Proceedings of the 2003 Winter Simulation Conference
S. Chick, P. J. Sánchez, D. Ferrin, and D. J. Morrice, eds.

AUTOMOBILE MANUFACTURING SUPPLY CHAIN SIMULATION IN THE GRIDS ENVIRONMENT

Gary Tan
Na Zhao

School of Computing
National University of Singapore
3 Science Drive 2
Singapore 117543, SINGAPORE

Simon J. E. Taylor

Department of Information Systems and Computing
Brunel University
Uxbridge UB8 3PH, U.K.

ABSTRACT

A Supply Chain is the series of activities that an organization uses to deliver value to its customers. In today's competitive environment, the globalization of markets has rapidly substituted the traditional integrated business. The competitive success of an organization no longer depends only on its own efforts, but relies on the efficiency of the entire supply chain. Therefore, building an effective supply chain is fast becoming paramount in today's marketplace. Distributed Supply Chain (DSC) Simulation has been identified as one of the best means to test and analyze the performance of supply chains. The Generic Runtime Infrastructure for Distributed Simulation (GRIDS) is a middleware that supports the reuse and interoperation of DSC simulations. This paper reports the experience on employing the GRIDS to support the distributed collaboration of an automobile manufacture supply chain simulation. Several advantages of GRIDS are also discussed here which make it an ideal middleware for DSC simulations.

1 INTRODUCTION

A Supply Chain is the series of activities that an organization uses to deliver value, either in the form of a product, a service, or a combination of both, to its customers (Archibald 1999). In today's globalization environment, the intense competition has driven the traditional monolithic integrated business to a more efficient practice of outsourcing. The trend of globalization of markets has forced organizations to rely on numerous external firms or suppliers to deliver value to the ultimate customers. The competitive success of an organization no longer depends only on its own efforts, but relies on the efficiency of the entire supply chain. Therefore, building an effective supply chain is fast becoming paramount in today's marketplace.

The effort of managing and coordinating the activities between separate entities is referred to as Supply Chain Management (SCM). A supply chain is a complex system with large sources of uncertainties. As a result of the mix of outside firms, it is often difficult to know the impact of changes or poor performance on the supply chain. Therefore successful SCM requires a carefully defined approach to test and analyse the performance of the chain. Distributed Supply Chain (DSC) Simulation has been identified as one of the best means to perform what-if analysis on supply chains. It offers analysts and decision-makers a means to replicate the behaviors of complex systems. There is also a reduction of costs and time of building a new model, which is derived from the reuse of existing models. Therefore, companies can react faster to the global competition by using this approach to investigate efficiency in their supply chains and implement constant improvements on them.

The Generic Runtime Infrastructure for Distributed Simulation (GRIDS) is a middleware that supports the reuse of capabilities available in different simulations and the possibility of distributed collaborative development of a complex DSC simulation application (Sudra and Taylor, 2002). It is a lightweight component-based runtime infrastructure with extensible features and package interfaces. 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, training, transportation and so on.

In this paper, we report the experience on how to develop an automobile manufacture supply chain simulation and integrate it with the GRIDS environment. The GRIDS architecture and GRIDS Object Exchange Model Template (OEMT) are briefly introduced in Section 2. Section 3 explains the theory background (JIT Production theory) and provides a conceptual model of the automobile manufacture supply chain simulation. Section 4 of the article covers the development process of the simulation system in GRIDS environment. Section 5 reports and analyzes the execution results of the automobile manufacture simulation. The experience we get from employing GRIDS to support the distributed collaboration of DSC supply chain

simulation is presented in Section 6. This paper ends with the conclusion and future work in Section 7.

2 THE GENERIC RUNTIME INFRASTRUCTURE FOR DISTRIBUTED SIMULATION (GRIDS)

Because the most attractive advantages of the distributed simulation approach are its modularity and the ability to mix and match the platforms and simulations to provide a business solution, the interoperability problem must be solved. Probably the most effective approach would be to use common programming interfaces and standard protocols that provide a seamless manner of access to various simulations that reside on different platforms. Such standardized interfaces and protocols have come to be referred to as **middleware** infrastructure. 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. The Generic Runtime Infrastructure for Distributed Simulation (GRIDS) is such a middleware for reuse and interoperation of simulations.

2.1 GRIDS Basic Architecture

The 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 using a message-based communication scheme and to investigate issues in distributed simulation and how they impact on simulation methodology and practice. Instead of the static and fixed functionality advocated by the High Level Architecture (HLA) RTI specification, the 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. The HLA is a general purpose middleware architecture 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 simulation. The differences between HLA RTI and GRIDS are discussed in (Sudra and Taylor 2002). The extensibility is the principal difference between the 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 equivalent to the Central Runtime Component (CRC).
- Client: used by the federate to interact with the other federates in the federation (in GRIDS, as with 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). It is equivalent to the Local Runtime Component (LRC).
- Thin Agent: the GRIDS term which provides an extensible component service such as performance optimization, Time Management, Data Distribution Management and other special simulation services.
- Metadatabase: the general data structure in GRIDS to store information.

For a more detailed discussion of GRIDS and the thin agent, the reader is referred to (Sudra, Taylor and Janahan 2000a, 2000b) and (Taylor, Saville and Sudra 1999).

Figure 1 illustrates graphically the middleware's setup in a typical GRIDS federation. 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.

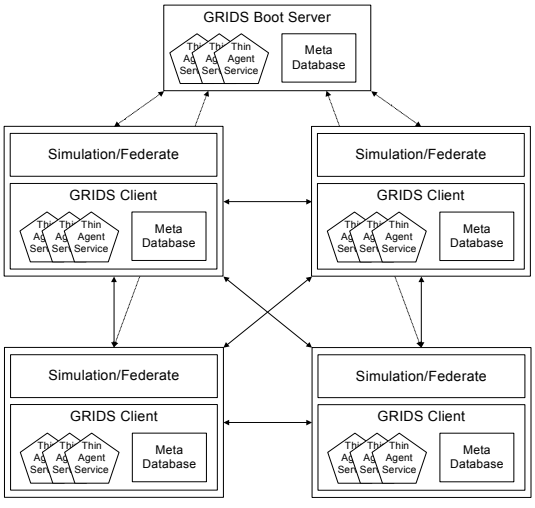

Figure 1: A Typical GRIDS Federation

2.2 GRIDS Object Exchange Model Template (OEMT)

In a distributed simulation application, each federate is standalone. The interoperation among them is realized through the exchange of information. Normally, the passing of information is built upon the message-passing concept. However, the message-passing is not enough to satisfy all the requirements of distributed simulation. In some cases, federates produce objects instead of messages. For example, in a DSC simulation, the interactions among federates are object instances that are produced, sent and received by the federates from one another. Employing object-passing in DSC simulation area meets this requirement

efficiently. It is easier to keep the details of the object and to establish a standard whereby objects can be recognized uniquely. In addition, there may be cases where variables are required to be input to the object to produce particular output or where the federate is interested in some intermediate information of the object which requires passing some parameters into the object before getting the results. The object-passing satisfies this requirement by allowing the provision of methods that the receiving federates may invoke to enter variables and get expected information.

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 text 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 focuses 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 aptly describe the exchange model, we introduced the Object Exchange Model Template (OEMT) of GRIDS.

2.2.1 OEMT

The GRIDS Object Exchange Model Template (OEMT) defines the format and syntax for recording information in GRIDS distributed simulation object models, as well as mandatory specific data that defines each model uniquely from others (Tan, Ng, and Taylor 2002). It is tasked to document object exchange models, which emulate as close to reality as possible, the actual entities passed between the simulation federates. The OEMT is also 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 need them.

2.2.2 OEMT Components

GRIDS Object Exchange Models are composed of a group of inter-related fields specifying information about the model. The template for the core of a GRIDS object exchange model shall use a tabular format for certain fields of the model, and shall consist of the following fields:

- 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.
- Name of Parent Object: catering to the inheritance characteristic of object-oriented programming, 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 mandatory object attributes and methods that must be implemented of this model and the optional object components that exist in complex model.
- For each object attribute, the name, description, accessibility, and the data type of the attribute are required.
- For each object method, the name, description, accessibility by the public, parameters if any (parameter name and data type), and the return data types of the method are required.
- If object contains certain components, for each object component, the name, description and attributes description of the component are required.
- Time Representation: to specify the timestamp and lookahead of the instances of the object which are used to synchronize all the federates in the federation. This field is described only when Time Management Thin Agent is used.
- 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 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.

To facilitate the internet transfer and storage of object exchange model, the OEMT also has an XML format to carry the foregoing information. The DTD schema of the OEMT is given in the Appendix.

3 AUTOMOBILE MANUFACTURE SUPPLY CHAIN MODEL

In this section, we describe the design of a typical DSC simulation, an **Automobile Manufacture Supply Chain Simulation,** in the GRIDS environment**.** This case study investigates the relationship between the suppliers and the automaker. An automaker factory and four supplier factories are developed to construct a simulation federation. The suppliers provide parts to the automaker based on Just-In-

Time parts delivery, so that the minimum inventory and on-the-wheel inventory can be achieved.

3.1 JIT Production Theory Overview

The Just-In-Time (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. As a result, suppliers are normally left with many weeks or even months of inventory. What is more serious is that the large inventory may not even ensure having the specific products the customers want. In small-batch production (JIT), by contrast, customers encourage suppliers to deliver only what is needed by the assembly plant at a particular time, even if this means partially filled trucks. Thus, products move rapidly through the suppliers' plant and to their customers, and suppliers maintain much less inventory. A small-batch production not only means lower inventory maintaining costs, but also brings lots of other important benefits. Firstly, suppliers can respond more quickly to changes in customers' demand. They can identify any defects in products quickly and thus have fewer potentially defective parts that need to be reworked (Liker and Wu 2000). Secondly, fewer people are needed to perform wasteful activities, such as moving large batches of inventory from place to place in the plant. Productivity rises via labour force saving.

3.2 Simulation 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. These five simulation models (federates) run separately and cooperate through the exchange of objects (product entities). The interaction relationship of the five semi-independent federates in the system is shown in Figure 2.

This system is designed based on the JIT Production theory. According to the JIT theory we introduced in section 3.1, 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

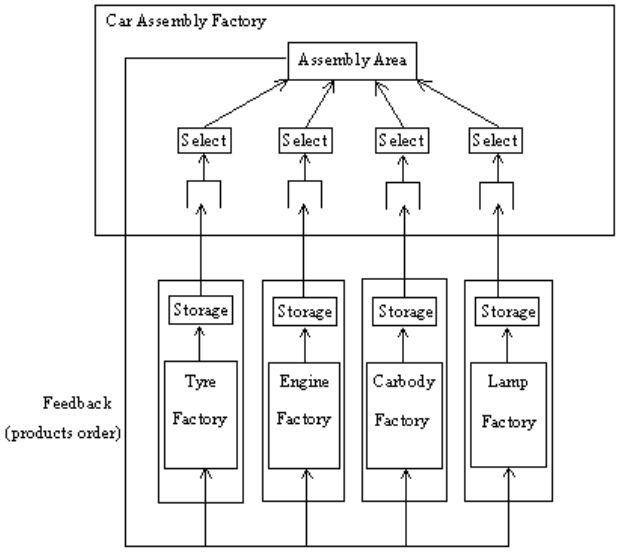

Figure 2: Models Interaction Relationship

Component Factory, when it receives 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 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 the Component Factory algorithm is as follows:

```
While simulation not terminated
  Waiting for next order (event)
  New order object arrive
  Extract the PartsNeeded and timeMark from
  order object
  Estimate current produce capability
  Produce parts
  Advance simulation time
  Deliver parts to Car Assembly Factory
  Decide production speed for next production
  cycle
  Adjust production speed
Endwhile
```

4 SIMULATION IMPLEMENTATION PROCESS

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 this section.

4.1 Object Exchange Model 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 them appropriately.

Tables 1 to 5 use the TyrePackage as an example to show the use of the OEMT in the system design. The TyrePackage object is published by the Tyre Factory Federate and is subscribed by the Car Assembly Factory. Each package contains many tyres. We can regard it as a truck which delivers tyres to the Car Assembly Factory.

The OEMT specifies the attributes, methods and other information of the TyrePackage object.

Table 1: Exchange Object Information

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.
Name of Parent Object	Nil
Names of Children Objects	Nil
Technical Details	……
Attribute Field Data Type	DOM Tree
General Details of each Implemented Object	……

Table 2: Exchange Object Attributes Table

Mandatory Object Attributes Table		
	Category	**Information**
1	Attribute Name	PackageSize
	Attribute Description	Max 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 3: Exchange Object Component Table

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 Tire Quality Grade (UTQG) labelling system is a rating for tread-wear, 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: Exchange Object Methods Table

Mandatory Object Methods Table			
	Category	**Information**	
1	Method Name	getCompNode	
	Method Description	get a component Node from Tyre-Package	
	Method Accessibility	public	
	Method Return Data Type	XML DOM Node	
2	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	
3	Method Name	getPackageSize	
	Method Description	get the capacity of the tyrepackage	
	Method Accessibility	public	
	Method Return Data Type	int	
4	Method Name	getTimeMark	
	Method Description	get the time mark of the TyrePackage object	
	Method Accessibility	public	
	Method Return Data Type	int	
5	Method Name	getCompType	
	Method Description	get the compType attribute of the tyre-package	
	Method Accessibility	public	
	Method Return Data Type	String	

Table 5: General Details of Exchange Object

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	

The Car Assembly Factory also subscribes to the EnginePackage, CarbodyPackage, LampPackage objects from the other three Component Factories and publishes order objects at the same time. The four children objects of the order object - TyreOrder, EngineOrder, CarBodyOrder and LampOrder are subscribed respectively by the Component Factories. The object specifications for the above exchange objects in this system are similar to that of the TyrePackage and are omitted due to the limited space. 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.

4.2 Exchange Object Implementation

To implement the object exchange model, the attributes of TyrePackage object take a unique hierarchical structure - Document Object Model (DOM) Tree. The DOM is a W3C product to facilitate XML file manipulation by parsing it into a computer-friendly tree structure (DOM 2002). As Figure 3 shows, the attributes and components described in OEMT are converted to the DOM Tree format.

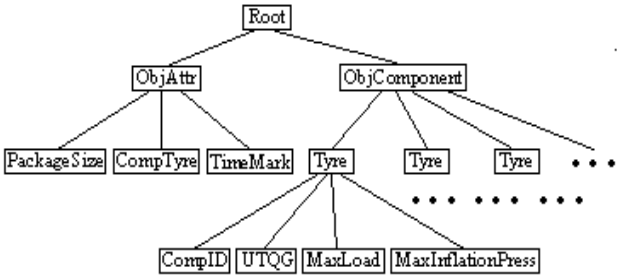

Figure 3: DOM Tree Framework

The value of each attribute or component can be accessed by invoking the methods of the object. 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.

4.3 Integration with GRIDS

To meet the interoperation requirement of the system, the 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 the 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. The Data Distributed Management thin agent is

employed to control the routes of objects transfer. Figure 4 shows the makeup of an individual federate.

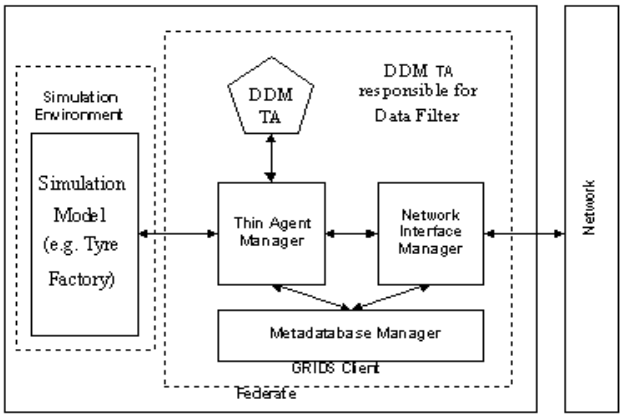

Figure 4: Federate Makeup with GRIDS Client

5 EXECUTION AND RESULT ANALYSIS

First of all, the GRIDS is initialized by starting the Boot Server. Then individual simulation nodes register to the boot server when the federate starts simulation through invoking GRIDS system functions "initSim" and "startSim". The boot server builds up the namespace of all the clients registering, and builds a central entity list of all the exchange objects in the simulation with the information of their publishers and subscribers. Upon a simulation "Start" event, the boot server broadcasts to all registered clients the entire entity list. The entity list is stored in the internal database on each GRIDS client. 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 then communicate directly as necessary in a peer to peer fashion to other nodes in the simulation.

For the setup, 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. 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. It was found that the Car Assembly Factory spent 97 hours and 12 minutes waiting for parts. The delay rate was 7.2%. This delay can be avoided by employing a safe stock in the Car Assembly Factory. Figures 5 and 6 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.

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

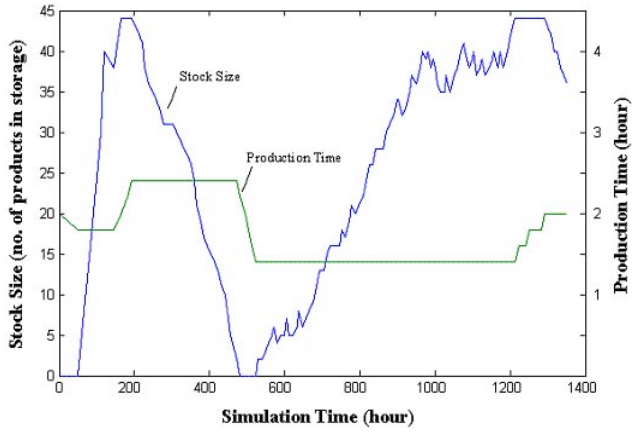

Figure 5: Carbody Factory Execution Result

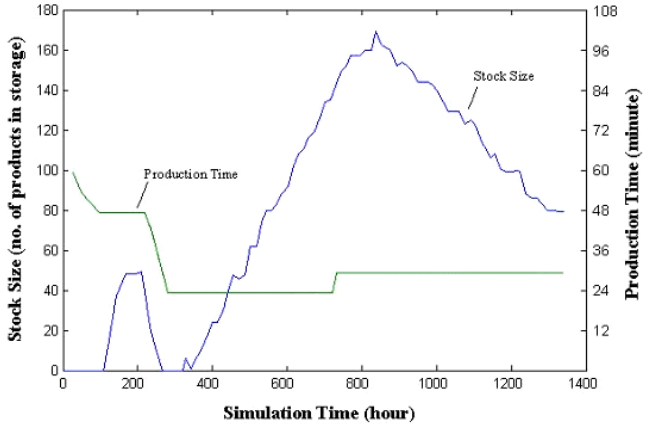

Figure 6: Tyre Factory Execution Result

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 5 and 6, 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 because 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 3000 is due to the fact that the defect rate of previous batch of tyres is abnormally high. This conclusion is made via analysis of the factory's production record.)

6 EXPERIENCE GAINED

6.1 GRIDS Federation Development Process

From the experience we obtained 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 might be more effective.

6.2 Benefits of Using GRIDS

The automobile manufacture simulation case study proved GRIDS as an alternative for HLA with simpler architecture and extensibility. The GRIDS has several advantages which make it an ideal middleware infrastructure for DSC simulations:

- 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 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 are easily and rapidly developed and dynamically added in.
- It reduces the costs and time for building a new model, which is derived from the GRIDS's ability to integrate an array of existing component-based simulation models.
- 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 development 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 a single node does not affect the execution of other nodes.

7 CONCLUSION AND FUTURE WORK

GRIDS, representing an early adopter of the component RTI philosophy, 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 (Sudra and Taylor 2002). It is a powerful infrastructure for implementing various types of simulation, especially DSC simulation due to its lightweight and ease of extensibility. This paper presented an implementation of an automobile manufacture supply chain simulation in the GRIDS environment. We have shown how to develop a component-based supply chain simulation and integrated it with GRIDS middleware. The experience gained from the case study paves the way for prospective users in using GRIDS in their simulations.

The current GRIDS and OEMT are built using the Java language. Our future work will realize interfaces to other programming languages, so that it can fit more simulation models from different organizations. An online web database, the Object Exchange Model Repository (OEMR) will also be developed as a central location where all object exchange models' information is stored to facilitate the reuse of object models.

APPENDIX

Below is the DTD schema for the OEMT:

```
<!ELEMENT objectModel (modelName, parentObjName?,
 childrenObjName*, modelLink|(description, technicalDetails),
 time?, attrFieldDataType?, implementedObjectDetails*)>
    <!ELEMENT modelName (#PCDATA)>
    <!ELEMENT parentObjName (#PCDATA)>
    <!ELEMENT childrenObjName (#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, semantics)>
            <!ELEMENT dataType (#PCDATA)>
            <!ELEMENT semantics (#PCDATA)>
        <!ELEMENT lookahead (dataType, semantics)>
            <!ELEMENT dataType (#PCDATA)>
            <!ELEMENT semantics (#PCDATA)>
    <!ELEMENT attrFieldDataType (#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)>
```

REFERENCES

Archibald,G., N. Karabakal, N. and P. Karlsson, P. 1999. *Supply Chain vs. Supply Chain: Using Simulation to Compete Beyond the Four Walls*. Proceedings of the 1999 Winter Simulation Conference, 1207-1214.

Document Object Model (DOM) Level 3 Core Specification Version 1.0. 2002. W3C Working Draft 22 October 2002.

Liker, J.K. and Y. Wu. 2000. Japanese Automakers, U.S. Suppliers and Supply-Chain *Superiority*, Sloan Management Review, Fall 2000. 81-93.

Sudra, R., S.J.E. Taylor and T. Janahan. 2000a. *Distributed Supply Chain Management in GRIDS*, proceedings of the 2000 Winter Simulation Conference. Florida, USA. J. A. Joines, R. R. Barton, K. Kang, and P. A. Fishwick (eds.) USA ACM Press New York. 356-361.

Sudra, R., S.J.E. Taylor and T. Janahan. 2000b. *GRIDS: A Novel Architecture for Distributed Supply Chain Management,* proceedings of the Fall 2000 Simulation Interoperability Workshop. Simulation Interoperability Standards Organization, Institute for Simulation and Training. Florida. 00F-SIW-051, 2000.

Sudra, R., S.J.E. Taylor. 2002. *Extensibility: Modular HLA RTI Services,* proceedings of 2002 European Simulation Interoperability Workshop (E-SIW), North London, UK, Jun 2002. 02E-SIW-049.

Tan, G., W.N. Ng, and S. Taylor. 2002. *Developing An Object Exchange Model Template (OEMT) for GRIDS Distributed Supply Chain (DSC) Simulations*, proceedings of 35th Annual Simulation Symposium, San Diego, USA.

Taylor, S.J.E., J. Saville, R. Sudra. 1999. *Developing Interest Management Techniques in Distributed Interactive Simulation Using Java*, proceedings of the 1999 Winter Simulation Conference, P. A. Farrington, H. B. Nembhard, D. T. Sturrock, and G. W. Evans, eds. 518-523.

AUTHOR BIOGRAPHIES

DR GARY TAN SOON HUAT is a Senior Lecturer at the School of Computing, National University of Singapore (NUS). His research interests include parallel and distributed computing, scheduling and load balancing, declarative multiprocessors, dataflow and parallel machine simulation, parallel and distributed (interactive) simulation and High Level Architecture. He is a member of the NUS Modeling and Simulation Group (MSG), and his email address is <gtan@comp.nus.edu.sg>

NA ZHAO is currently doing her Masters in the National University of Singapore (NUS). She received her Bachelor in Computer Science from Beijing Normal University, PRC. Her research area is in the field of distributed simulation. Her research interests include distributed and parallel systems, data base, and networking.

DR SIMON J.E. TAYLOR is the Chair of the Simulation Study Group of the UK Operational Research Society. He is a Senior Lecturer in the Department of Information Systems and Computing and is a member of the Centre for Applied Simulation Modeling, both at Brunel University, UK. His main research interests are distributed simulation and applications of simulation health care. His email address is <simon.taylor@brunel.ac.uk>