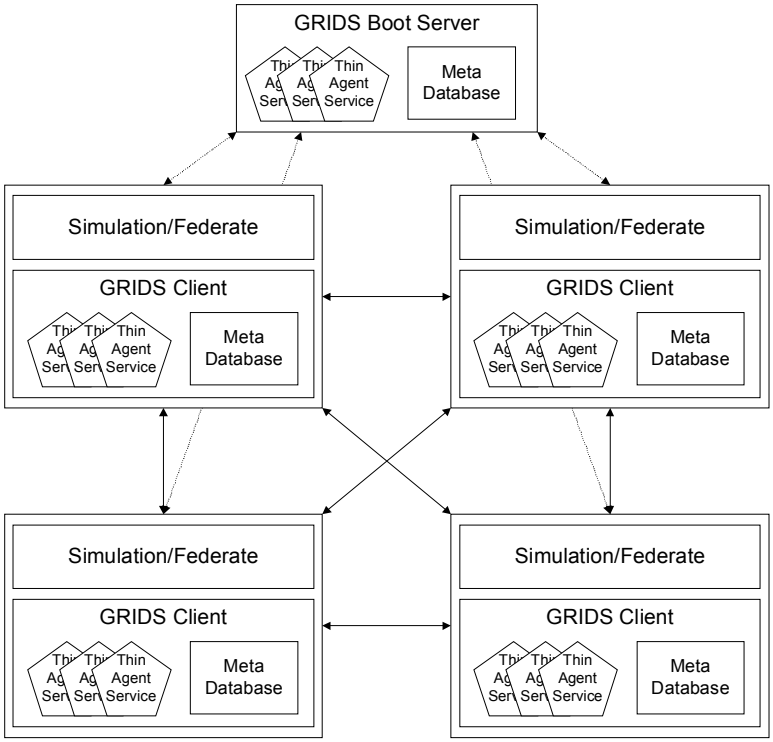

Figure 2-1: A Typical GRIDS federation

2.2 GRIDS Execution Stages

An integrated GRIDS execution session is divided into 5 stages: Initialization, Register, Broadcast, Runtime and Terminate which are detailed as follows [32]:

- **Initialization:** Initialization involves the starting of a GRIDS boot server. The server is loaded with thin agents that are to be used to support the simulation exercise.
- **Register:** Registering involves individual simulation nodes making their presence known to the GRIDS boot server and publishing the initial state variables of that node. Additionally, the boot server builds up the namespace of all the registered clients, and builds a central entity list of all entities in the simulation. Once all clients are registered the server closes all incoming connections for registration.

- **Broadcast:** Upon a simulation "Start" event, the boot server broadcasts to all registered clients the entire entity list built up during registration. The entity list is stored in the internal database on each GRIDS client. In addition to broadcasting the entity list, the server broadcasts the namespace for all participating clients to be stored internally within each GRIDS client.
- **Run:** Once all entity lists and namespaces are broadcast to the individual clients, the server issues a "go" command to all the clients, signaling the start of the simulation. At this point, the server ceases its interactions with the clients. The clients now communicate directly as necessary in a peer to peer fashion with other nodes in the simulation. The GRIDS client is responsible for synchronizing entity attributes between the local and remote nodes.
- **Terminate:** Once the simulations have completed executing, the clients register back with the boot server signaling that they are exiting gracefully from the federation.

2.3 Comparison between GRIDS and Other Popular Middleware

2.3.1 High Level Architecture (HLA)

The High Level Architecture (HLA) is a general purpose middleware architecture for simulation reuse and interoperability. It was developed under the leadership of the Defense Modeling and Simulation Office (DMSO) to support reuse and interoperability across the large numbers of different types of simulations developed and maintained by the DoD.

The HLA is defined by three concepts:

- Object Model Template (OMT): The HLA OMT defines the format and syntax for recording information in HLA object models, to include objects, attributes, interactions, and parameters. It does not define the specific data (e.g., vehicles, unit types) that will appear in the object models, but provides a commonly understood mechanism for specifying the exchange of data and general coordination among members of a federation and describing the capabilities of potential federation members [16, 29].
- HLA Rules: The HLA rules comprise a set of underlying technical principles and conventions which must be followed to achieve HLA compliance [14]. They describe the responsibilities of federates and federations designers.
- Runtime Infrastructure (RTI): The HLA RTI can be viewed as the special purpose distributed operating system software that provides a set of common interface services utilized during the runtime of an HLA federation [15]. The run-time services of the RTI fall into six categories:
  - Federation Management
  - Declaration Management
  - Object Management
  - Ownership Management
  - Time Management
  - Data Distribution Management

An example of a federation in a HLA environment is shown in figure 2-2. As shown, each federate presents to the RTI an interface called FederateAmbassador, and the RTI offers an RTIambassador interface to each federate. The Federates and the RTI communicate through invoking operations on the two interfaces.

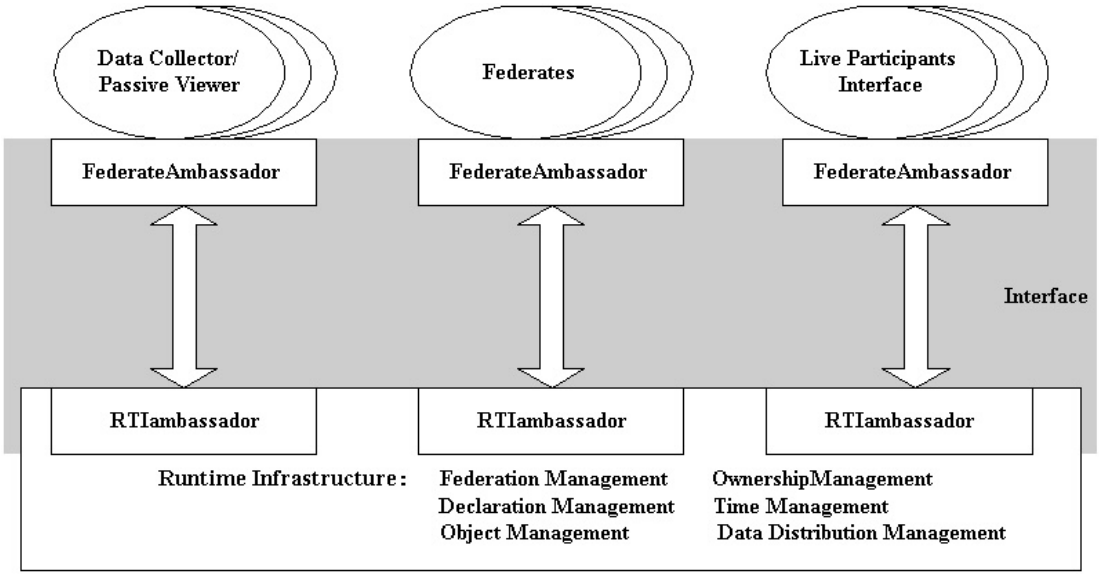

Figure 2-2: A Typical HLA Federation

2.3.2 Common Object Request Broker Architecture (CORBA)

The Common Object Request Broker Architecture (CORBA) is an emerging open distributed object computing infrastructure being promulgated by the Object Management Group (OMG). It is designed based on the OMG Object Model and supports object-oriented standardization and interoperability.

The two most important features of CORBA are language independence and platform independence. It automates many common network programming tasks such as object registration, location, and activation; request de-multiplexing; framing and error-handling; parameter marshalling and de-marshalling [6]. It allows applications to communicate with one another no matter where they are located or who has designed them.

The following figure illustrates the primary components in the CORBA ORB architecture [21, 41].

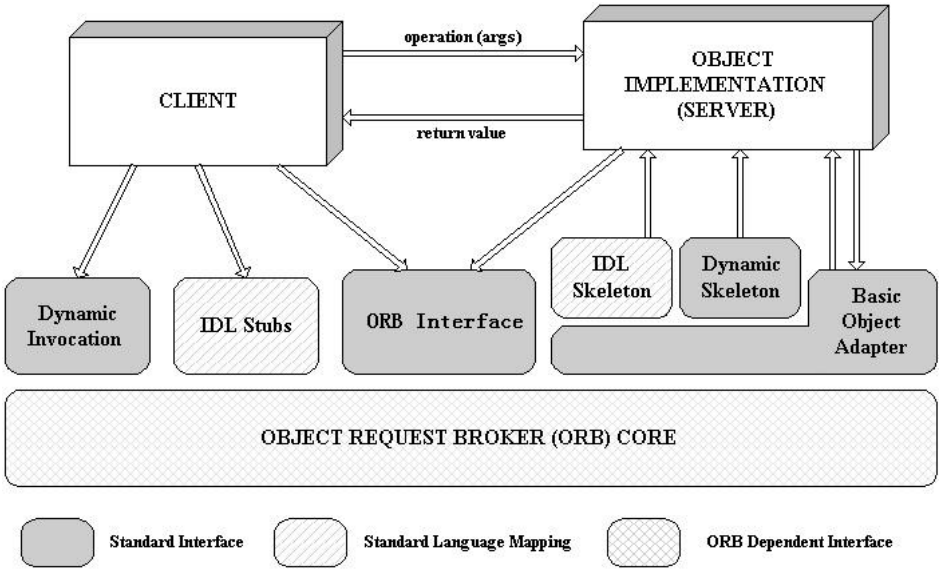

Figure 2-3: The CORBA ORB Architecture

The central component of CORBA is the *Object Request Broker* (ORB). The ORB is the middleware that establishes the client-server relationships between objects. It hides the low-level details of platform-specific networking interfaces, allowing developers to focus on solving the problems specific to their application domains rather than having to build their own distributed computing infrastructures.

Using an ORB, a client can transparently invoke a method on a server object, which can be on the same machine or across a network. The ORB intercepts the call and is responsible for finding an object that can implement the request, pass it the parameters, invoke its method, and return the results. In order to make a request the client communicates with the ORB Core through the IDL *stub* or through the DII. The ORB Core then transfers the request to the object implementation which receives the request as an up-call through either an IDL skeleton, or a DSI. By assistance of these components, the ORB provides interoperability between applications on different machines in heterogeneous distributed environments and seamlessly interconnects multiple object systems. These strong points have gained CORBA universal business

notice and acceptance in distributed simulation application [4].

2.3.3 Comparing GRIDS with HLA and CORBA

W.N. Ng has provided general comparisons between HLA and GRIDS for function, implementation and design issues [42]. This section will compare the federation interoperation mechanisms in HLA and GRIDS. We also discuss the suitability of GRIDS and CORBA within the DSC Simulation field.

The HLA evolved from a military simulation background and has been used in other application areas [5, 40], whereas GRIDS originally catered to the needs of DSC simulation. Although both of the middleware have been used in other application areas, their different origins make them distinct in several aspects. One of the most important differences between them is the dissimilar interoperation mechanism among federates supported by HLA and GRIDS.

The HLA achieves federate cooperation via message-passing. In the HLA federation, federates cooperate through object attributes update or interaction. For example, in an HLA federate, the position (latitude and longitude) of a tank object is always changing referring to the moving tank. These changes are presented by updating attributes. Other federates realize the changes by subscribing to the update messages of the tank object from its owner when the tank object still resides in its owner federate.

GRIDS, on the contrary, supports another type of information passing in DSC simulations: object passing as well as message passing. In other words, in GRIDS environment, federates can cooperate through exchanging whole objects instead of exchanging their attribute information only. The exchange objects refer to the entities transferred in a GRIDS federation. For example, in a GRIDS DSC simulation, the exchange objects represent the flow of materials or intermediate products in the supply

chain. The ability to support object-passing mechanism is a distinguishing characteristic of GRIDS. It offers a lot of benefits to distributed simulation which we will discuss in detail in chapter 3.

Fundamentally, CORBA is a basic application integration technology developed for the distributed systems which is a much wider application area than distributed simulation. Although CORBA has been employed as middleware to facilitate distributed simulation in air traffic control, video games and entertainment, and other needs, it still faces the problem of lack of special services for simulation. Compared with CORBA, GRIDS is originally intended for distributed simulation. It offers services that focus on simulation requirements such as time management, data distribution management and so on.

Other deficiencies also restrict the application of CORBA in DSC simulation. First of all, the current lack of the capability to pass objects by value for most CORBA products can especially affect the design of an application [21]. Furthermore, for a class to be understood by CORBA, its interface must be expressed in IDL. The classes referenced or used by that class must also have their interfaces expressed in IDL because they also need to be accessible to CORBA components. This characteristic make converting an existing application to CORBA architecture an arduous task indeed [21]. GRIDS supports passing objects by value, which is a necessary mechanism in DSC simulation. And modifying an existing application to use GRIDS middleware costs relatively less time and effort, because in GRIDS applications, each federate has a single point of contact with the RTI. There are no requirements of changing interface for every class.

In addition, most CORBA applications work in a mechanism of client/server relationship, such as real-time ATM service. The status of federates in GRIDS

application is relatively equal as they work in a cooperation mechanism.

On the other hand, GRIDS can greatly benefit from CORBA's language independence and platform independence even though CORBA has the foregoing shortcomings. Since its inception in 1991, CORBA has provided great facilities for distributed object-oriented programming that have allowed developers to seamlessly integrate diverse applications into heterogeneous distributed systems. Although there exist lots of alternative technologies such as Socket programming, Remote Procedure Call (RPC), Java Remote Method Invocation (RMI), CORBA still gains preponderance via its language independence, platform independence and suitability for complex applications. It brings true interoperability to today's computing environment. Actually, GRIDS is considering combining CORBA into its architecture to benefit from the strong integration capability it offered. Gaining more powerful simulation middleware from the combination with CORBA might not be far away.

Chapter 3 Object Exchange Model Template (OEMT)

Decision support of supply chain management is one important area that has been fast gaining attention in distributed simulation. In supply chains, each company produces materials or intermediate products that are delivered to the next company in the chain. The products or material flow are represented by objects being exchanged among federates in a DSC federation. This property of DSC simulation is described in section 3.1. Section 3.1 also presents the requirement for middleware to provide for an object-passing mechanism in addition to the traditional message-passing mechanism in DSC and similar kind of simulations.

However, the problem in object-passing is the specification of these objects so that they may be understood commonly by different federates in the federation. The Object Exchange Model Template (OEMT) is a template which provides a standardized way to specify the DSC-relevant object models. Section 3.2 will present this standard in detail.

Section 3.3 highlights the Object Exchange Model Repository (OEMR), which is the common well-known location of a library, where the OEMT and all the OEMT specifications of existing objects are stored for simulation builders for reference and reuse.

3.1 Defining the Problem

3.1.1 Object-Passing in Distributed Simulation

In a distributed simulation application, each federate is standalone. They might use different simulation product (SIMUL8, WITNESS) and even run at remote locations. Their interoperation is realized through the exchange of information. Normally, the passing of information is built upon the message-passing paradigm. This means information is not encapsulated in any structured way, and it is transferred by the middleware as a message with information.

However, the message-passing is not enough to satisfy all the requirements of distributed simulation. In some case, federates produce objects instead of messages. A DSC simulation is such an example. As we mentioned in section 1.1.4, a DSC simulation is a simulation that is composed of models (federates) that represent each of the elements of the supply chain. In a DSC simulation, the interactions among federates can be object instances that are produced, sent and received by the federates from one another. Hence, this is the main concern in a DSC federation, i.e. the exchange of objects among federates.

Of course, the object can be represented by a group of messages that describe this object and contain values that are to be used by another federate. However, several problems arise when using message-passing instead of object-passing in the DSC simulation. Firstly, this approach suffers from loss of details. There is no standard as to how and what type of information should be enclosed in the messages to describe the object exactly. In addition, describing an object requires numerous messages, there may be large overheads for the federates to establish, make agreement and process the information to pass across.

Secondly, there may be cases where variables are required to be used as input to the object to produce a particular output or where the federate is interested in some intermediate information of the object which requires passing some parameters to the object before getting results. Message representations cannot accomplish such requirements.

Finally, the attributes of the objects may be public, protected or private for different secrecy requirements. Protect access modifier specifies that object attributes are accessible only to methods in this object class and its subclasses, and private one is more restrictive. Message representations lack the ability to satisfy the secrecy requirements of federates.

Employing object-passing in DSC simulation solves the above problems efficiently. It is easier to keep the details of the object and to establish a standard thereby objects can be recognized uniquely. Object-passing also allows the provision of methods that the receiving federates may invoke.

Many other application areas in distributed simulation will also benefit from object-passing. The transportation system is another example. As an instance, an underground train system simulation can consist of models that represent different stations in a city. Trains moving among these stations also take the form of objects. A station model takes the arrival trains as input objects and departure trains as output. The train object class includes two kinds of members, one is attributes values (capacity, passenger number, train type, etc.) which carry the data describing the train, and the other is methods which provide mechanism to access attributes and other services.

This research proposes the use of an object-passing mechanism for DSC simulation and even other fields in distributed simulation. A standardization problem of the object will be raised and solved by giving a standard in section 3.2. By following the standard

specification of object, the receiving federate knows what sort of methods can be invoked and what information can be expected from the object.

3.1.2 Standardization of Objects

One important issue of distributed simulation is how to represent parameters and results in messages or objects so that they may be understood commonly by different federates in the federation. There will be no problem if all the federates are programmed in identical programming languages on the same type of machines with the same operating system. However, if there are differences in these areas, the way that numbers and even texts are represented in different federates might be different. The best way to solve this problem is to provide a standard format, so that the native parameters on any machine can be converted to the form of the standard representation. The HLA OMT is such a standard to describe the HLA object model with individual federates or federation. It concentrates on the requirements and capabilities for federate information exchange through message-passing and interactions. However, the OMT does not provide appropriate definitions for the foregoing exchange of objects. To describe the exchange model perfectly, we introduced the Object Exchange Model Template (OEMT) for GRIDS.

3.2 Object Exchange Model Template (OEMT) Specification (version 1.0)

The Object Exchange Model Template defines the format and syntax for recording information in GRIDS distributed simulation object models, as well as mandatory specific data that define each model uniquely from others [11]. The first version of the OEMT Specification was created in 2002.

3.2.1 OEMT Original Elements

The GRIDS Object Exchange Model Template is composed of a group of inter-related elements specifying information about the model. Each model is identified uniquely by the mandatory attributes and methods that must be implemented for each implemented object of the model. The original template for the core of a GRIDS object exchange model uses a tabular format and consists of the following elements [10, 42]:

- Name of Model: to record the product name that the model is emulating.
- Model Description Link: to record the URL link if the description of the model is located somewhere else on the internet. "Model Description Link" cannot co-exist with "Model Description" and "Technical Details".
- Model Description: to record a detailed description of the purpose of this model and the product description.
- Technical Details: to specify the mandatory object attributes and methods that must be implemented of this model.
  - For each object attribute, the attribute name, the attribute description, the attribute accessibility, and the data type of the attribute are required.
  - For each object method, the method name, the method description, the method accessibility by the public, the method's parameters if any (parameter name and data type), and the method return data types are required.
- General Details of each Implemented Object: to specify for each object that has been implemented using this model, the details of the company and the location of the object in the form of an URL.

Actual examples of the OEMT will be given in section 5 when the exchange objects in the case studies are specified using the OEMT standard.

3.2.2 OEMT Data Interchange Format (DIF)

The GRIDS Object Exchange Model Template (OEMT) Data Interchange Format (DIF) is a standard file exchange format used to store and transfer OEMT specifications of object models between simulation builders [10]. The DIF is built upon a common meta-model that represents the information needed to represent and manage object models created using the GRIDS OEMT standard. The DIF uses XML as the standard for declaring object exchange models in the OEMT. The DTD schema of the DIF is given as follows:

```
<!ELEMENT objectModel (modelName, (modelLink|(description, technicalDetails)),
                       implementedObjectDetails*)>
    <!ELEMENT modelName (#PCDATA)>
    <!ELEMENT modelLink (#PCDATA)>
    <!ELEMENT description (#PCDATA)>
    <!ELEMENT technicalDetails (objAttr+, objMethod*)>
        <!ELEMENT objAttr (attrName, attrDescription, attrAccess, attrDataType)>
            <!ELEMENT attrName (#PCDATA)>
            <!ELEMENT attrDescription (#PCDATA)>
            <!ELEMENT attrAccess (private|protected|public)>
            <!ELEMENT attrDataType (#PCDATA)>
    <!ELEMENT objMethod (methodName, methodDescription, methodAccess,
                         methodParameters, methodReturnDataType)>
        <!ELEMENT methodName (#PCDATA)>
        <!ELEMENT methodDescription (#PCDATA)>
        <!ELEMENT methodAccess (private|protected|public)>
        <!ELEMENT methodParameters (pName,pDataType)*>
            <!ELEMENT pName (#PCDATA)>
            <!ELEMENT pDataType (#PCDATA)>
        <!ELEMENT methodReturnDataType (#PCDATA|void)>
    <!ELEMENT implementedObjectDetails (companyName, companyContactInfo,
      companyContactPerson, companyEmail, version, versionDate, referenceLink+)>
        <!ELEMENT companyName (#PCDATA)>
        <!ELEMENT companyContactInfo (#PCDATA)>
        <!ELEMENT companyContactPerson (firstName, lastName)>
            <!ELEMENT firstName (#PCDATA)>
            <!ELEMENT lastName (#PCDATA)>
        <!ELEMENT companyEmail (#PCDATA)>
        <!ELEMENT version (#PCDATA)>
        <!ELEMENT versionDate (day, month, year)>
            <!ELEMENT day (#PCDATA)>
            <!ELEMENT month (#PCDATA)>
            <!ELEMENT year (#PCDATA)>
        <!ELEMENT implementationPlatform (#PCDATA)>
        <!ELEMENT referenceLink (#PCDATA)>
```

3.2.3 Relationship of the OEMT and Object-Oriented Concepts

Although the OEMT is the standardized documentation structure for exchange object models, it does not completely correspond to common definitions of object models in object-oriented (OO) analysis and design (OOAD) techniques. In the OOAD literature, an object model is described as an abstraction of a system developed for the purpose of fully understanding the system. To achieve this understanding, most OO techniques recommend defining several views of the system. OEMT has a much narrower scope than OOAD. It does not intend to represent a scope of the whole system, but focuses on providing a standardized way of specifying object models to be used as input or output, and facilitating understanding between federates of a distributed simulation. In addition, OO objects interact via message-passing, in which one OO object invokes an operation provided by another OO object and gets a returned value. OEMT objects do not directly interact. It is the federates that interact, via exchanging OEMT specified objects. The OEMT specified objects function as the vehicle which carries information among federates.

Although OEMT does not completely correspond to OOAD principles and concepts, it has lots of similarities with OOAD in some sense. At the individual object level, in the OEMT, objects are defined as information encapsulations of data and operations (methods), which is the same with object definition in the OOAD literature. Furthermore, like OO objects, OEMT specified objects encapsulate state locally and associate update responsibilities with operations that are closely tied to the object's implementation in an OO programming language. Federates access object attributes though operations included in object class encapsulation, which also provide security guarantee to private information in the object.

3.3 The Object Exchange Model Repository (OEMR)

The Object Exchange Model Repository (OEMR) is the central location where all object exchange models' information is stored. It is a library which stores all the object models in different areas of applications separately, for distributed simulation builders to access and re-use object models [10]. This work will implement the main functions of the OEMR on-line service. Putting the OEMR online facilitates reuse and allows easy access by simulation builders all over the globe, as well as to provide new object models.

Chapter 4 Case Studies: Applications Using GRIDS and OEMT

Although the OEMT standard has been developed, it is still in the early phase. Its structure is not complete and how to apply it in real applications is still a great challenge. In this chapter, two case studies are designed and implemented to investigate and evaluate the GRIDS architecture and the OEMT standard. Section 4.1 details the design and implementation of an **Automobile Manufacture Supply Chain Simulation** in GRIDS environment. The second case study is a **Singapore Mass Rapid Transit (MRT) System Simulation**. This case study is presented in Section 4.2.

The rationale behind the choice of these two case studies is that, on one hand, both of the case studies are typical distributed systems, and many objects are transferred between distributed nodes in both systems; On the other hand, the two case studies are different in several ways. First of all, the case studies represent distinct application areas of distributed simulation. Case study one is a typical DSC simulation example. Since GRIDS and the OEMT are initiated based on the requirements of DSC simulation, case study one investigates their application in the original application field. The MRT simulation in case study two is an instance of transportation system which is another important commercial area that can benefit greatly from distributed simulation technique.

Secondly, the two simulations differ in the setup of the nodes. In case study one, the nodes are tightly coupled, but in the MRT system, the nodes (stations) are loosely coupled with no immediate or direct feedback when an entity is passed (the differences will be further discussed in section 4.2.5).

These characteristics of the two case studies make them ideal case studies to investigate GRIDS and the OEMT. Through the automobile supply chain and the MRT simulation, we can evaluate the ability of GRIDS to handle the simulation of DSC and transportation system, and the capability of OEMT to describe manufacturing product and vehicle as object.

In addition, the research also introduces an actual implementation tool of the OEMT---XML Document Object Model (DOM) [7, 43]. This tool is accomplished by using the XML parser to render the document into a structured format --- hierarchical tree structure (DOM Tree), which allows each element of the document to be accessed and manipulated by DOM provided APIs.

4.1 Case Study One: Automobile Manufacture Supply Chain Simulation

4.1.1 System Conceptual Model

The distributed automobile manufacture supply chain simulation system is a model representation of the real life process of a typical DSC. This system, which is called AutoSim Federation, consists of one automaker, the Car Assembly Factory where the cars are assembled, and four suppliers, the Tyre Factory, the Engine Factory, the Carbody Factory and the Lamp Factory which supply necessary parts to the Car Assembly Factory. The interaction relationship of the five semi-independent federates in the system is showed in Figure 4-1-1.

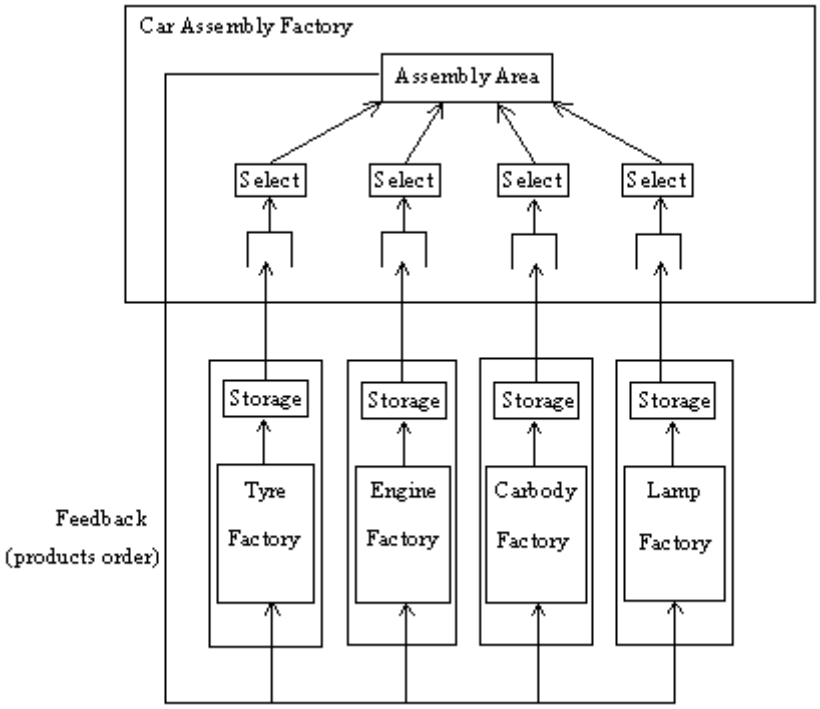

Figure 4-1-1: Models Interaction Relationship

These five simulation models (federates) run separately and cooperate through the exchange of objects (representative of product entities transferred between the models). The suppliers provide parts to the automaker based on Just-In-Time (JIT, which means getting the right parts to the right place at the right time) production theory [20, 27], so that the minimum inventory can be achieved.

The JIT production theory of manufacturing supply chain (also known as lean production or stockless manufacturing) is a management philosophy that strives to increase value added and eliminate sources of manufacturing waste by producing the necessary parts in necessary quantities at the necessary time. The benefits of JIT include improved delivery, low inventory levels, reduced operating costs, greater performance, higher quality and increased flexibility.

A key point of successful JIT is maintaining low inventory levels, which leads to faster reaction to customer's demands. Ideally, the supplier should produce a part just

before the part is needed by a customer. In conventional production processes, suppliers build products according to a self-ordained and pre-defined schedule. No consideration of customers' requirements is made. In small-batch production (JIT), customers encourage suppliers to deliver only what is needed by the assembly plant at a particular time, even if this means partially filled trucks [20]. Thus, products move rapidly through the suppliers' plant and to their customers, and suppliers maintain much less inventory.

The investigation of US and Japanese automakers by Jeffry K. Liker and Yen-Chun Wu has proved the efficiency of lean manufacturing. Johnson Controls Company is a good example. This company is famous for supplying seats for Toyota just hours before the seats are to be installed on the assembly line [20]. Its inventory levels dropped from 32 days[2] of inventory to 4.1 days after employing JIT and supply-chain logistics. And also, researches show that factories using JIT delivery system are not paying more for emergency delivery than the factories that use tradition MRP (Material Requirement Planning) system.

According to the JIT, in this case study, the Car Assembly Factory does not keep large inventory, it sends an order to the corresponding Component Factory when a certain kind of component is lacking and expects immediate supply. A late supply will result in delay of car production. To avoid this harmful condition, the Component Factory, upon receiving an order, must fulfill the order and deliver parts according to the demand as soon as possible. But, given that it takes time to machine the parts, buffer stock is required to keep the Car Assembly Factory from waiting. The problem that needs to be addressed is how fast each Component Factory should produce parts so that it can satisfy the requirement of the Car Assembly Factory and keep minimum

[2] Refers to how many days the current products in the inventory can satisfy the customer's requirement without new products being produced.

buffer stock at the same time. In this system, the Component Factories will adjust their production speed dynamically according to the order number and current buffer stock size. The pseudocode of Component Factory algorithm is shown as follows:

```
While simulation not terminated
    Waiting for next event
    When new order come from Car Factory, read the number of parts
    Required(PartsNeeded) and event time(timeMark) from order object
    If (timeMark<0)
        Terminate simulation;
        //assume negative timeMark is simulation terminate signal.
    endif
    Calculate produce capability of component factory between its
    simulation time(clock) and event time:
    produceCapability=(int)(timeMark-clock) / productionTime
    If (current produce capability is too high and products overstock:
       currentStoreSize+ produceCapability >=maxStoreSize)
       Produce parts and stop when store is full
        Adjust production speed by increase productionTime (decrease
        production speed)
       Advance simulation time to timemark
    else if (current produce capacity is appropriate:
           crntStoreSize+ produceCapability >=PartsNeeded)
           Produce parts until produceCapability is met
          Advance simulation time to timemark
         else (current produce capability is too low and order
             requirement cannot be satisfied)
            Produce parts until order requirement is met
            Advance simulation time by productionTime when every part
            is produced
           Adjust production speed by decrease productionTime
           (increase production speed)
    endif
    Deliver parts to Car Assembly Factory marked with current
    simulation time
 endwhile
```

To implement the automobile manufacture supply chain simulation within GRIDS environment, the first step is to decide the messages and objects that are produced and exchanged in the federation, and the object publish-subscribe relationships of the federates should also be confirmed. Then the object exchange models are specified using the OEMT standard so that they can be commonly understood by all the GRIDS federates. After that, the execution requirements of the federation are considered to determine which GRIDS thin agent services are needed to support the simulation. Certain documents are created to assist these services. Finally, the simulation system is developed, integrated and tested before executing the system to get the results. These

processes are detailed separately in the following sections.

4.1.2 Exchange Object Specification

In this simulation system, the federates keep sending and receiving objects between one another during their life time. The objects include the product objects (parts) transferred from the four Component Factories to the Car Assembly Factory, and the order object which the Car Assembly Factory sends to the Component Factories to notify the order demand of parts. The GRIDS OEMT is employed to specify these objects. The template has a certain set of information to be filled in. The basic methods and variables are made known in this template, so that other federates can access appropriately.

The tables 4-1-1 to 4-1-7 use tyrepackage and tyreorder object as examples to show the use of the OEMT version 1.0 in the system design. The tables 4-1-1 to 4-1-4 are the specification of the tyrepackage object. The tyrepackage object is published by the Tyre Factory Federate and is subscribed by the Car Assembly Factory. Each package encapsulates numbers of tyres. It can be regarded as a truck which delivers tyres to the Car Assembly Factory. The object exchange model specifies the attributes, methods and other information of the tyrepackage object.

Object Exchange Model Template (OEMT)	
Category	**Information**
Name of Model	tyrepackage
Model Description Link	Nil
Model Description	The tyrepackage represent the object transferred from the Tyre Factory to the Car Assembly Factory. Each package encapsulates numbers of tyres as components. We can regard it as a truck which delivers tyres to Car Assembly Factory.
Technical Details	……
General Details of each Implemented Object	……

Table 4-1-1: Exchange Object Information Table

Mandatory Object Attributes Table		
	Category	**Information**
1	Attribute Name	PackageSize
	Attribute Description	Stores the maximum number of components the package can carry.
	Attribute Accessibility	private
	Data Type	int
2	Attribute Name	CompType
	Attribute Description	Stores the type of components in the package.
	Attribute Accessibility	private
	Data Type	String
3	Attribute Name	TimeMark
	Attribute Description	Stores the time stamp of the package object
	Attribute Accessibility	private
	Data Type	Int

Table 4-1-2: Exchange Object Attributes Table

Mandatory Object Methods Table			
	Category	**Information**	
1	Method Name	insertCompNode	
	Method Description	insert a component Node into tyrepackage	
	Method Accessibility	public	
	Method Parameters	Parameter Name	CompNode
		Parameter Data Type	XML DOM Node
	Method Return Data Type	void	
2	Method Name	getCompNode	
	Method Description	get a component Node from tyrepackage	
	Method Accessibility	public	
	Method Return Data Type	XML DOM Node	
3	Method Name	setCompNodeAttribute	
	Method Description	Set attribute value from a component Node	
	Method Accessibility	public	
	Method Parameters	Parameter Name	CompNode
		Parameter Data Type	XML DOM Node
		Parameter Name	attrName
		Parameter Data Type	String
		Parameter Name	attrValue
		Parameter Data Type	String
	Method Return Data Type	void	
4	Method Name	getCompNodeAttribute	
	Method Description	extract String attribute value from a component Node	
	Method Accessibility	public	
	Method Parameters	Parameter Name	CompNode
		Parameter Data Type	XML DOM Node
		Parameter Name	attrName
		Parameter Data Type	String
	Method Return Data Type	String	
5	Method Name	setTimeMark	
	Method Description	set the timeMark attribute of tyrepackage	
	Method Accessibility	public	
	Method Parameters	Parameter Name	intTimeMark
		Parameter Data Type	int
	Method Return Data Type	void	
6	Method Name	getTimeMark	

	Method Description	get the time mark of the tyrepackage object	
	Method Accessibility	public	
	Method Return Data Type	int	
7	Method Name	setPackageSize	
	Method Description	set the maximum capacity of the tyrepackage	
	Method Accessibility	public	
	Method Parameters	Parameter Name	intSize
		Parameter Data Type	int
	Method Return Data Type	void	
8	Method Name	getPackageSize	
	Method Description	get the capacity of the tyrepackage	
	Method Accessibility	public	
	Method Return Data Type	int	
9	Method Name	setCompType	
	Method Description	set the compType attribute of tyrepackage	
	Method Accessibility	public	
	Method Parameters	Parameter Name	strType
		Parameter Data Type	String
	Method Return Data Type	void	
10	Method Name	getCompType	
	Method Description	get the compType attribute of the tyrepackage	
	Method Accessibility	public	
	Method Return Data Type	String	
11	Method Name	len	
	Method Description	get the number of components in the tyrepackage	
	Method Accessibility	public	
	Method Return Data Type	int	
12	Method Name	isFull	
	Method Description	return true if the number of components in this tyrepackage reaches the tyrepackage's capacity	
	Method Accessibility	public	
	Method Return Data Type	boolean	
13	Method Name	cleanPackage	
	Method Description	delete the componens in the tyrepackage	
	Method Accessibility	public	
	Method Return Data Type	void	

Table 4-1-3: Exchange Object Methods Table

General Details of Each Implemented Object		
Category	**Information**	
Company Name	NUS SOC	
Company Contact Info	(65) 68744366	
Company Contact Person	First Name	Na
	Last Name	Zhao
Company Email Address	Nil	
Version	1.1	
Date of Version	Day	30
	Month	12
	Year	2002
Implementation Platform	Java	
Reference Link to Implemented Object	Nil	

Table 4-1-4: General Details of Each Implemented Object

The Car Assembly Factory also subscribes to the enginepackage, carbodypackage, lamppackage objects from the other three Component Factories. The OEMT specifications for these three exchange objects in this system are similar with tyrepackage and omitted due to the limited space. The OEMT specified model of tyreorder object is given below. The tyreorder object with the other three objects --- engineorder, carbodyorder and lamporder are published by the Car Assembly Factory and are subscribed respectively by the Component Factories (The table of General Details of Each Implemented Object is omitted since it is the same with tyrepackage above.).

Object Exchange Model Template (OEMT)	
Category	**Information**
Name of Model	tyreorder
Model Description Link	Nil
Model Description	This object is published by Car Assembly Factory and is subscribed respectively by Tyre Factory.
Technical Details	……
General Details of each Implemented Object	……

Table 4-1-5: Exchange Object Information Table

Mandatory Object Attributes Table		
	Category	**Information**
1	Attribute Name	PartsNeeded
	Attribute Description	Stores the number of certain parts need by Car Factory
	Attribute Accessibility	private
	Data Type	int
2	Attribute Name	TimeMark
	Attribute Description	Stores the time stamp of the order object
	Attribute Accessibility	private
	Data Type	int

Table 4-1-6: Exchange Object Attributes Table

Mandatory Object Methods Table			
	Category	**Information**	
1	Method Name	setPartsNeeded	
	Method Description	set the number of parts need by Car Factory	
	Method Accessibility	public	
	Method Parameters	Parameter Name	intPartsNeeded
		Parameter Data Type	int
	Method Return Data Type	void	
2	Method Name	getPartsNeeded	
	Method Description	get the number of parts need by Car Factory	
	Method Accessibility	public	
	Method Return Data Type	int	
3	Method Name	setTimeMark	
	Method Description	set the time stamp of the object	
	Method Accessibility	public	
	Method Parameters	Parameter Name	intTimeMark
		Parameter Data Type	int
	Method Return Data Type	void	
4	Method Name	getTimeMark	
	Method Description	get the time stamp of the object	
	Method Accessibility	public	
	Method Return Data Type	int	

Table 4-1-7: Exchange Object Methods Table

The object specifications provide the uniform meaning of each exchange object, so that the federates can access the objects and extract required information from them through calling the methods. However, we also find that the current OEMT standard is not adequate to specify complex objects. So many necessary details are missed that the information provided by the object specifications is not enough for the federates to make an agreement based on them. We will summarize the experience in section 4.3, and offer an improved OEMT standard in next chapter.

4.1.3 Exchange Object Implementation

The most important issue in this case study is to implement object transfer and manipulation. We note that the OEMT is a Data Representation Model, that is, a data model about what the “things” will be like or how to represent “things”, but not a data model “of the things”. A mechanism is required to represent the “things” --- objects actually exchanged in supply chain simulation environment. In this case study, the java

object class is employed to implement the exchange object, which includes data and methods. It is the object/entity which represents the actual product.

However, another problem is how to organize the data, and how to differentiate them from methods. An object might have multiple hierarchical attributes. In addition, an object can carry more than one item, which has the same attribute name with different value. For example, in this case study, the tyre factory supplies tyres to the car assembly factory. As it is obviously inefficient to send tyre one by one, tyres are sent in bulk. Each tyre has different Uniform Tyre Quality Grade (UTQG), Maxload and so on which are presented as different values for certain attributes. So how to represent the attributes is a significant problem. To solve this problem, XML and the Document Object Model Tree (DOM Tree) are employed.

XML is a language portable over the Internet. It has opened the door to the sharing of information in various ways. It supports a variety of applications, allowing easy writing of programs to process XML documents based on the user-defined tags [25]. It can store information in various types of elements and show the relationship of the elements as classes and subclasses. This allows parallel object parts like carriages or tyres to be described as children of a single parent. XML provides an encapsulation mechanism for object attributes.

Using XML as the communication vehicle between disparate applications requires a mechanism that will read and interpret the XML document into a computer-friendly format. Application programs require a means to access the individual pieces of information (elements) contained within each XML document. This is accomplished by using the XML parser to render the document in a structured format --- hierarchical tree structure (DOM Tree). The Document Object Model (DOM) is a platform- and language-neutral interface that will allow programs and scripts to dynamically access

and update the content, structure and style of documents [7]. The DOM also offers a group of APIs to facilitate the access of the elements within the tree at run time. By following the tree (hierarchical) structure, the APIs allow traversing the tree freely, moving one part of the document tree to another without destroying and re-creating the content, and creating elements and attach them to any point in the document tree.

For these reasons above, DOM Tree is chosen in this project to specify the attribute field of object exchange model in the case study. Many tools have been built by different vendors for this purpose. What we used in our case study, the XML4J parser, is such a tool provided by IBM that renders an XML document in a DOM Tree.

Figure 4-1-2 gives the structure of the exchange object model.

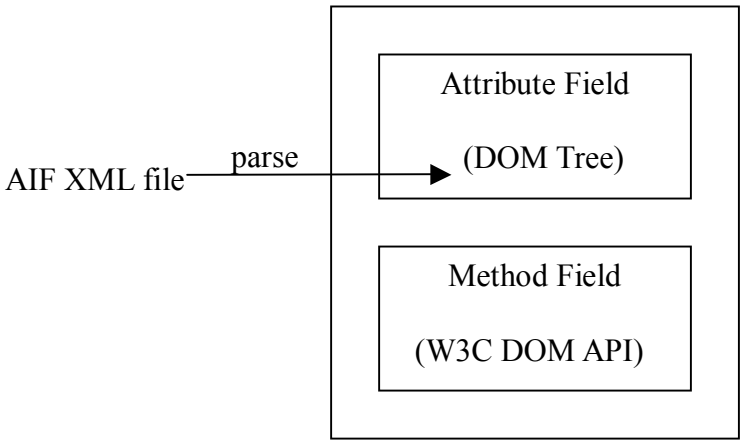

Figure 4-1-2: Exchange Object Model (Java object class)

- **Attribute Field of OEMT Object (AIF and DOM Tree)**

  The Object Exchange Model Template (OEMT) attribute information format (AIF) is a standard file format used to store the object attributes. The OEMT corresponds to common definitions of object models in object-oriented analysis and design (OOAD) techniques. In the OEMT literature, objects are defined as information encapsulations of attributes and operations (methods). This characteristic of the OEMT object requires all the attributes used to describe the

object be placed in an entire independent structure. The AIF is such a format to group all object attributes describing the object (e.g. attribute value). It stands for the attribute part of the OEMT. The AIF file takes the style of eXtensible Markup Language (XML) file.

- **Method field of OEMT Object**

  The set of methods is developed according to the description in OEMT **Mandatory Object Methods Table**. These methods can be invoked from outside the OEMT object to perform certain functions, facilitate the access to object's attributes (within DOM Tree) and provide other related services, such as providing some statistics, changing the state of the object, updating some of its attributes or acting on outside resources to which the object access.

As application OEMT in this case study, the AIF of tyrepackage object is parsed to the DOM Tree framework (Figure 4-1-3). The attributes described in the OEMT are converted to the DOM Tree. The Methods are based on W3C Document Object Model APIs that allow access to the elements within the DOM Tree. For example, when the automaker receives the TyrePackage object, it can get the package size or a single tyre through invoking the methods "**getPackageSize**" or "**getCompNodeAttribute**", which are described and acknowledged in the OEMT method table.

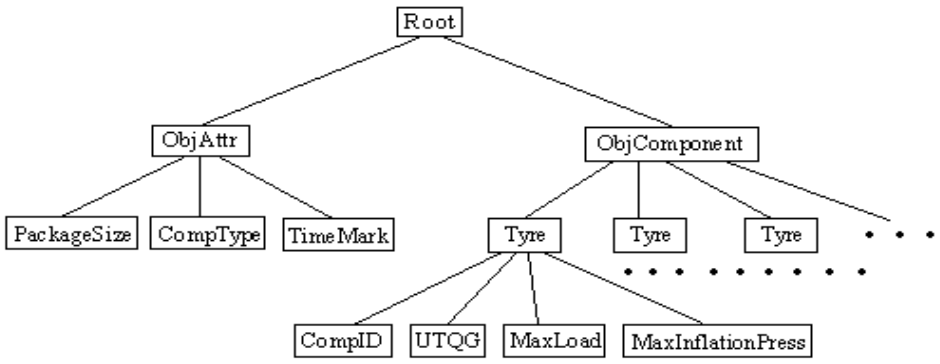

Figure 4-1-3: DOM Tree Framework of TyrePackage Object

4.1.4 Integration with GRIDS

To meet the interoperation requirement of the system, GRIDS is used as the distributed simulation middleware to integrate the federates together. It integrates seamlessly with Java's Object Serialization technology, enabling object-passing between remote federates. The timely transfer of objects between the elements of the automobile manufacture supply chain is the responsibility of GRIDS. To connect the GRIDS Client, the federates are required to realize two interfaces: SimInterface and SimStartInterface. A ".DDM" file is created for each federate to declare the publication or subscription of objects. Each federate keeps certain attributes such as netPort, federateName and a clock to record its current "time". These publication, subscription information and namespace will be used in the future to register to the Boot Server. Figure 4-1-4 is a UML Class Diagram to show the structure and details of Federate Classes.

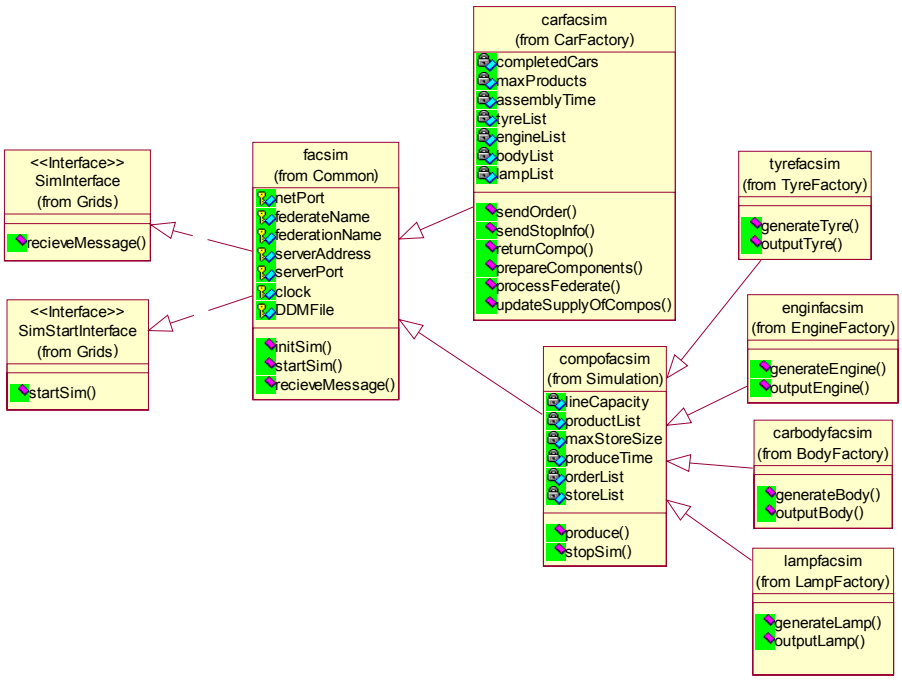

Figure 4-1-4: UML Structure of Federate Classes

Thin agents are employed to control the routes of objects transfer and synchronize the federates. Figure 4-1-5 shows the makeup of an individual federate [38].

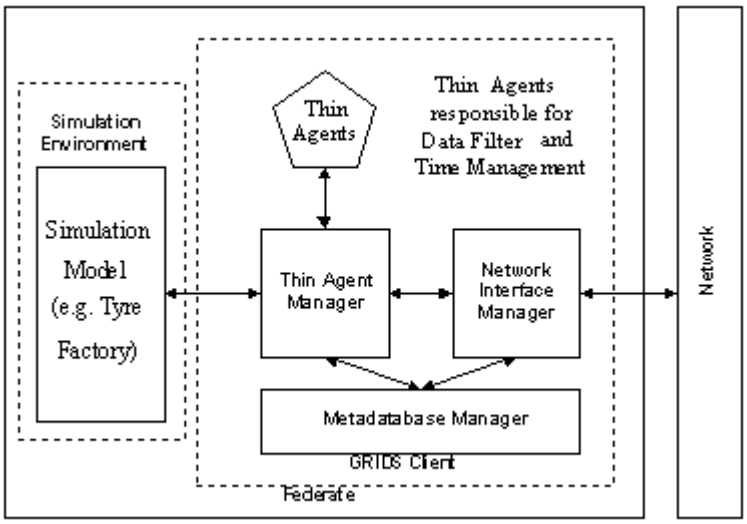

Figure 4-1-5: Federate Makeup with GRIDS Client

4.1.5 Execution and Result Analysis

Six PIII 700 MHz PCs with 256 MB RAM are used to run the whole system. One PC is used to run the GRIDS Boot Server; the other 5 PCs each carry one federate. We assume assembling a car needs two to three hours (random) in real life. The simulation terminates after 500 cars were produced in simulation time of 1356 hours and 18 minutes. The average car assembly time is about 2 hours and 43 minutes per car. The Car Assembly Factory spent 97 hours and 12 minutes on waiting for parts. Delay rate was 7.2%. This delay can be avoided by employing a safe stock in the Car Assembly Factory.

Figures 4-1-6 & 4-1-7 are two graphs that show the fluctuation of the stock size and the production time in the Carbody Factory and the Tyre Factory throughout the production process.

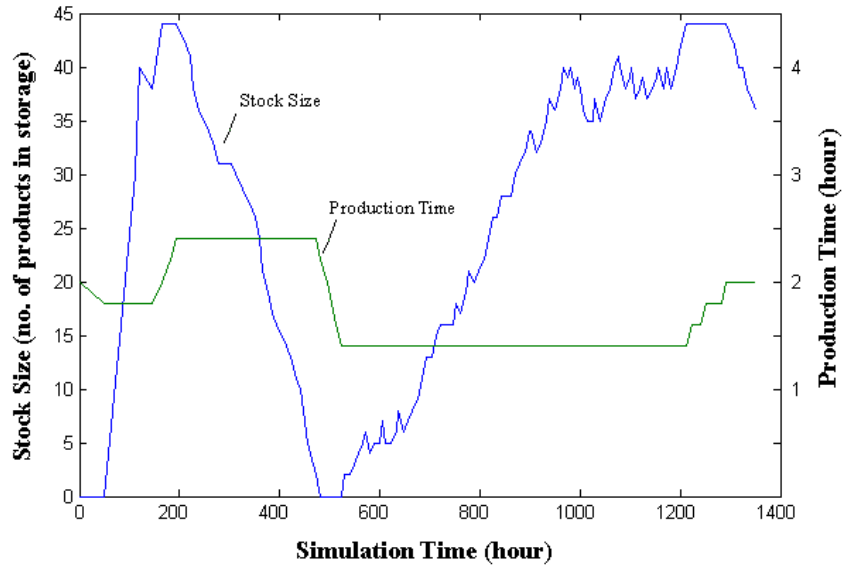

Figure 4-1-6: Carbody Factory Execution Result

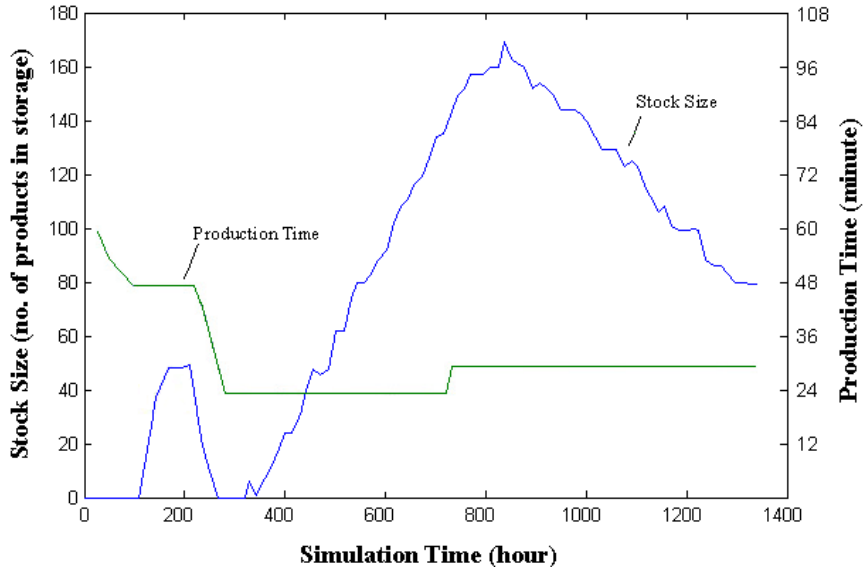

Figure 4-1-7: Tyre Factory Execution Result

During the production process, the Carbody Factory received 141 orders from the Car Assembly Factory when the Tyre Factory fulfilled 87 orders. This difference is due to the different number of parts required in the order by considering the volume of parts and capacity of delivery tools in real life. The production time is extended (production speed increase) by a given coefficient (different for each factory according real life experience) when the stock is close to maximum capacity and is decreased when the order demand cannot be satisfied. As we can see in figures 4-1-6 and 4-1-7,

the ability to automatically adjust production time makes the stock size fluctuate within a certain range. Other factors such as the random defect rate, order demand, different inventory capacity and so on also affect the wave period and form of the curve. Comparing the two figures above, the stock size of the Carbody Factory fluctuates faster than the Tyre Factory. The main reason is the inventory capacity of the Carbody Factory is much smaller than the Tyre Factory. The production time of the Tyre Factory is also more stable. (**Note**: the exceptional drop of stock size in Tyre Factory around time 300 is due to the fact that the defect rate of previous batch of tyres is abnormally high. This conclusion is made via analysis of factory's production record.)

4.2 Case Study Two: Singapore Mass Rapid Transit (MRT) System Simulation

The MRT system in Singapore is one of the most important public transport systems in the country. It spreads all over the country and provides mass rapid transit passenger service along major high-density travel corridors in Singapore.

The second case study investigates the ability of GRIDS and OEMT to facilitate transportation simulation. This case study designs a distributed system to simulate the MRT system and uses this to investigate the flow of passengers and trains on the Singapore MRT system, specifically to investigate the relationship of the model to the safe capacity of a MRT station.

Sections 4.2.1, 4.2.2 and 4.2.3 will discuss the design issues of the case study including building objects based on the Object Exchange Model Template. Then the case study implementation will be presented in section 4.2.4. Section 4.2.5 presents the execution and experience analysis.

4.2.1 System Design and Function Description

To simplify our system, only major stations, eight stations and four garage models, are simulated in a four line MRT system. This means the MRT federation, named **MRTSim**, consists of twelve federates. The station simulation models are object-oriented and distributed running on network computers. Train objects carrying passengers are transferred between stations. They are based on the OEMT as the information transfer standard and apply OEMT standard on full scale. GRIDS is used as simulation middleware to connect the federates. This case study therefore gives another practical instance of GRIDS and the OEMT application.

Figure 4-2-1 illustrates the federation structure graphically. The function of each station is briefly described as follow.

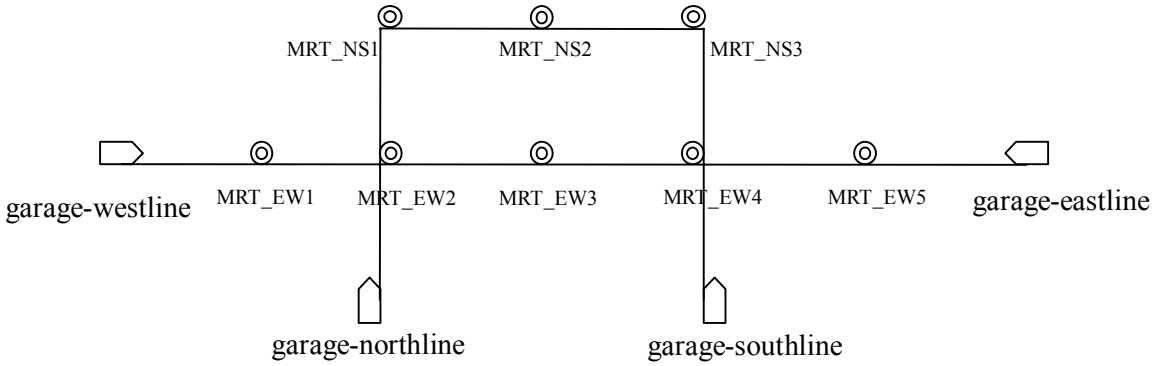

Figure 4-2-1: MRT Simulation System Structure

The garages are in charge of generating empty trains (train objects with initial attribute values) for each MRT line in a fixed interval. Each of these MRT stations has a maximum safe capacity. Passengers reach a departure station at a given rate, which could be changed for each station and/or for different time periods. Each passenger has a randomly assigned destination and takes the best route (normally the shortest way to destination). The passengers are put into a waiting queue of a particular platform according to their destinations.

A train object carries information such as trainID, capacity, total passenger number, number of passengers to each station and so on. This information forms the attributes of the OEMT object, which are defined in the following specification. When a train arrives at a station, the station model invokes the methods of train object to access the attributes and finish the operations such as passengers getting on and off this train, and then sends the train to the next station. If the number of passengers on a station exceeds the safe capacity, the federation terminates by producing warning information together with a summary of statistical data. The data are used to adjust train interval or train capacity so that the stations could keep within the safe capacity.

Figure 4-2-2 takes station EW2 as an example to portray the exchange of train objects in the system. As it can be seen, federate EW2 exchanges object with other 4 federates. It receives train objects from 4 federates, and sends objects to 3. For example, a west line train comes to EW2 from EW1; EW2 processes the train (updates train object, including update timestamp, insert or delete passengers and other statistics work), adjusts station's simulation time and then sends the object to the next station (EW3). The behavior of other federates is similar to EW2.

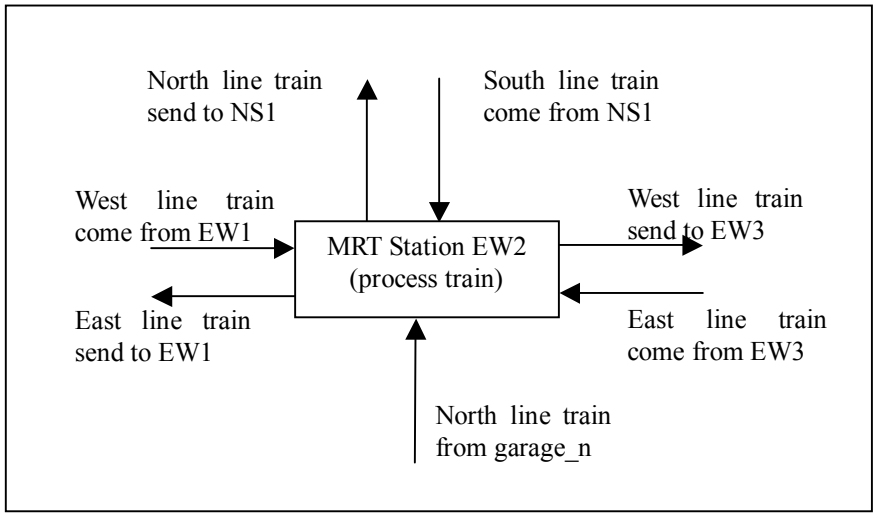

Figure 4-2-2: MRT Station Object Exchange Relationship

4.2.2 Exchange Object Specification

The motivation of this case study was to show the workings of the OEMT. We wanted to know if it is a suitable and efficient way to specify object exchange in the transportation system. So our main concern is on the design of the object exchange model in the MRT Simulation based on the OEMT format.

There are four train objects --- EastLineTrain, WestLineTrain, NorthLineTrain and SouthLineTrain that represent the trains come from different lines (refer to Figure 4-2-2). In the following four tables we use EastLineTrain as an instance to provide the specifications of exchange object in MRTSim using OEMT. The specifications include all the attributes of the MRT train object. The methods to access attributes are also represented. The real exchange object will be developed based on these descriptions.

Object Exchange Model Template (OEMT)	
Category	**Information**
Name of Model	EastLineTrain
Model Description Link	Nil
Model Description	A model to emulate a train which carries numbers of passengers from current station to their destination station. The important features are the number of passenger going to each station and the capacity of the train.
Technical Details	……
General Details of each Implemented Object	……

Table 4-2-1: Exchange Object Information Table

Mandatory Object Attributes Table		
	Object Attributes Field	**Data**
1	Attribute Name	capacity
	Attribute Description	Stores the max number of passenger which can be carried by the train
	Attribute Accessibility	private
	Data Type	int
2	Attribute Name	totalpsgnum
	Attribute Description	Stores current number of passenger on the train
	Attribute Accessibility	private
	Data Type	int
3	Attribute Name	psgnum_to_EW1
	Attribute Description	Stores passenger number whose destination is station EW1.
	Attribute Accessibility	private
	Data Type	int
4	Attribute Name	psgnum_to_EW2
	……	……

Table 4-2-2: Exchange Object Attributes Table

Mandatory Object Methods Table			
	Object Attributes Field	**Data**	
1	Method Name	setAttrValueInt	
	Method Description	set attribute value to each attribute in the object.	
	Method Accessibility	public	
	Method Parameters	Parameter Name	AttrName
		Parameter Data Type	String
		Parameter Name	AttrValue
		Parameter Data Type	int
	Method Return Data Type	void	
2	Method Name	extractIntValue	
	Method Description	extract attribute value from object	
	Method Accessibility	public	
	Method Parameters	Parameter Name	AttrName
		Parameter Data Type	String
	Method Return Data Type	int	
3	……	……	

Table 4-2-3: Exchange Object Methods Table

General Details of Each Implemented Object		
Category	**Information**	
Company Name	NUS SOC	
Company Contact Info	(65) 68744366	
Company Contact Person	First Name	Na
	Last Name	Zhao
Company Email Address	Nil	
Version	1.1	
Date of Version	Day	30
	Month	8
	Year	2002
Implementation Platform	Java	
Reference Link to Implemented Object	Nil	

Table 4-2-4: General Details of Exchange Object

4.2.3 Exchange Object Implementation

In this case study, the train objects take the form of Java object classes which included the DOM Tree as data member and multiple methods. Figure 4-2-3 is the AIF file being used to carry attributes of the train object. It is parsed into DOM Tree and stored in the object class in the MRTSim program.

```
<?xml version="1.0"?>
<objectModel>
  <modelName>EastLineTrain</modelName>
  <description>Singapore MRT simulation</description>
  <technicalDetails>
    <objAttr>
      <capacity></capacity>
      <totalpsgnum></totalpsgnum>
      <psgnum_to_EW1></psgnum_to_EW1>
      <psgnum_to_EW2></psgnum_to_EW2>
      <psgnum_to_EW3></psgnum_to_EW3>
      <psgnum_to_EW4></psgnum_to_EW4>
      <psgnum_to_EW5></psgnum_to_EW5>
      <psgnum_to_NS1></psgnum_to_NS1>
      <psgnum_to_NS2></psgnum_to_NS2>
      <psgnum_to_NS3></psgnum_to_NS3>
    </objAttr>
  </technicalDetails>
</objectModel>
```

Figure 4-2-3: AIF File for Train Object in MRTSim

Through using the DOM APIs in this case study, any element within the AIF XML document can be accessed, changed, deleted, or added. The value of each attribute can be accessed by invoking the methods of the object. Figure 4-2-5 gives an example of the methods in the train object. Figure 4-2-4 shows the original state of part of the attributes in the train object. The method "**setAttrValueInt**" functions as finding specified element by name and setting the value passed as parameters.

```
<objAttr>
  <capacity></capacity>
  <totalpsgnum></totalpsgnum>
 <psgnum_to_EW1></psgnum_to_EW1>
</objAttr>
```

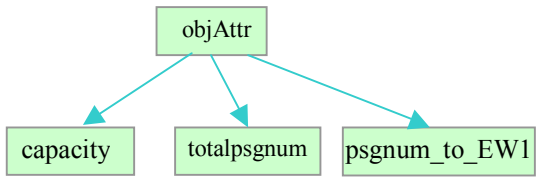

Figure 4-2-4: Example DOM Tree Fragment

```
public void setAttrValueInt(String AttrName,int AttrValue)
 {
   String AttrV_=Integer.toString(AttrValue);
   NodeList Elements=this.doc.getElementsByTagName(AttrName);
   Node refernode=elements.item(0);
   //creat a new node with particular value, and
    append it under element node.
   Node new_node=this.doc.createTextNode(AttrV_);
   refernode.appendChild(new_node);
}
```

Figure 4-2-5: Method setAttrValueInt(String AttrName, Int AttrValue)

By calling the method, a user can add value to given attribute of the object. For example:

```
setAttrValueInt("capacity",500);

setAttrValueInt("totalpsgnum",300);

setAttrValueInt("psgnum_to_EW1",50);
```

Figure 4-2-6 shows the result of methods invocation from outside the object.

```
<objAttr>
  <direction>1</direction>
  <capacity>500</capacity>
  <totalpsgnum>300</totalpsgnum>
</objAttr>
```

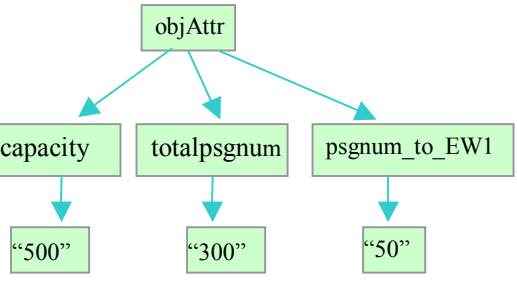

Figure 4-2-6: Example DOM Tree Fragment after Calling Method

As Figure 4-2-6 showed, each time the method is called, a new text node with attribute value (parameter two) is created and appended as child node of the corresponding element node (refer to parameter one, attribute name). Hence the attribute field is updated.

Other methods of train object behaved similarly as the above example. The train object with all the attributes and methods was transferred among the federation in the MRTSim. The program detail of the train object was transparent to users. The method name, parameters and function were listed in the OEMT specification. Users can search the specification for the required operation. Through calling these methods, a federate (station) can obtain useful information (e.g. passenger number in current station) from the exchange object, update it (e.g. passengers get off or on the train), then transfer the object to the next federate. Providing methods in the OEMT can standardize object input, output and information transmission in simulation, thus simplifying simulation builder's work dramatically and promoting re-use of simulation components.

4.2.4 Case Study Implementation in GRIDS Environment

The experimental platform of the case study is outlined below:

- GRIDS was used as the middleware in this case study.
- All the 12 federates were written in Java to take advantage of its object-orientedness, multithreading, and also the strong object serialization technology.
- The train object was implemented on the format of a Java object class.
- Seven *PIII 700 MHz* with *256 MB RAM* PCs within a 7-PC cluster were used to run the whole system. One PC was used to run the GRIDS Boot Server; the other 6 PCs each carried two federates.

The stations are discrete-event based, in which a train object is regarded as an incoming event. The timely transfer of objects between federates was the responsibility of this middleware. Time management, data distributed management and other basic services are achieved by employing the corresponding GRIDS thin agents. The makeup of an individual federate with the GRIDS client is similar with case study one (refer to Figure 4-1-5). Figure 4-2-7 illustrates the structure of the MRT federation in GRIDS environment.

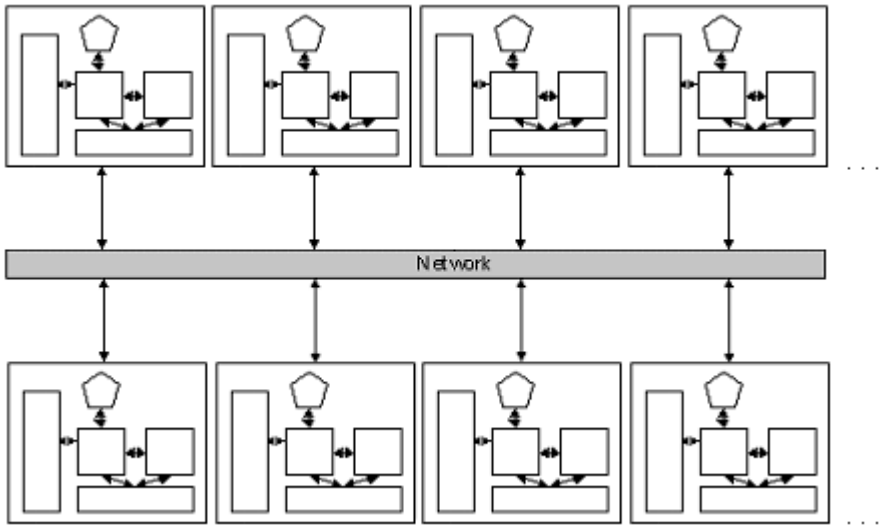

Figure 4-2-7: MRT Federation Structure in GRIDS Environment

4.2.5 Execution and Experience Analysis

As stated in the beginning of this chapter, the setup of the nodes in the second case study differs with the first one. In the first case study, the component factories and the assembly factory are tightly coupled. The component factories cannot send parts to assembly factory until they receive orders. They fall into sleep to avoid the overflow of their stores and are awaked by the order objects. On the other hand, the assembly factory would also sleep if the component factories could not meet its requirement. However, in the MRT system, the stations are loosely coupled. It means that there is no

immediate or direct feedback when an object/entity is passed between the stations. A station works in a simple internal logic according to a self-ordained and pre-defined schedule.

Conventionally, the synchronization of systems like the MRT can be achieved by a "spreadsheet solution", which is simply taking the results of the first model in a spreadsheet and then uploading it into the second model. However, while spreadsheet solution is a rudimentary approach to time management, it is not a scalable one. To handle logically connected models, it often requires some assumptions such as zero idle time. We need a generic distributed simulation approach without such limitations to handle systems with loosely coupled nodes and implement complex business logic in them.

The second case study investigates the ability of GRIDS to be such a generic solution. To be generic, it must be able to transfer timestamped entities from one model to another; allow a model to correctly receive and process the timestamped entities from one or more models; and correctly synchronize these models.

Through the MRT simulation, we find that GRIDS can meet these basic requirements. It transferred and processed the timestamped train objects in correct order. And it coordinated and synchronized the MRT system well. But, since GRIDS was originally developed based on the DSC requirements, its ability to handle transportation system is limited. The algorithm of the current GRIDS TM thin agent is not powerful enough to meet the requirement of the transportation system simulation. And because the task of each station is not balanced, some station may have a big waiting queue after executing a period of time. To maintain the queue, TM thin agent consumes large amounts of memory. For this restriction, the MRT simulation did not run stably. We fail to give exact results in this case study. To improve the thin agent is

beyond the scope of this research work and will be left to future work. However, we still gain experience from it.

First of all, this case study proves valuable experience to improve the structure of the OEMT. Although the OEMT successfully describes the basic properties of the train in the MRT simulation, its elements are not enough to specify objects effectively in transportation system. This case study also proves that the OEMT helps the DDM thin agent work well in this simulation.

Second, we find that object passing is efficient in transportation system. A single train travels through several stations, therefore the ability to carry methods is promising in enhancing the efficiency of the system.

Third, through this case study, we find the deficiency of GRIDS in handling transportation system and indicate a direction in further improvement for GRIDS.

The above experience will be summarized in section 4.3. The goal of our future work will be making GRIDS an efficient, convenient and all-purpose simulation middleware.

4.3 Experience Gained from Case Studies

The case studies helped to test, analyze, and verify GRIDS infrastructure and the OEMT. The experience gained from case studies will be summarized in this section. Certain opinions and suggestions will be given to help the improvement of GRIDS and the OEMT.

4.3.1 GRIDS Federation Development Process

Through the experience we got from the development of the automobile supply chain simulation, the GRIDS federation development process can be described in six steps as follows:

1. Define federation objectives;
2. Develop federation conceptual model;
3. Design federation (particular OEMT object specification development);
4. Develop federation;
5. Integrate and test federation;
6. Execute federation and results analysis.

This series of activities is necessary to design and build the GRIDS federation. The six steps need not be performed in a strictly sequential manner. A spiral development approach[3] might be more effective.

4.3.2 Benefits of GRIDS

GRIDS has several advantages which make it an ideal middleware infrastructure for DSC simulations. These advantages also make GRIDS suit other application areas such as transportation as shown in case study two:

- It supports high extensibility in several levels from user-defined message types to functionality extensions via the thin agents and internal data storage extensions via the MetaDataBase (MDB) interface. Functionality extensibility via thin agents is the key feature of GRIDS. This property allows additional services required for a DSC or other types of simulation to be easily and rapidly developed and dynamically added in. The environment's functionality is also enhanced through the extension mechanism in response to specific requirements made by the vast distributed simulation areas.
- It reduces the time and costs for building a new simulation model, which is derived from the GRIDS's ability to integrate an array of existing

[3] Spiral development is a family of software development processes characterized by repeatedly iterating a set of elemental development processes and managing risk so it is actively being reduced.

component-based simulation models. It also takes advantage of the functionality of various vendors' simulation products.

- It is a light-weight and portable middleware and converting an existing application to use GRIDS is comparatively easy. The application interface within GRIDS is in the form of two basic object-oriented interfaces that must be implemented during the development of an application: the SimInterface (equivalent to the federate ambassador) and the GridsInterface (equivalent to the RTI ambassador).
- It supports peer-to-peer communication between federates. The traffic bottleneck is avoided because there is no central server that handles communications.
- The architecture is easy to understand and hence, easy to learn and grasp.
- The fault tolerance level is high because an error in single node does not affect the execution of other nodes.

4.3.3 Deficiencies of GRIDS

Although the advantages of GRIDS are aplenty, its deficiencies cannot be ignored. The deficiencies of GRIDS exposed in the case studies are listed below:

- GRIDS lacks the ability for dynamically federates discovering, joining in and withdrawing at runtime.
- Inadequate integration mechanisms are provided by GRIDS. An interface which maps different program languages to the middleware is in need.
- Thin agent services in GRIDS are inadequate. More services are expected such as secrecy, ownership management, execution data collection and maintenance etc.
- Current Data Distributed Management and Time Management Thin agents provides single algorithm. This limitation seriously restricts the function of

federations. More algorithms are needed to meet simulation requirements flexibly.

- The ability of the Time Management Thin agent to synchronize simulation models is limited. This limitation seriously restricts the function of federations.
- Since valid federates cannot ensure a valid integrated federation, verifying and validating GRIDS federation is hard and still need further research.

These experience and suggestions will greatly help the improvement of GRIDS.

4.3.4 OEMT Evaluation

Through the two case studies, we approve that the OEMT is suitable for specifying object exchange models for the following reasons:

- It provides a valid template for documenting object models which emulate as close to reality as possible, the actual entities passed between the simulation federates. This common template provides a standardized way of specifying models to be used as input or output, and facilitates understanding between federates of a distributed simulation.
- It is particularly useful for implementing Data Distribution Management (DDM) for distributed simulations in GRIDS, which reduces the network latency by filtering the data and sending output objects only to federates that needs them. It provides the GRIDS Boot Server a standard to distribute object information to the DDM Thin Agents of the various federates. Based on the OEMT, the DDM Thin Agent places the correct object on the relevant output lines, and differentiates objects coming in from the various input lines.
- It separates the manipulation of exchange object from simulation models. Therefore facilitate the design and reuse of common tool sets for development of distributed simulation object models.

- It provides Mandatory Object Methods Table to describe the methods to access the content of the object.

However, we also found the deficiencies in the practical applications of the OEMT from the case studies. Although the template has been set up, it is still not complete. It needs to be improved to describe the object model completely.

1. Lack of the capability to specify the components of the object model.

Sometimes an object is not atomic, it can contain more than one component of the same type or even of more than one component type. For example, in case study one, the tyrepackage object is not atomic. Each tyrepackage encapsulates a number of tyres. It can be regarded as a truck which delivers tyres to the Car Assembly Factory. A tyre is the component of tyrepackage object. It has attributes such as UTQG, Max Load and Max Inflation Press. However, the OEMT version 1.0 has no such element to specify the attributes of object component. The OEMT needs to be extended to describe the special attributes of each component type.

Table 4-3-1 is an optional table introduced into the OEMT by this research. It is used to describe component (tyre) of the object (tyrepackage).

Exchange Object Component Table			
Category	**Information**		
Component Name	Tyre		
Component Description	Tyre is the component in Tyre package.		
Component Detail	1	Component Attribute Name	CompID
		Attribute Description	Index of individal tyre
		Attribute Accessibility	Private
		Data Type	int
	2	Component Attribute Name	UTQG
		Attribute Description	The Uniform Tyre Quality Grade (UTQG) □rained□g system is a rating for treadwear, traction, and temperature resistance.
		Attribute Accessibility	private
		Data Type	String
	3	Component Attribute Name	MaxLoad
		Attribute Description	Maximum load
		Attribute Accessibility	private
		Data Type	float
	4	Component Attribute Name	MaxInflationPress
		Attribute Description	Maximum inflation press
		Attribute Accessibility	private
		Data Type	float

Table 4-3-1: Exchange Object Component Table

2. Lack of the capability to define timestamp and lookahead which are necessary in GRIDS environment.

Because the Time Management thin agent of GRIDS used conservative algorithm [2, 3] to synchronize all the federates, each federate should have its timestamp and lookahead. However the current OEMT cannot describe the data format and scale of timestamp and lookahead. A table like 4-3-2 is needed in both the case studies.

Object Time Representation Table		
Category	**Datatype**	**Scale (time unit : simulation time)**
Time stamp	Integer (non-negative)	1:6minute
Lookahead	Integer (non-negative)	1:6minute

Table 4-3-2: Object Time Representation Table

3. Lack of the ability to define the inheritance relationship between objects.

In the second case study, there are four types of object models that represent trains from north line, south line, west line and east line. These objects have almost the same attributes and methods. However, using the current OEMT, we had to specify the four similar objects separately. This means we had to describe same attributes and methods four times. The duplicated work can be avoided if we introduce inheritance relationship into the OEMT.

The Exchange Object Information table can be extended with two more items as shown in table 4-3-3 (bold and italic). They are set up to cater for the current trend of object-oriented programming. One of the most important properties of object-oriented program design is inheritance, by which a child object inherits and uses attributes and methods from its parent object and all of its ancestors or it can hide the member variables or override the methods. Introducing inheritance into the OEMT simplifies the description of children objects by starting from the definition of existing objects.

Object Exchange Model Template (OEMT)	
Category	**Information**
Name of Model	MRT Train
Model Description Link	Nil
Model Description	A model to emulate a train which carries numbers of passengers from current station to their destination station. The important features are the number of passenger going to each station and the capacity of the train.
Name of Parent Object	Nil
Names of Children Objects	East Line Train
	West Line Train
	North Line Train
	South Line Train
Technical Details	……
Time Representation	……
Attributes Field Data Type	DOM Tree
General Details of each Implemented Object	……

Table 4-3-3: Exchange Object Information Table

As shown in the table, in case study two, the trains from four directions are all the

children of a single parent object (MRT train object). The children objects inherit all the attributes and methods from its parent object. They can also extend new attributes and methods for themselves with only the new attributes or methods being presented. This new function can also be used into case study one to simplify the specification of the package objects and the order objects.

4. Need to add Attribute Field Data Type item.

As it was mentioned in section 4.1, the OEMT implementation involves an independent attribute field of object. This item in Table 4-3-3 (italic) is used to specify the data type of attribute field that carries the object attribute information. It can be array, list, file or even more complex data structure. For example in case study two, the attribute field of train object takes the form of DOM Tree.

In addition, the object class may be a large structure. Sometimes transferring the whole object can cause a heavy load on the network. However, the object instances of uniform object model are different only by the values of attribute members and they have the same method members. The method members become redundant in the transfer process. To improve the efficiency of simulation system, one consideration is to transfer the attribute field without the method members of the object class. In this case, after the first introduction of the object class to the subscriber, the object class model resides in the subscriber federate. Only the attribute field is transferred between federates instead of the whole object. An independent attribute field provides a mechanism to extract the attribute information as a transferable entity from the object.

Based on these deficiencies we found, we update the OEMT with new elements. The standard format of each new element above is defined in chapter 5 when we provide a more robust and complete OEMT standard (version 2.0). The OEMT DIF specifications for the tyrepackage and MRT train objects using the OEMT version 2.0

can be found in appendix B and C.

From the case studies, we also find that the resource of existing OEMT in OEMR is far from enough for efficient reuse and development of the OEMT DIF files is truly time-consuming. So we develop an application tool kit which facilitate the development of DIF file and provides the OEMR on-line service in chapter 5.

Chapter 5 OEMT Enhancements

The goal of this research work was to investigate problems involved in the development of a distributed simulation in GRIDS environment, specifically the object-passing mechanism and the standardization of the exchange objects.

From the two case studies we can see that the basic elements of the OEMT are definitely not sufficient to cater to even the needs of the specialized DSC Simulation saying nothing of other fields. Hence, there was a need to upgrade the OEMT and identify as complete as possible, the additional requirements of distributed simulation, especially DSC simulation that have to be provided by the OEMT.

The OEMT currently includes 5 elements. This is clearly inadequate to represent the complexity of the object involved in the various simulation applications. This research work contributes both in extending OEMT elements and in providing a new tool in section 5.2 to aid the building of object model.

5.1 OEMT Elements Evolvement (version 2.0)

In the original release of the OEMT in June 2001, the format definition was entirely driven by the experiments of some simple GRIDS prototype federations, which focused on DSC simulation area. Subsequent to this original release, the OEMT has continued to mature and evolve, primarily based on the research on complex simulation systems both for DSC application and other simulation types.

In this section, a fine-tuned version of the OEMT will be provided based on the experience collected from the case studies.

The OEMT evolved elements is presented as below:

GRIDS object exchange models are composed of a group of inter-related elements specifying information about the model. Each model is identified uniquely by the mandatory attributes and methods that must be implemented for each implemented object of the model. And there are also certain optional elements for users to select according to the requirements of different simulations. The template for the core of a GRIDS object exchange model uses a tabular format and shall consist of the following elements. The four main elements and one sub-element supplemented by this research project are in bold:

- Name of Model: to record the product name that the model is emulating.
- Model Description Link: to record the URL link if the description of the model is located somewhere else on the internet.
- Model Description: to record a detailed description of the purpose of this model and the product description.
- **Name of Parent Object:** to record the name of parent object of the current model.
- **Names of Children Objects:** to record the names of children objects of the current model.
- Technical Details: to specify the object attributes that must be implemented for this model, **object components** and mandatory methods.
- **Time Representation**: to specify the timestamp and lookahead of the instances of the object.

- **Attribute Field Format Representation:** to specify the data structure of object attributes used in object model implementation.
- General Details of each Implemented Object: to specify for each object that has been implemented using this model, the details of the company and the location of the object in the form of an URL.

The tabular format of basic OEMT is showed as follow, from Table 5-1 to 5-6. Detailed content of each field is also defined.

Object Exchange Model Template (OEMT)	
Category	**Information**
Name of Model	
Model Description Link	
Model Description	
Name of Parent Object	
Names of Children Objects	
Technical Details	
Time Representation	
Attributes Field Data Type	
General Details of Each Implemented Object	

Table 5-1: Exchange Object Information Table

5.1.1 Name of Model

This field records the product name that the model is emulating. The name of the model shall be clear and unique, to clearly differentiate this model from the others in the OEMR.

5.1.2 Model Description Link

This is a URL link and shall only be used only when the Model Description and Technical Details of the model is located away from the OEMR. “Model Description Link” shall not co-exist with “Model Description”
and “Technical Details”. Should “Model Description Link” be used, “Model Description” and “Technical Details” shall have an entry of “Nil” and vice versa.

5.1.3 Model Description

This field shall record the purpose for which the model was created, and the description of the model. It may also contain a brief description of key features of this model for easy understanding between simulation builders.

5.1.4 Name of Parent Object and Names of Children Objects

These two elements are optional. In the OEMT standard, each object can have maximum one parent object but several children objects. The names of all children objects are listed. Each name shall be clear and unique. The children object inherits all the attributes and methods from its parent object. Only the new or overlaid attributes and methods in the children object are described in its models. The inheritance ability meets the current trend of object-oriented programming and facilitates the development and reuse of object exchange model.

5.1.5 Technical Details

This field is used to describe the technical details of this model. It shall further be sub-divided into three tables, the Mandatory Object Attributes Table, the Object Components Table and the Mandatory Object Methods Table. This field shall document all necessary attributes and methods that must be implemented for each object implemented using this model. At least one mandatory attribute must be defined, in

accordance to making the model unique. For mandatory methods, it is optional in the model's declaration. And the components table is also optional. If the object is atomic which has no components (e.g. tyre), this table shall be ignored.

For data types that are permitted, only primitive data types are currently allowed for each attribute and for the return data type of methods in this version of the OEMT specifications.

The following are the permitted data types for attributes and return data of methods:

- ✓ boolean
- ✓ char
- ✓ double
- ✓ float
- ✓ int
- ✓ string or char arrays
- ✓ arrays of primitive data types

Mandatory Exchange Object Attributes Table

The attributes mandatory to the object exchange model shall be recorded in a tabular format. For each attribute, the following items must be recorded:

- Attribute Name: Name of the attribute. The data shall be clear and shall be store in a string format and shall be able to define the attribute uniquely within this object model.
- Attribute Description: Describes the purpose and use of this attribute. The unit of the attribute value should also be recorded in this field if needed.
- Attribute Accessibility: Describes the accessibility (scope) of this attribute. This field accepts only three entries – Private, Public or Protected. The meaning of

these three entries is equivalent to the usage of these entries in Object Oriented Programming.

- Data Type: As was described above, primitive data types only are allowed currently.

Mandatory Object Attributes Table		
	Object Attributes Field	**Data**
1	Attribute Name	
	Attribute Description	
	Attribute Accessibility	
	Data Type	
2	Attribute Name	
	…	

Table 5-2: Mandatory Object Attributes Table

Object Components Table

If the object exchange model has certain component-objects (e.g. in case study one, the tyre is the component-object of tyrepackage object model), the component-objects shall be recorded in a tabular format. For each component-object type, the following items must be recorded:

- Component Name: Name of the component-object. The data shall be stored in a string format, and shall be able to define the component-object uniquely within this object model.
- Component Description: Describes the purpose of this component and includes a brief description of its key features.
- Component Detail: Describes the attributes of the component.

- Attribute Name: Name of the attribute. The data shall be clear and shall be store in a string format and shall be able to define the attribute uniquely within this object model.
- Attribute Description: Describes the purpose and use of this attribute. The unit of the attribute value should also be recorded in this field if needed.
- Attribute Accessibility: Describes the accessibility (scope) of this attribute. This field accepts only three entries – Private, Public or Protected. The meaning of these three entries is equivalent to the usage of these entries in Object Oriented Programming.
- Data Type: As was described above, primitive data types only are allowed currently.

Exchange Object Component Table			
Category	**Information**		
Component Name			
Component Description			
Component Detail	1	Component Attribute Name	
		Attribute Description	
		Attribute Accessibility	
		Data Type	
	2	Component Attribute Name	
		…	

Table 5-3: Exchange Object Component Table

Mandatory Object Methods Table

The methods mandatory to the object exchange model shall be recorded in a tabular format. These methods are used to access attributes of both the object and its component-object, and they also provide other services such as statistics and so on. For

each method, the following items must be recorded:

- Method Name: Name of the method. The data shall be stored in a lower-case string format, and shall be able to define the method uniquely within this object model.
- Method Description: Describes the purpose and use of this method.
- Method Accessibility: Describes the accessibility (scope) of this method. This field accepts only three entries – Private, Public or Protected. The meaning of these three entries is equivalent to the usage of these entries in Object Oriented Programming.
- Method Parameters: Stores the information about the parameters that are required for this method in order to execute the method. This field is optional. A method may not require any parameters. This field is further broken down into the following items:
  - Parameter Name: Name of the parameter. The data shall be stored in a lower-case format, and shall be able to define the method uniquely within this method.
  - Parameter Data Type: As was described at the beginning of this section, primitive data types only are allowed currently. This data types follow the permitted data types of attribute data types.
- Method Return Data Type: Stores the return data type that will be output as a result of executing this method. As was described at the beginning of this section, primitive data types only are allowed currently.

Mandatory Object Methods Table			
	Object Methods Field	**Data**	
1	Method Name		
	Method Description		
	Method Accessibility		
	Method Parameters	Parameter Name	
		Parameter Data Type	
	Method Return Data Type		
2	Method Name		
	Method Description		
	Method Accessibility		
	Method Parameters		
	Method Return Data Type		
3	……		

Table 5-4: Mandatory Object Methods Table

5.1.6 Time Representation

This field specifies the timestamp and lookahead of the object, which is used to synchronize all the federates in a distributed federation. The object arrived is processed in time order. The datatype for the timestamp and lookahead must be recorded. The scale presenting the proportion of the time unit to the simulation time it presented shall also be recorded.

Object Time Representation Table		
Category	**Datatype**	**Scale (time unit : simulation time)**
Timestamp		
Lookahead		

Table 5-5: Object Time Representation Table

For example, the data type may be “Integer (non-negative)” and scale being set “1:1minute” (see example in section 4.3).

5.1.7 Attribute Field Format Representation

This item is used to specify the data type of attribute field that carries the object attribute information. It can be array, list, file or even more complex data structure.

5.1.8 General Details of each Implemented Object

This field of the object model is optional, and multiple instances of this item can be existent in an object exchange model. Each instance shall be a description of the details of each known implemented object using this model. This item of the object exchange model shall consist of the following fields:

- Company Name: Stores the name of the company that implemented this object.
- Company Contact Info: Stores the contact information of the company that implemented this object, most probably the phone number.
- Company Contact Person: Stores the information of the person to contact from within the company should any enquiries be made. This field is further divided into two sub-fields:
  - First Name: The first name of the contact person
  - Last Name: The last name of the contact person
- Company Email Address: Stores the valid email address that any enquiries can send to.
- Version: Stores the version number of this implemented object.
- Date of Version: Stores the date that this version of the implemented object had been implemented. This field is further divided into three sub-fields:
  - Day: Shall store the numerical value of the day.
  - Month: Shall store the numerical value of the month.

- Year: Shall store the numerical value of the year.

- Implementation Platform: Stores the platform in which the object was implemented.
- Reference Link to Implemented Object: Stores the URL where the implemented object is located.

General Details of Each Implemented Object		
Category	**Information**	
Company Name		
Company Contact Info		
Company Contact Person	First Name	
	Last Name	
Company Email Address		
Version		
Date of Version	Day	
	Month	
	Year	
Implementation Platform		
Reference Link to Implemented Object		

Table 5-6: General Details of Each Implemented Object

The new DTD schema (version 2) of the OEMT DIF can be found in appendix A.

5.2 Application Tool Kit Development

In the GRIDS environment, the exchange objects are specified using the OEMT. However, the generation of the OEMT DIF object specification file and the object exchange model is extremely time consuming. Therefore, the ability to automate the process of object model development is a significant concern. In this work, an OEMT application tool kit is developed to aid the simulation builders in the automatic creation of the DIF file and object exchange model. This tool kit also implements the OEMR

services. In addition, in this chapter, we propose a novel concept --- **Object Exchange Model Dictionary (OEMD)** which provides both the standard nomenclature that specifies common representations of data used within an OEMT and the reusable components to be used in object model construction.

This section describes the functions of the OEMT application tool kit in detail. Section 5.2.1 introduces the basic function of the tool kit. Section 5.2.2 represents the OEMR services provided by the kit. The motivation, conception and functionality of the OEMD are introduced in section 5.2.3.

5.2.1 Basic Function

The OEMT application tool kit is developed to facilitate the generation of DIF object specification file and the object model. It releases simulation builders from hard work and saves time and effort in the GRIDS federation development process.

In this tool, the OEMT is given in tabular format as shown in Figure 5-1, 5-2 and 5-3 so that it is very easy for users to follow.

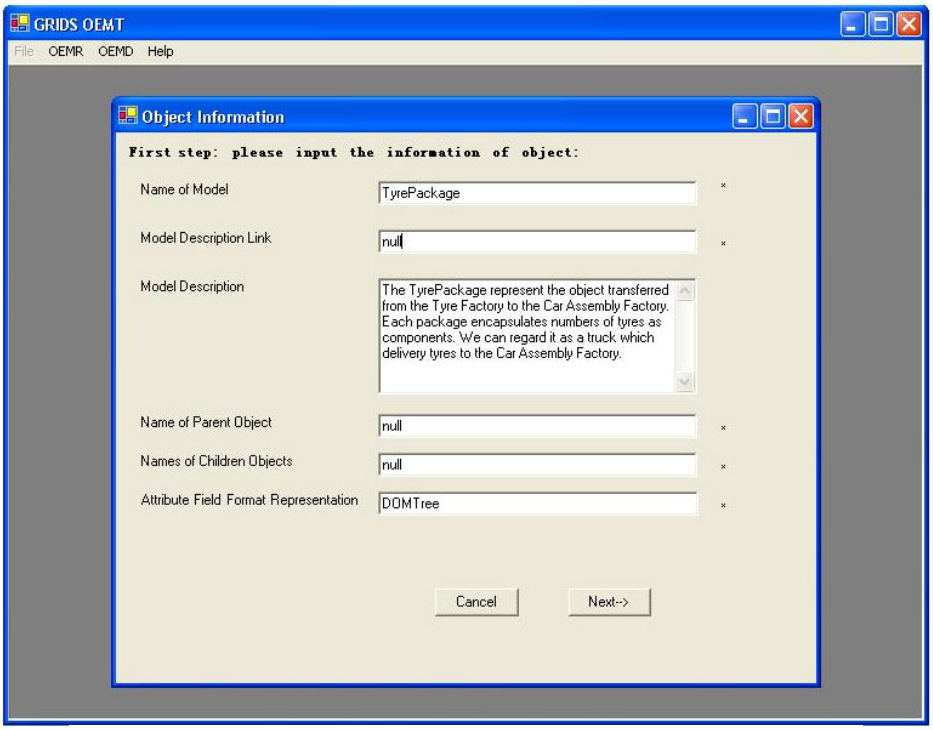

Figure 5-1: GRIDS OEMT Software User Interface---Exchange Object Information Table

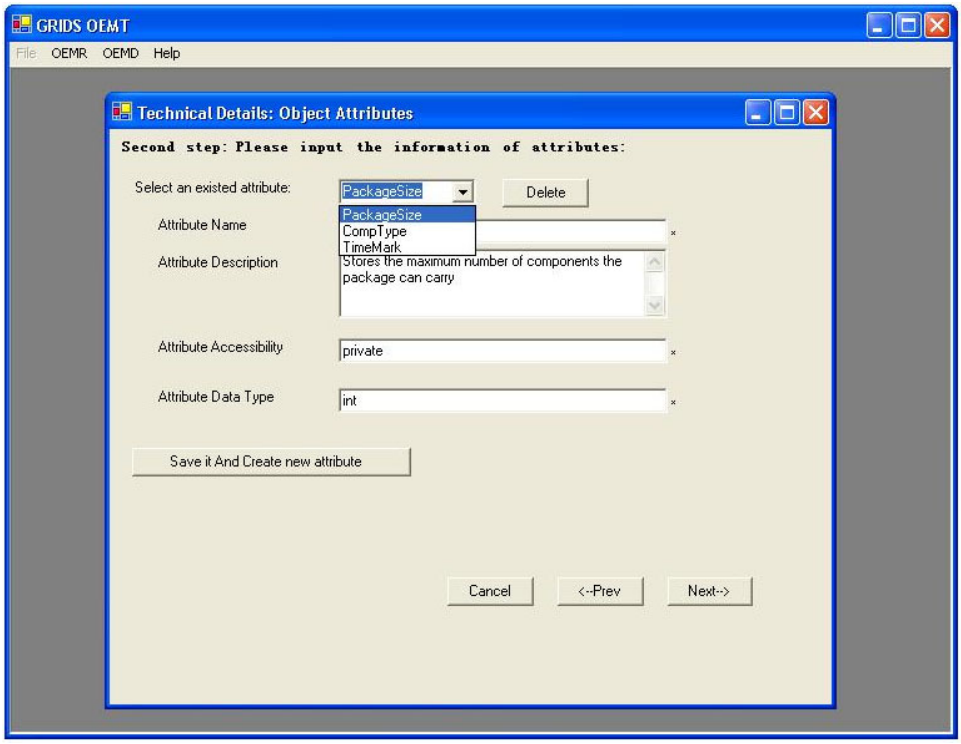

Figure 5-2: Exchange Object Attributes Table

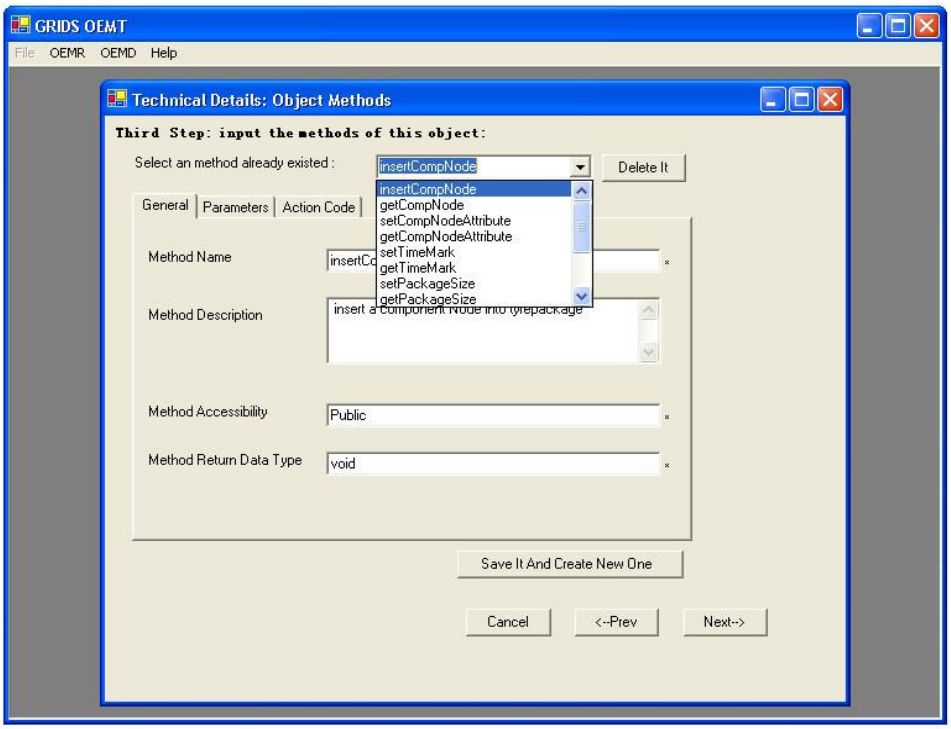

Figure 5-3: Exchange Object Methods Table

There are six such tables according to the tables in the new OEMT specification in

section 5.1. Figure 5-1, 5-2 and 5-3 shows the first three tables --- Exchange Object Information Table, Attributes Table and Methods Table. The other three tables are omitted due to the limited space. By filling a certain set of object information, the OEMT DIF object specification file is automatically created based on the tables (e.g. Appendix B). And a framework of the object model is also created by this tool with user defined attributes and methods. Users can start their object model development process from this framework, thus reducing time and cost for building a new object model.

5.2.2 OEMR On-Line Services

While some applications may choose to construct the object model entirely from scratch, significant savings can be achieved in the object model development process through reuse of existing models. The Object Exchange Model Repository (OEMR) is the central location where all object exchange models' information is stored for distributed simulation builders to access and re-use object models. A developer may utilize an existing object model chosen from the OEMR as a starting framework for new object model construction, modifying the model as appropriate to address the particular requirements of the application.

In this work, the OEMR is developed to store object models in different areas of applications separately. The models are well classified in the library, thereby ensuring easy searching by simulation builders for relevant object models for their federation. New implemented object models can be uploaded to the OEMR, thus providing a centralized area for simulation builders to exchange their models for reuse, affording convenience to the users. The OEMR is stored in a web server, at a known URL and known port for easy global accessibility.

Figure 5-4 and 5-5 show the OEMT user interface of this OEMR tool. Upon clicking the **Update Local Samples Storage** label in the OEMR menu, the system will provide several server locations in different countries for users to select (figure 5-4), and then the system will connect to the chosen OEMR server to get update information. An interface like Figure 5-5 will be shown with a list of application areas in the OEMR and the latest update date of each application area. Simulation builders can select the application areas they are interested in, click the **Update** button, and all the object models in this application area will be downloaded to the local storage of samples.

Figure 5-4: OEMR Server Location Information

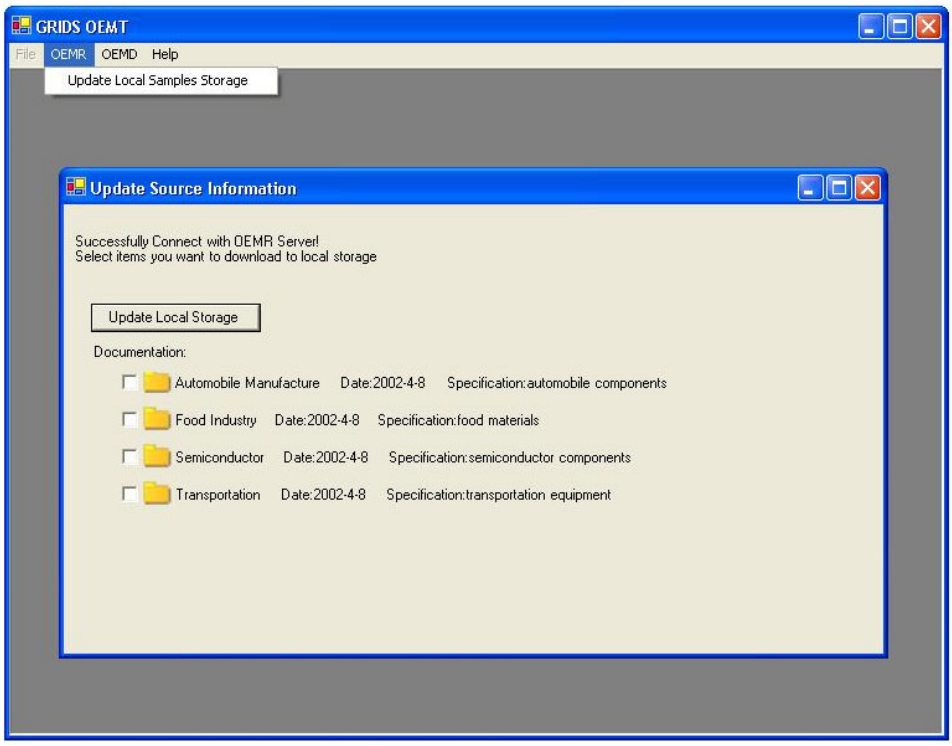

Figure 5-5: OEMR User Interface

The object exchange model can be created by reusing the existing models in OEMR with the following procedures:

- On-line search for the object model templates of relevant application areas in the OEMR and download them to local sample storage from the server.
- Select and open the object model that the simulation builder wants to refer.
- Modify the object model according to the requirement of the simulation builder. Additional attributes, methods or other information can also be added.

Creating new object model by this means makes the simulation builder's work easier and also promotes the reuse of object models.

5.2.3 Object Exchange Model Dictionary (OEMD)

5.2.3.1 Motivation and Conception

The OEMT development process for a given application represents a key GRIDS federation design activity. GRIDS OEMT does not mandate names or attributes

structures. For GRIDS compliance, the exchange object model is only required to be internally consistent. However, the lack of universal consistency causes ambiguities which obstruct the understanding and reuse of object models. For example, the name of an object model can be varied. An "automobile" can also be called "car" or "vehicle". Moreover, the encoding of the name may be different as different designers may use "AUT", "AUTO" or even "AUTO_". This confusion is unacceptable in the OEMT development.

When the OEMR keeps expanding, the unambiguous description of each object exchange model in the repository becomes necessary. In addition, if federates use the same terminology in their OEMT object models, it will make them more readable to each other and enhances general interoperability. The requirement is recognized and this work brings forward an idea to create an Object Exchange Model Dictionary (OEMD) which assists GRIDS federation developers by specifying common representations of data used within an OEMT that will be shared with other federates. The OEMD is inspired by the HLA OMDD [19, 26], but does not follow it. It is intended to support both pre-runtime and runtime data exchange via GRIDS RTI.

5.2.3.2 OEMD Components

In order to support precision in representation, the initially designed OEMD will include the following components:

- **The standard code:** defines the standard character which can be used in the naming of object, attribute, method and parameter of the OEMT. It includes ASCII alphabetic characters "a" to "Z", numeric characters "0" to "9" and some other symbol including "_"and ".".
- **Standard encoding rules:** provides the rules to encode object name, attribute name, method name and parameter name in the OEMT.

- The first character of object name, attribute name, method name and parameter name must all be alphabetic characters.
- OEMT does not distinguish uppercase letter and lowercase (e.g. length and LENGTH will be regarded as same word).
- For objects whose full name in real world is a single word, the whole word with the first character in capital is preferred in the OEMT; for those which are named more than one word, mnemonic abbreviation (e.g. acronyms) in capital is recommended. Attribute and parameter naming also obey this rule.
- Method name should clearly represent its purpose and operation objective. One verb in lowercase followed by a noun (with the first character in capital) or an abbreviation (all characters in capital) is recommended (e.g. setLength()).
- Abbreviation must result in a shared understanding of "What the thing is" or "What it potentially means" to all participating applications.

- **Standard object name list:** This is a list of existing object standardized names for simulation builder reference. The names are lexicographically ordered. Locating an appropriate code typically requires a linear search.
- **Standard attribute list:** Each standard object name is linked to a list of standard attributes. These attributes can but need not follow the object. They can be extracted as components to construct other objects. The OEMD offers a "loose couple" between attributes and their data types. Each attribute is provided a recommended default data type but the actual used data type is decided by the federate builders.
- **Standard method database:** This is the storage of standardized methods for simulation builders' reference and reuse. The methods are categorized into

different functionalities. The method names in each category are lexicographically ordered and locating an appropriate code typically requires a linear search. For each method, a general description, accessibility, parameters and return data type are recorded. Users can change these items according to their requirements. The method code is also provided for reference.

The OEMD is an assistant tool for the OEMT standard object development. The OEMR provides the whole GRIDS object exchange model, whereas the OEMD provides both the standard nomenclature and the components to be used in object model construction. The OEMD does not specify an object exchange model structure in detail, but rather provide the "pieceparts" which may be merged together to construct a new GRIDS object exchange model. These characteristics enhance the reusability and also facilitate the integration of federates during federation development.

The OEMD is currently under development. There are also lots of questions that need to be answered urgently. For example, which statement is preferred between a "truck" object and a generic "vehicle" object with a functional category attribute of "truck"? Is there a standard set of attributes for a particular object? Or which attribute is absolutely necessary? Should all "vehicles" have a "color" and "size"? Is it an error if these values are not present in a template? Is it an error if a "car" has a "depth"? These questions still need further research and the next step will also include verification and validation of the OEMD.

5.2.3.3 OEMD Services in the OEMT Application Tool Kit

The OEMT application too kit includes the OEMD services as described in section 5.2.3.2. For example, by clicking the **Standard Object Name List** label in the OEMR menu, system will show the list in Figure 5-6 for the simulation builders' reference. A

user can also view the recommended attributes of each object by clicking the object name. Figure 5-7 shows the recommended attributes of Carbody Object.

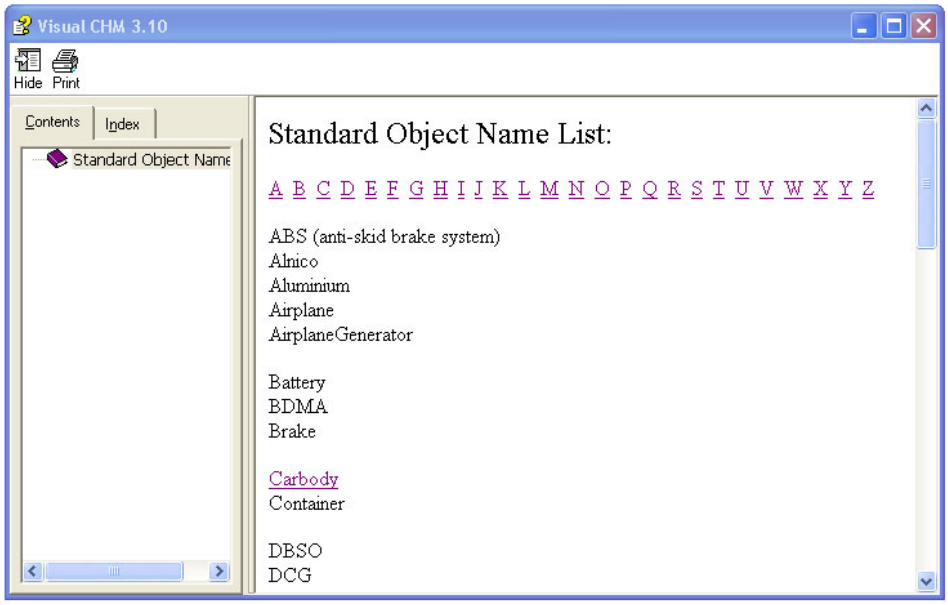

Figure 5-6: Standard Object Name List

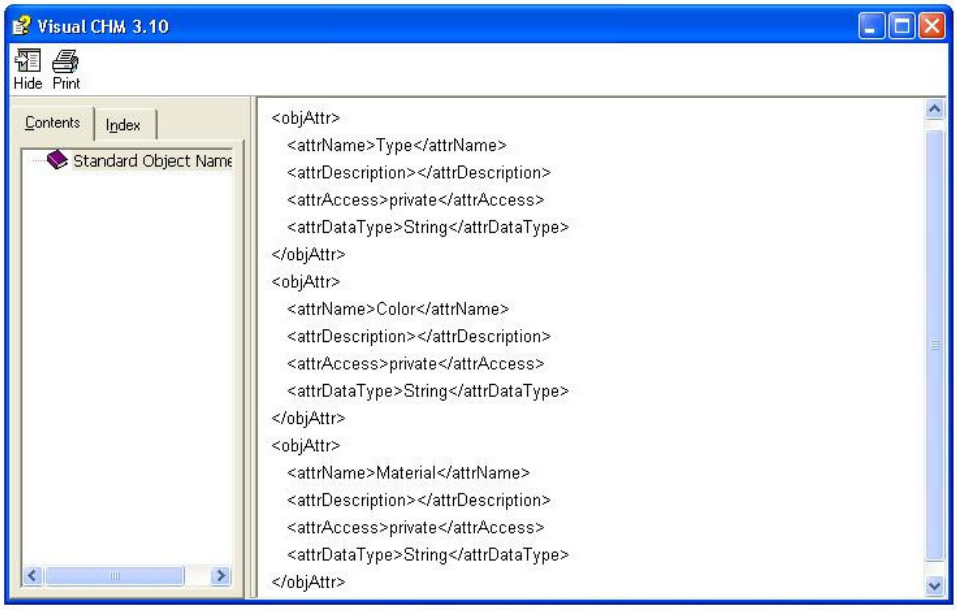

Figure 5-7: Recommended Attributes of Carbody Object

The OEMD tool provides not only the standard code, encoding rules, standard object name and attributes list, but also offers methods database service that can be invoked during the creation of the object models. The methods database provides many

basic methods for simulation builders to reuse. As Figure 5-8 shows, user will be asked to choose a method database file before opening the object methods table in a template.

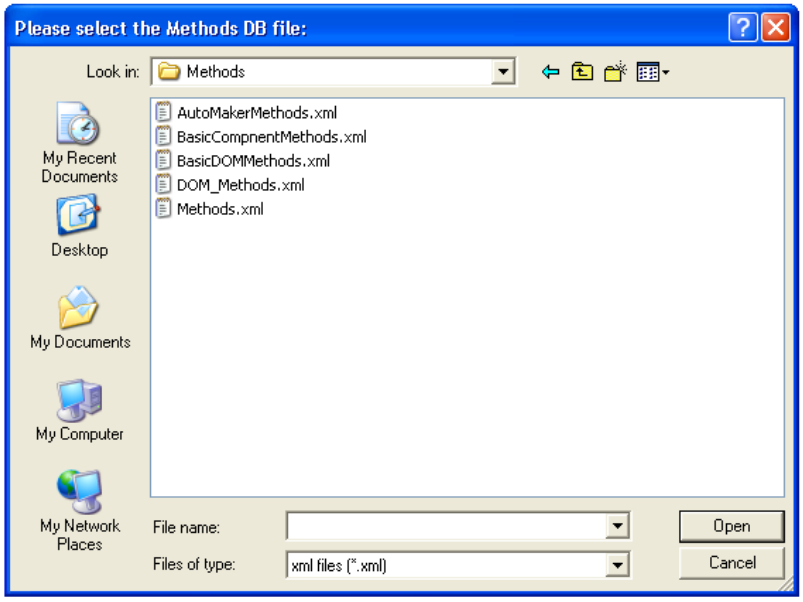

Figure 5-8: Methods Database File List

The methods in the selected file will be imported into the template and help the user fill in the Exchange Object Methods Table (Figure 5-3). For each method, its name, general description, accessibility, parameters and return data type are recommended. The method code is also provided which is used to create object model framework. Simulation builders can either create new method by themselves or reuse suitable existing methods provided by the system (Figure 5-9).

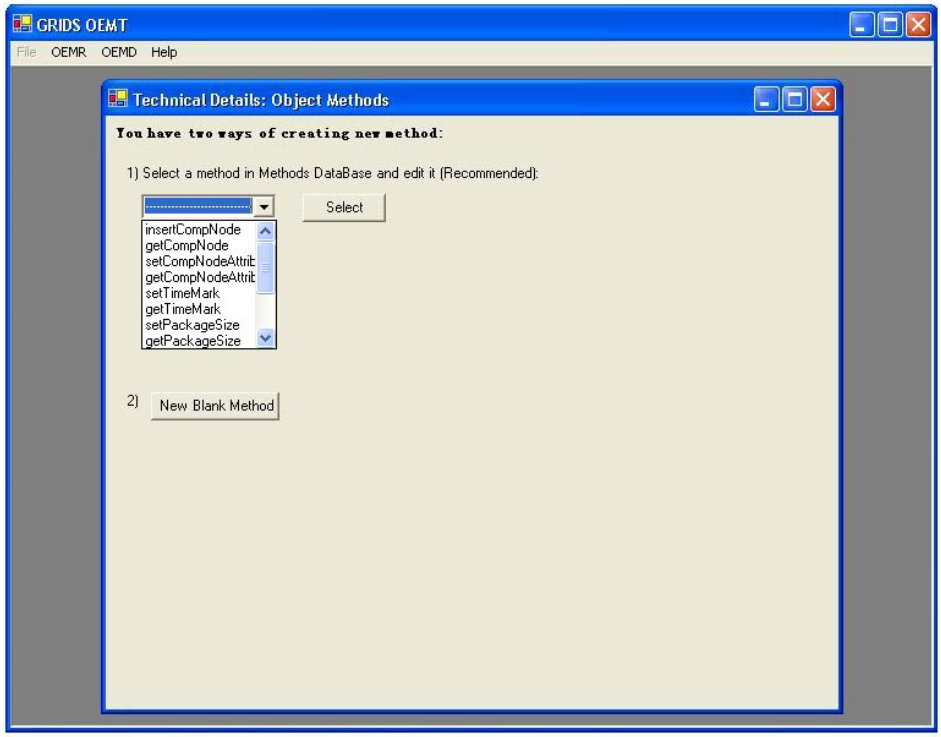

Figure 5-9: Methods Database User Interface

The tool also provides on-line update service for the local OEMD method storage. The method files are categorized under different functionalities and stored in the Web server as XML files. As the interface shows in figure 5-10, a user can connect to the server to get the up-to-date information of the reusable methods.

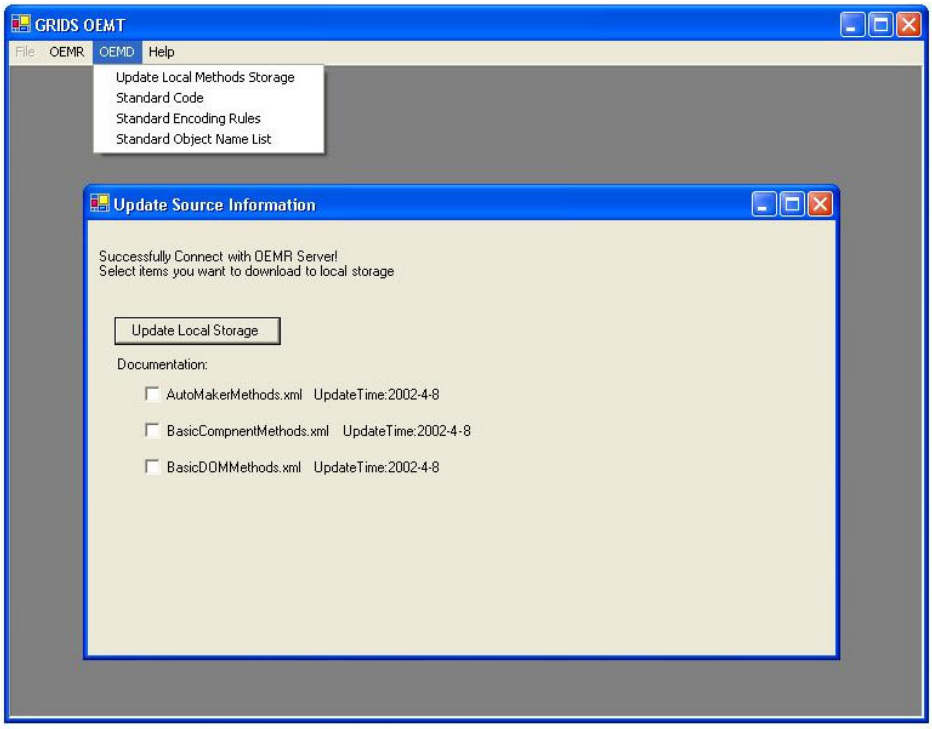

Figure 5-10: OEMD Methods Database Update Interface

Chapter 6 Conclusion and Future Work

With the globalization of the commercial markets and the great development of World Wide Web, distributed simulation is becoming more and more important in investigating issues. Researches related to distributed simulation such as middleware, standard information format also gain great attention. The aim of the research described in this thesis was to investigate the standardization of object exchange models and the GRIDS middleware in component-based distributed simulation to support DSC and other various types of simulation applications. This thesis presented the series of research effort for achieving this aim.

Section 6.1 gives the conclusions of the research. The contributions achieved through this work are highlighted. Section 6.2 discusses some potential future directions of this research.

6.1 Conclusion

The following gives the conclusions and contributions of the research work described in this thesis.

1. Make a thorough literature survey on past work and documentations to set up a solid background that is useful in fulfilling subsequent research.

This includes understanding distributed simulation environment, especially the DSC simulation; understanding the requirements and challenges faced by all

distributed simulation which are the foundation stone of supporting middleware development; investigating popular middleware including HLA, GRIDS and CORBA and determining the advantages and disadvantages inherent in these middleware and make a fair comparison.

2. Evaluate the capability of the GRIDS as a middleware to facilitate distributed simulation.

GRIDS represents an early adopter of the component RTI philosophy [33]. It provides an extensibility mechanism to add additional service components in the form of thin agents and package interfaces capable of supporting the demands of distributed simulation. It is a suitable infrastructure for implementing various types of simulations due to its lightweight and ease of extensibility. However, GRIDS is still a preliminary infrastructure. There is still a long way to go before GRIDS can be used in industry.

This thesis gave an objective comparison of GRIDS with two other middleware architectures. The suitability of GRIDS in distributed simulation was examined and investigated by two case studies. GRIDS was verified and validated as a valid middleware to facilitate DSC simulation applications. However, it was also shown that the architecture of GRIDS was still incomplete. The thesis summarized the advantages and disadvantages of GRIDS and provided proposals of future improvement and utility of the architecture.

The case studies also offered practical distributed simulation instances to test the DDM and TM thin agents. Through the practical implementation, numerous suggestions are given to improve the TM thin agent.

3. Investigate and evaluate the capability of the OEMT to standardize object exchange model; provide proposal for the improvement of the OEMT.

Most current simulation products use message-passing mechanism to cooperate in a federation. However, the requirement of object-passing mechanism cannot be ignored. This work further convinced the current simulation community of the usefulness of object-passing within distributed simulation environment, which requires simulation packages to incorporate the use of actual objects and specified the problem to standardize object exchange model.

The OEMT has been proposed and implemented to aid object-passing in distributed simulation environment. The OEMT establishes an open standard for interchanging information between federates. The establishment of this open framework directly supports the promoting of an interoperable set of simulation models. From the original release of the OEMT in 2001, the format definition has continued to mature and evolve. In chapter 5, two case studies were designed and implemented to examine and evaluate the capability of OEMT. Since this was the first attempt to implement the OEMT in complex simulation systems, a number of problems arised and the solutions were given through great effort. During the process, new demands were detected and valuable experience and lessons were gained. This thesis offered a new version of the OEMT with some important fine-tuning based on the in-depth research.

4. Develop an OEMT application tool kit; implement the OEMR on-line services; and introduce a new tool---OEMD to facilitate the creation and reuse of object models.

Since the generation of the OEMT DIF object specification files and object models is time and effort consuming, the ability to automate the process of object model development is significant. The other important contribution of this research work is an application tool kit implemented specially for simulation builders to

create object models easily. This kit aided the simulation builders in the automatic generation of DIF object specification files and the object exchange models. It provided OEMR on-line services which allow easy access by simulation builders all over the globe.

Furthermore, the ability of the OEMT was enhanced by adding the new tool, OEMD. As a common template, the OEMT does not restrict the names, lexicon, or data type definition for the contents of the GRIDS object exchange model. However, tremendous benefits can be obtained by using standard nomenclature in GRIDS object exchange model development and maintenance. An additional tool is required to satisfy the foregoing requirements. Another important contribution of this research work is that it proposed the Object Exchange Model Dictionary and implemented the original framework. The OEMD provides a solid basis for the OEMT by providing a rigorous, extensible, universally consistent and increasingly complete data dictionary from which to select object parts or attributes for use in simulation. It addresses an extensible manner capable of meeting the needs of both pre-simulation and run-time data representation. The understanding and reusability of the object model were enhanced because of the commonality of names and lexicons that the OEMD facilitates.

6.2 Future Work

There is no end for all research, and the research presented in the thesis is no exception. The next researcher is recommended to do further work as follows:

- **OEMT and OEMD work to be accomplished**

  Although this work did significant improvement on the OEMT, it still can be further expanded.

  The development of the OEMD is currently in the developmental stage. The

review from outside the OEMD development team is needed. There are also lots of problems to be solved urgently, such as how to define a standard set of attributes for a particular object, how to determine which attribute is absolutely necessary. These questions still need deep research and the next step will also include verification and validation of the OEMD.

- **Make OEMT an International Standard**

  One consideration is merging OEMT into HLA OMT; this possibility needs future investigation.

- **Further work on perfecting GRIDS architecture**
  - Current GRIDS is based on Java. Future work will realize interface to mapping other programming languages, so that it can fit more simulation models from different organizations.
  - Future studies can be done for dynamical federate discovering, joining in and withdrawing at runtime.
  - Future work is expected on an object handling interface between the simulation package and the GRIDS architecture. This would thus remove the need to amend the simulation packages, and the applications need only inform this interface of the information it needs, and the interface sends the information to the application.
  - Thin agent services in GRIDS need to be improved. More services are expected such as secrecy, ownership management, execution data collection and maintenance etc. and more algorithms are expected for existing thin agent.

References

[1] A. Chan and T. Spracklen, "*Web-Based Distributed Object Simulation Framework*", Proceedings of the Summer Computer Simulation Conference, 1999.

[2] A.M. Law and W.D. Kelton, *Simulation Modeling and Analysis,* McGraw-Hill, third edition, 2000.

[3] B.P. Gan, L. Liu, S. Jain, S.J. Turner, W. Cai, and W.J. Hsu, "*Distributed Supply Chain Simulation Across Enterprise Boundaries*", Proceedings of the Winter Simulation Conference, pp 1245-1251, 2000.

[4] B.P. Zergler, D. Kim, S.J. Buckley, "*Distributed Supply Chain Simulation IN A DEVS/CORBA Execution Environment*", Proceedings of the Winter Simulation Conference, pp 1333-1340, 1999.

[5] C. Mclean and F. Riddick, "*The IMS Mission Architecture for Distributed Manufacturing Simulation*", Proceedings of the Winter Simulation Conference, pp 1539-1548, 2000.

[6] D. C. Schmidt, *Overview of CORBA, 2003.*
URL: http://www.cs.wustl.edu/~schmidt/corba-overview.html

[7] *Document Object Model (DOM) Level 3 Core Specification*, Version 1.0, W3C Working Draft 22, 2002. URL: http://www.w3.org/DOM

[8] E. Christiaanse and K. Kumar, "*ICT-enabled coordination of dynamic supply webs*", International Journal of Physical Distribution and Logistics Management, Vol. 30, No. 3/4, pp.268-285, 2000.

[9] G. Archibald, N. Karabakal and P. Karlsson, "*Supply Chain vs. Supply Chain: Using Simulation to Compete Beyond the Four Walls*", Proceedings of the

Winter Simulation Conference, pp.1207-1214, 1999.

[10] G. Tan, W.N. Ng and S.J.E. Taylor, *Generic Runtime Infrastructure for Distributed Simulations (GRIDS) Distributed Supply Chain (DSC) Object Exchange Model Template (OEMT) Specifications Version 1.0*, Modeling and Simulation Group, National University of Singapore, 2001.

[11] G. Tan, W.N. Ng and S.J.E. Taylor, "*Developing An Object Exchange Model Template (OEMT) for GRIDS Distributed Supply Chain (DSC) Simulations*", Proceedings of the 35th Annual Simulation Symposium, pp 259–266, 2002.

[12] G. Tan, W.N. Ng and S.J.E. Taylor, "*Developing Data Distribution Management (DDM) Thin Agent (TA) and Services for GRIDS Distributed Supply Chain (DSC) Simulation*", Proceedings of the European Simulation Interoperability Workshop, 02-ESIW-41, 2002

[13] H.B. Chen, O. Bimber, C. Chhatre, E. Poole, and S. J. Buckley. "*eSCA, A Thin-Client/Server/Web-enabled System for Distributed Supply Chain Simulation*", Proceedings of the Winter Simulation Conference, pp.1371-1377, 1999.

[14] *IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) – framework and rules*, Institute of Electronic and Electrical Engineers, IEEE Standard 1516, 2000.

[15] *IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) – federate interface specification*, Institute of Electronic and Electrical Engineers, IEEE Standard 1516.1, 2000.

[16] *IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) – object model template (OMT) specifications*, Institute of Electronic and Electrical Engineers, IEEE Standard 1516.2, 2000.

[17] J. Banks, J.S. Carson and B.L. Nelson, *Discrete-Event System Simulation*, Prentice Hall, 2nd edition, 1996.

[18] J. Dahmann, F. Kuhl and R. Weatherly, "*Standards for Simulation: As Simple as Possible But Not Simpler-The High Level Architecture for Simulation*", Simulation, Vol. 71, No. 6, pp 378-387, 1998.

[19] J. Hammond, M. Dey, R. Scrudder and F. Merkin, "*Populating the HLA Object Model Data Dictionary-A Bottom-Up Approach*", Proceedings of the Spring Simulation Interoperability Workshop, 98S-SIW-075, 1998.

[20] J.K. Liker and Y.C. Wu, "*Japanese Automakers, U.S. Suppliers and Supply-Chain Superiority*", Sloan Management Review, pp. 81-93, 2000.

[21] J.L. Rosenberger, *Teach Yourself CORBA in 14 Days,* SAMS Publishing, 1998.

[22] J. Misra, *Distributed Discrete-Event Simulation,* ACM Computer Survey 18, Vol. 1, March 1986, pp 39-65, 1986.

[23] J. Ulriksson, F. Moradi and O. Svensson, "*A web-based Environment for building Distributed Simulations*", Proceedings of the European Simulation Interoperability Workshop, 02E-SIW-036, 2002.

[24] K. Kumar, "*Technology for Supporting Supply Chain Management*", Communications of the ACM, Vol. 44, No. 6, pp. 58-61, 2001.

[25] L.W. Lacy and M.T.D. Dugone, "*Developing International Standards for Interchanging Simulation Data*", Proceedings of the European Simulation Interoperability Workshop, 01E-SIW-048, 2001.

[26] P.A. Birkel, "*SEDRIS Data Coding Standard (SDCS)* ", Proceedings of the Spring Simulation Interoperability Workshop, 99S-SIW-001, 1999.

[27] P. Davidsson and F. Wernstedt, "*Characterization and Evaluation of Just-In-Time Production and Distribution*", Proceedings of the AAMAS

workshop on MAS Problem Spaces and Their Implications to Achieving Globally Coherent Behavior, 2002.

[28] P.G. Goh, *IT and Supply Chain Management,* Course Notes, School of Computing, National University of Singapore, 2001.

[29] R. Lutz, R. Scrudder and J. Graffagnini, "*High Level Architecture Object Model Development and Supporting Tools*", Simulation, Vol. 71, No. 6, pp 401-409, 1998.

[30] R.M. Fujimoto, "*Parallel and Distributed Simulation*", Proceedings of the Winter Simulation Conference, pp. 122-130, 1999.

[31] R.M. Fujimoto, "*Parallel and Distributed Simulation Systems*", Proceedings of the Winter Simulation Conference, pp. 147-157, 2001.

[32] R. Sudra, S.J.E. Taylor and T. Janahan, "*Distributed Supply Chain Simulation in GRID*", Proceedings of the Winter Simulation Conference, pp. 356-361, 2000.

[33] R. Sudra, S.J.E. Taylor, "*Extensibility: Modular HLA RTI Services*", Proceedings of the European Simulation Interoperability Workshop, 02E-SIW-049, 2002.

[34] S. Jain, C.C. Lim, B.P. Gan, and Y.H. Low, "*Criticality of Detailed Modeling in Semiconductor Supply Chain Simulation*", Proceedings of the Winter Simulation Conference, pp 888-896, 1999.

[35] S. Jain, R.W. Workman, L.M. Collins, E.C. Ervin and A.P. Lathrop, "*Development of a High-Level Supply Chain Simulation Model*", Proceedings of the Winter Simulation Conference, pp.1129-1137, 2001.

[36] S.J.E. Taylor, G. Tan, and J. Ladbrook, "*Data Distribution Management for Distributed Supply Chain Simulation*", Proceedings of the Fifth UK Simulation

Society Conference, pp.35-41, 2001.

[37] S.J.E. Taylor, R. Sudra, T. Janahan, G. Tan, and J. Ladbrook, "*Towards COTS Distributed Simulation Using GRIDS*", Proceedings of the Winter Simulation Conference, pp 1372-1379, 2001.

[38] S.J.E. Taylor, R. Sudra, G. Tan and J. Ladbrook, "*Issues in Developing a Distributed Supply Chain Simulation for the Automotive Industry*", Proceedings of the European Simulation Interoperability Workshop, pp. 629-637, 2001.

[39] S.J.E. Taylor, R. Sudra, T. Janahan, G. Tan and J. Ladbrook, "*GRIDS-SCF: An Infrastructure for Distributed Supply Chain Simulation*", Simulation, Vol. 78, No.5, pp. 312-320, 2002.

[40] S.J. Turner, W. Cai and B.P. Gan, "*Adapting a Supply-Chain Simulation for HLA*", Proceedings of the Distributed Simulation and Real Time Applications, pp 71-78, 2000.

[41] *The Common Object Request Broker: Architecture and Specification*, Version 3.0, 2002.

URL: http://www.omg.org/technology/documents/spec_catalog.htm

[42] W.N. Ng, *Adapting A Middleware for Distributed Supply Chain Simulation* M.Sc. Thesis, Department of Computer Science, National University of Singapore, 2002.

[43] *XML DOM Tutorial*, 2003.

URL: http://www.w3schools.com/dom/default.asp

[44] Y. Lois, J. Steinman and G. Blank, "*Adapting Your Simulation for HLA*", Simulation, Vol. 71, No. 6, pp 410-420, 1998.

Appendix A: OEMT DIF DTD Schema (version 2.0)

```
<!ELEMENT OEMT_Library (Application_Area)*>
  <!ELEMENT Application_Area (areaName,objectModel*)>
    <!ELEMENT areaName (#PCDATA)>
    <!ELEMENT objectModel (modelName, parentObjName?, childrenObjName*, attrFieldDataType?,
                           modelLink|(description, technicalDetails), time?, implementedObjectDetails*)>
      <!ELEMENT modelName (#PCDATA)>
      <!ELEMENT parentObjName (#PCDATA)>
      <!ELEMENT childrenObjName (#PCDATA)>
      <!ELEMENT attrFieldDataType (#PCDATA)>
      <!ELEMENT modelLink (#PCDATA)>
      <!ELEMENT description (#PCDATA)>
      <!ELEMENT technicalDetails (objAttr+, objectComponent* ,objMethod*)>
        <!ELEMENT objAttr (attrName, attrDescription, attrAccess, attrDataType)>
          <!ELEMENT attrName (#PCDATA)>
          <!ELEMENT attrDescription (#PCDATA)>
          <!ELEMENT attrAccess (private|protected|public)>
          <!ELEMENT attrDataType (#PCDATA)>
        <!ELEMENT objectComponent (ComponentName, CompDescription, CompDetail)>
          <!ELEMENT ComponentName (#PCDATA)>
          <!ELEMENT CompDescription(#PCDATA)>
          <!ELEMENT CompDetail (CompAttr+)>
            <!ELEMENT CompAttr (attrName, attrDescription, attrAccess, attrDataType)>
              <!ELEMENT attrName (#PCDATA)>
              <!ELEMENT attrDescription (#PCDATA)>
              <!ELEMENT attrAccess (private|protected|public)>
              <!ELEMENT attrDataType (#PCDATA)>
        <!ELEMENT objMethod (methodName, methodDescription, methodAccess, methodParameters,
                             methodReturnDataType)>
          <!ELEMENT methodName (#PCDATA)>
          <!ELEMENT methodDescription (#PCDATA)>
          <!ELEMENT methodAccess (private|protected|public)>
          <!ELEMENT methodParameters (pName,pDataType)*>
            <!ELEMENT pName (#PCDATA)>
            <!ELEMENT pDataType (#PCDATA)>
          <!ELEMENT methodReturnDataType (#PCDATA|void)>
      <!ELEMENT time (timeStamp?, lookahead?)>
        <!ELEMENT timeStamp (dataType, scale)>
          <!ELEMENT dataType (#PCDATA)>
          <!ELEMENT scale (#PCDATA)>
        <!ELEMENT lookahead (dataType, scale)>
```

```
        <!ELEMENT dataType (#PCDATA)>
        <!ELEMENT scale (#PCDATA)>
<!ELEMENT implementedObjectDetails (companyName, companyContactInfo, companyContactPerson,
                                    companyEmail, version, versionDate, referenceLink+)>
    <!ELEMENT companyName (#PCDATA)>
    <!ELEMENT companyContactInfo (#PCDATA)>
    <!ELEMENT companyContactPerson (firstName, lastName)>
        <!ELEMENT firstName (#PCDATA)>
        <!ELEMENT lastName (#PCDATA)>
    <!ELEMENT companyEmail (#PCDATA)>
    <!ELEMENT version (#PCDATA)>
    <!ELEMENT versionDate (day, month, year)>
        <!ELEMENT day (#PCDATA)>
        <!ELEMENT month (#PCDATA)>
        <!ELEMENT year (#PCDATA)>
    <!ELEMENT implementationPlatform (#PCDATA)>
    <!ELEMENT referenceLink (#PCDATA)>
```

Appendix B: DIF Specification for the TyrePackage Object

```
<?xml version="1.0"?>
 <objectModel>
   <modelName>TyrePackage</modelName>
   <description>
    The TyrePackage represent the object transferred from Tyre Factory to Car Assembly Factory. Each
    package encapsulates numbers of tyres as components. We can regard it as a truck which delivery tyres
    to Car Assembly Factory.
  </description>
  <descriptionLink>nil</descriptionLink>
  <parentObj>nil</parentObj>
  <childObj>nil</childObj>
  <attrFieldDataType> XML Document Object Model Tree </attrFieldDataType>
  <technicalDetails>
    <objAttr>
      <attrName>PackageSize</attrName>
      <attrDescription>Stores the maximum number of components the package can carry</attrDescription>
      <attrAccess>private</attrAccess>
       <attrDataType>int</attrDataType>
    </objAttr>
      <objAttr>
      <attrName>CompType</attrName>
      <attrDescription>Stores the type of components in the package</attrDescription>
      <attrAccess>private</attrAccess>
       <attrDataType>String</attrDataType>
    </objAttr>
    <objAttr>
      <attrName>TimeMark</attrName>
      <attrDescription>Stores the time stamp of the object</attrDescription>
      <attrAccess>private</attrAccess>
       <attrDataType>int</attrDataType>
    </objAttr>
    <objectComponent>
      <ComponentName>tyre</ComponentName>
      <ComponentDescription> Tyre is the component in Tyre package </ ComponentDescription >
      <ComponentDetail>
        <CompAttr>
          <attrName>CompID</attrName>
          <attrDescription>Stores the index of individual tyre</attrDescription>
          <attrAccess>private</attrAccess>
```

```
            <attrDataType>int<</attrDataType>
        </ CompAttr >
        <CompAttr>
            <attrName>UTQG </attrName>
            <attrDescription>
            The Department of Transportation requires each manufacturer to grade its tyres under the
            Uniform Tyre Quality Grade (UTQG) labeling system and establish ratings for treadwear,
            traction, and temperature resistance.
            </attrDescription>
            <attrAccess>private</attrAccess>
            <attrDataType>String</attrDataType>
        </ CompAttr >
        < CompAttr >
            <attrName>MaxLoad</attrName>
            <attrDescription> Stores the maximum load of the tyre </attrDescription>
            <attrAccess>private</attrAccess>
            <attrDataType>float</attrDataType>
        </ CompAttr >
        < CompAttr >
            <attrName> MaxInflationPress </attrName>
            <attrDescription>Stores the maximum inflation press of the tyre </attrDescription>
            <attrAccess>private</attrAccess>
            <attrDataType>float</attrDataType>
        </ CompAttr >
    <ComponentDetail>
</objectComponent>
<objMethod>
    <methodName>insertCompNode</methodName>
    <methodDescription> insert a component Node into tyrepackage </methodDescription>
    <methodAccess>public</methodAccess>
    <methodParameters>
        <pName>CompNode</pName>
        <pDataType>XML DOM Node</pDataType>
    </methodParameters>
        <methodReturnDataType>void</methodReturnDataType>
 </objMethod>
 <objMethod>
      <methodName>getCompNode</methodName>
      <methodDescription>get a component Node from tyrepackage </methodDescription>
      <methodAccess>public</methodAccess>
      <methodReturnDataType>XML DOM Node </methodReturnDataType>
 </objMethod>
 <objMethod>
```

```
        <methodName>setCompNodeAttribute</methodName>
        <methodDescription>set attribute value to a attribute of the component Node </methodDescription>
        <methodAccess>public</methodAccess>
        <methodParameters>
            <pName>compNode</pName>
            <pDataType>XML DOM Node</pDataType>
            <pName>attrName</pName>
            <pDataType>String</pDataType>
            <pName>attrValue</pName>
            <pDataType>String</pDataType>
        </methodParameters>
        <methodReturnDataType>void </methodReturnDataType>
    </objMethod>
    <objMethod>
        <methodName>getCompNodeAttribute</methodName>
        <methodDescription> extract String attribute value from a component Node </methodDescription>
        <methodAccess>public</methodAccess>
        <methodParameters>
            <pName>compNode</pName>
            <pDataType>XML DOM Node</pDataType>
            <pName>attrName</pName>
            <pDataType>String</pDataType>
        </methodParameters>
        <methodReturnDataType>String </methodReturnDataType>
    </objMethod>
    <objMethod>
        <methodName>setTimeMark</methodName>
        <methodDescription>set the timeMark attribute of tyrepackage</methodDescription>
        <methodAccess>public</methodAccess>
        <methodParameters>
            <pName>intTimeMark</pName>
            <pDataType>int</pDataType>
        </methodParameters>
        <methodReturnDataType>void</methodReturnDataType>
     </objMethod>
      <objMethod>
          <methodName>getTimeMark</methodName>
          <methodDescription>get the timeMark attribute value from tyrepackage</methodDescription>
          <methodAccess>public</methodAccess>
          <methodReturnDataType>int </methodReturnDataType>
     </objMethod>
     <objMethod>
          <methodName>setPackageSize</methodName>
```

<methodDescription>**set the maximum capacity of the tyrepackage**</methodDescription>
<methodAccess>**public**</methodAccess>
<methodParameters>
<pName>**intSize**</pName>
<pDataType>**int**</pDataType>
</methodParameters>
<methodReturnDataType>**void**</methodReturnDataType>
</objMethod>
<objMethod>
<methodName>**getPackageSize**</methodName>
<methodDescription>**get the capacity of the tyrepackage**</methodDescription>
<methodAccess>**public**</methodAccess>
<methodReturnDataType>**int** </methodReturnDataType>
</objMethod>
<objMethod>
<methodName>**setCompType**</methodName>
<methodDescription>**set the compType attribute of tyrepackage**</methodDescription>
<methodAccess>**public**</methodAccess>
<methodParameters>
<pName>**strType**</pName>
<pDataType>**String**</pDataType>
</methodParameters>
<methodReturnDataType>**void**</methodReturnDataType>
</objMethod>
<objMethod>
<methodName>**getCompType**</methodName>
<methodDescription>**get the compType attribute of the tyrepackage**</methodDescription>
<methodAccess>**public**</methodAccess>
<methodReturnDataType>**String**</methodReturnDataType>
</objMethod>
<objMethod>
<methodName>**isFull**</methodName>
<methodDescription>
return true if the number of components in this tyrepackage reaches the tyrepackage's capacity.
</methodDescription>
<methodAccess>**public**</methodAccess>
<methodReturnDataType>**boolean**</methodReturnDataType>
</objMethod>
<objMethod>
<methodName>**cleanPackage**</methodName>
<methodDescription>**delete the componens in the tyrepackage**</methodDescription>
<methodAccess>**public**</methodAccess>

```
            <methodReturnDataType>void</methodReturnDataType>
        </objMethod>
        <objMethod>
            <methodName>len</methodName>
            <methodDescription>get the number of components in the tyrepackage</methodDescription>
            <methodAccess>public</methodAccess>
            <methodReturnDataType>int</methodReturnDataType>
        </objMethod>
    </technicalDetails>
    <time>
        <timeStamp>
            <dataType> Integer (non-negative)</dataType>
            <scale>1:6min</scale>
        </timeStamp>
        <lookahead>
            <dataType> Integer (non-negative)</dataType>
            <scale>1:6min</scale>
        </lookahead>
    </time>
    <implementedObjectDetails>
        <companyName>NUS</companyName>
        <companyContactInfo>(65)68744366</companyContactInfo>
        <companyContactPerson>
            <firstName>na</firstName>
            <lastName>zhao</lastName>
        </companyContactPerson>
        <companyEmail>zhaona@comp.nus.edu.sg</companyEmail>
        <version>1.1</version>
        <versionDate>
            <day>25</day>
            <month>12</month>
            <year>2002</year>
        </versionDate>
        <implementationPlatform>Java</implementationPlatform>
        <referenceLink>nil</referenceLink>
    </implementedObjectDetails>
</objectModel>
```

Appendix C: DIF Specification for the MRT Train Object

```
<?xml version="1.0" ?>
  <objectModel>
    <modelName>MRT Train</modelName>
    <descriptionLink>nil</descriptionLink>
    <description>A model to emulate a train which carries numbers of passengers from current station to
      their destination station. The important features are the number of passenger going to each station and
      the capacity of the train.
    </description>
    <parentObj>nil</parentObj>
    <childObj>East Line Train</childObj>
    <childObj>West Line Train</childObj>
    <childObj>North Line Train</childObj>
    <childObj>South Line Train</childObj>
    <attrFieldDataType>XML DOM Tree</attrFieldDataType>
    <technicalDetails>
      <objAttr>
          <attrName>direction</attrName>
          <attrDescription>Stores the direction of the train. This is used to distinguish the train from 4 lines.
          </attrDescription>
          <attrAccess>Private</attrAccess>
          <attrDataType>String</attrDataType>
      </objAttr>
      <objAttr>
          <attrName>capacity</attrName>
          <attrDescription>Stores the max number of passenger which can be carried by the train.
          </attrDescription>
          <attrAccess>Private</attrAccess>
          <attrDataType>Int</attrDataType>
      </objAttr>
      <objAttr>
          <attrName>totalpsgnum</attrName>
          <attrDescription>Stores current number of passenger on the train.</attrDescription>
          <attrAccess>Private</attrAccess>
          <attrDataType>Int</attrDataType>
      </objAttr>
      <objAttr>
          <attrName>psgnum_to_EW1</attrName>
          <attrDescription>Stores passenger number whose destination is station EW1.</attrDescription>
          <attrAccess>Private</attrAccess>
          <attrDataType>Int</attrDataType>
```

```
</objAttr>
<objAttr>
    <attrName>psgnum_to_EW2</attrName>
    <attrDescription>Stores passenger number whose destination is station EW2.</attrDescription>
    <attrAccess>Private</attrAccess>
    <attrDataType>Int</attrDataType>
 </objAttr>
<objAttr>
    <attrName>psgnum_to_EW3</attrName>
    <attrDescription>Stores passenger number whose destination is station EW3.</attrDescription>
    <attrAccess>Private</attrAccess>
    <attrDataType>Int</attrDataType>
</objAttr>
<objAttr>
    <attrName>psgnum_to_EW4</attrName>
    <attrDescription>Stores passenger number whose destination is station EW4.</attrDescription>
    <attrAccess>Private</attrAccess>
    <attrDataType>Int</attrDataType>
</objAttr>
<objAttr>
    <attrName>psgnum_to_EW5</attrName>
    <attrDescription>Stores passenger number whose destination is station EW5.</attrDescription>
    <attrAccess>Private</attrAccess>
    <attrDataType>Int</attrDataType>
</objAttr>
<objAttr>
    <attrName>psgnum_to_NS1</attrName>
    <attrDescription>Stores passenger number whose destination is station NS1.</attrDescription>
    <attrAccess>Private</attrAccess>
    <attrDataType>Int</attrDataType>
</objAttr>
<objAttr>
    <attrName>psgnum_to_NS2</attrName>
    <attrDescription>Stores passenger number whose destination is station NS2.</attrDescription>
    <attrAccess>Private</attrAccess>
    <attrDataType>Int</attrDataType>
</objAttr>
<objAttr>
    <attrName>psgnum_to_NS3</attrName>
    <attrDescription>Stores passenger number whose destination is station NS3.</attrDescription>
    <attrAccess>Private</attrAccess>
    <attrDataType>Int</attrDataType>
 </objAttr>
```

```
<objMethod>
    <methodName>setAttrValueInt</methodName>
    <methodDescription>set int value to attribute in DOM Tree</methodDescription>
    <methodAccess>Public</methodAccess>
    <methodReturnDataType>void</methodReturnDataType>
    <methodParameters>
        <pName>AttrName</pName>
        <pDataType>String</pDataType>
        <pName>AttrValue</pName>
        <pDataType>int</pDataType>
    </methodParameters>
</objMethod>
<objMethod>
    <methodName>setAttrValueString</methodName>
    <methodDescription>set String value to attribute in DOM Tree</methodDescription>
    <methodAccess>Public</methodAccess>
    <methodReturnDataType>void</methodReturnDataType>
    <methodParameters>
        <pName>AttrName</pName>
        <pDataType>String</pDataType>
        <pName>AttrValue</pName>
        <pDataType>int</pDataType>
    </methodParameters>
</objMethod>
<objMethod>
    <methodName>extractIntValue</methodName>
    <methodDescription>extract attribue value as int from the DOM Tree</methodDescription>
    <methodAccess>Public</methodAccess>
    <methodReturnDataType>void</methodReturnDataType>
    <methodParameters>
        <pName>AttrName</pName>
        <pDataType>String</pDataType>
    </methodParameters>
</objMethod>
<objMethod>
    <methodName>extractStringValue</methodName>
    <methodDescription>extract attribute value as string from the DOM Tree</methodDescription>
    <methodAccess>Public</methodAccess>
    <methodReturnDataType>void</methodReturnDataType>
    <methodParameters>
        <pName>AttrName</pName>
        <pDataType>String</pDataType>
    </methodParameters>
```

```
    </objMethod>
    <objMethod>
        <methodName>updateobj</methodName>
        <methodDescription>extract attribute value as string from the DOM Tree</methodDescription>
        <methodAccess>Public</methodAccess>
        <methodReturnDataType>void</methodReturnDataType>
        <methodParameters>
            <pName>AttrName</pName>
            <pDataType>String</pDataType>
            <pName>num</pName>
            <pDataType>int</pDataType>
        </methodParameters>
    </objMethod>
  </technicalDetails>
  <time>
    <timeStamp>
        <dataType> Integer (non-negative)</dataType>
        <scale>1:1min</scale>
    </timeStamp>
    <lookahead>
        <dataType> Integer (non-negative)</dataType>
        <scale>1:1min</scale>
    </lookahead>
  </time>
  <implementedObjectDetails>
    <companyName>NUS SOC</companyName>
    <companyContactInfo>(65) 68744366</companyContactInfo>
    <companyContactPerson>
        <firstName>Na</firstName>
        <lastName>Zhao</lastName>
    </companyContactPerson>
    <companyEmail>nil</companyEmail>
    <version>1.1</version>
    <versionDate>
        <day>30</day>
        <month>8</month>
        <year>2002</year>
    </versionDate>
    <implementationPlatform>Java</implementationPlatform>
    <referenceLink>nil</referenceLink>
  </implementedObjectDetails>
</objectModel>
```