Proceedings of the 2000 Winter Simulation Conference
J. A. Joines, R. R. Barton, K. Kang, and P. A. Fishwick, eds.

DISTRIBUTED SUPPLY CHAIN SIMULATION IN GRIDS

Rajeev Sudra
Simon J.E. Taylor
Tharumasegaram Janahan

Centre for Applied Simulation Modelling
Department of Information System and Computing
Brunel University, Uxbridge
UB8 3PH, UK

ABSTRACT

Amongst the majority of work done in Supply Chain Simulation, papers have emerged that examine the area of model distribution. The executions of simulations on distributed hosts as a coupled model require both coordination and facilitating infrastructure. A distributed environment, the Generic Runtime Infrastructure for Distributed Simulation (GRIDS) is suggested to provide the bonding requirements for such a model. The advantages of transparently connecting the distributed components of a supply chain simulation allow the construction of a conceptual simulation while releasing the modeler from the complexities of the underlying network. The infrastructure presented demonstrates scalability without loosing flexibility for future extensions based on open industry standards.

1 INTRODUCTION

Distributed simulations most frequent area of application is currently in military simulation. Realization of these techniques into the commercial sector are slowly forthcoming. Simulationists can benefit from using this type of technology by connecting existing models together, reducing efforts of re-coding and pushing for model reuse. Internal details of models can also be protected by providing proprietary interfaces during collaborative projects. However, investigation into the types of infrastructures that can support disparate model types and allow transparency for the entities of a typical coupled model are forthcoming.

Zeigler et al. (1999) discusses the use of DEVS/CORBA as a distributed execution environment for supply chain simulation. The use of CORBA acts as the linking infrastructure between the distributed components of the simulation. CORBA, a middleware standard promoted by the Object Management Group (OMG) for connecting distributed components of software, actually performs a very similar role to our own extensible infrastructure, GRIDS (Saville and Taylor 1998, Taylor et al. 1999).

This paper is structured as follows. In section 2 we review the techniques and reasons for simulating supply chains. In section 3 we present the Generic Runtime Infrastructure for Distributed Simulation (GRIDS), our contribution to the field of distributed simulation, and discuss the extensible features of the infrastructure as well as a description of the package interfaces. Section 4 gives a case study illustrating how the distributed service mechanism can be applied in supporting a coupled simulation. The paper ends with some conclusions in section 5.

2 SUPPLY CHAIN MANAGEMENT

Supply Chain Management (SCM) is the series of activities that an organization uses to deliver products, services or a combination of both to its customers. Recent opinions coupled with a shift in modern economics have shown a de-emphasis in the benefits of vertical integration such as economies of scale towards a focus on the benefits reaped from being specialized.

Supply Chain Simulations (referred to as SCS in this paper) are implemented in order to observe how the supply chains perform. Observations are made on processes such as customer demand planning and production logistics. Archibald et al. (1999) present's a more comprehensive list of the processes that, when optimized, are suggested to yield a successful supply chain configuration. Simulation provides one mechanism for finding these optimizations.

A supply chain of a given organization initially describes the processes that occur internally but must also consider the activities of both the suppliers (inputs) and the customers (outputs) as part of the total supply function. Actual supply chains usually involve multiple sites and often different partners are located at disparate positions geographically resulting in a distribution of models that

require aggregation. These distributed supply chain models can be linked together and organized into complete chains covering source materials through to end customers as has been demonstrated by Zeigler et al. (1999). The paper goes on to suggests that the distribution of a supply chain simulation can sometimes benefit from parallel execution as well basing their foundation on DEVS, a discrete event formalism. However, for any performance improvement and also for coordination, parallel and distributed execution is partially dependent on some type of time synchronization mechanism (Fujimoto 1999a), which we address in a later section as a feature of our execution environment.

3 REVIEW OF GRIDS

GRIDS has already been seen to support Distributed Interactive Simulation (Taylor et al. 1999). It is an execution environment capable of supporting a broad range of simulation types. The infrastructure's primary function is to coordinate the activities of distributed components with additional functionality via the use of a novel service distribution model known as Thin Agents.

GRIDS is built using the Java language (Arnold and Gosling 1996). Java was chosen since it offers a large set of features appropriate to distributed systems construction, figure 1 describes the protocol layers present in a typical GRIDS implementation. In particular, Java provides the facility to load objects from across the network using the dynamic class loader. Java also allows deployment of its code across most of the popular operating systems ensuring an increased potential user base and allows connectivity of simulations running on several platforms.

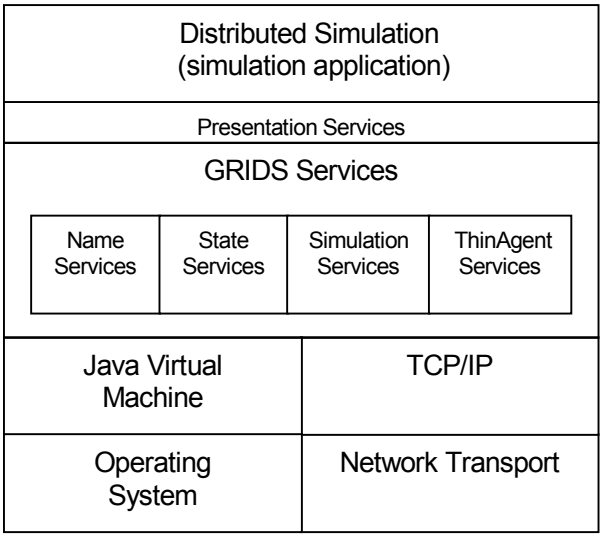

Figure 1: Protocol Layers

Simulations require services to both supplement and enhance execution. The GRIDS extensible service architecture is realized by thin agents. These agents may be used to support the simulation by providing tasks such as optimizations and assistance. Figure 2 provides an illustration of a typical GRIDS coupled model. Simulation objects/federates are connected to a GRIDS client via the published interface. Thin agents that are distributed to participating clients are instantiated to provide the required services. A description of the execution model and the client and thin agent interface is now examined.

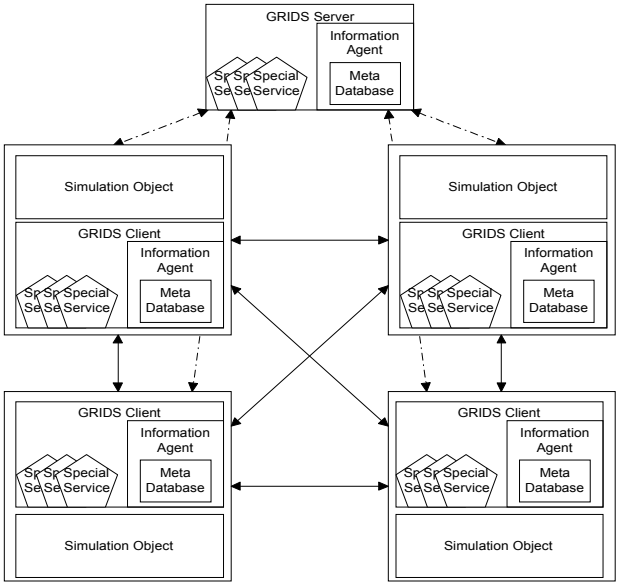

Figure 2: GRIDS Coupled Model

3.1 GRIDS Execution Model

A GRIDS session has three distinct stages of execution: *Register, Broadcast, and Run.*

3.1.1 Stage 1: 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 the namespace of all the clients registering, and constructs a central entity list of all entities in the simulation. Once all clients are registered the server closes all incoming connections for registration.

3.1.2 Stage 2: 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.

3.1.3 Stage 3: 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 will terminate and cease to be a part of the simulation. The clients communicate directly as necessary in a peer to peer fashion. The GRIDS client is responsible for "ticking" the host application to perform a simulation cycle, and for synchronizing entity attributes between the local and remote nodes.

3.2 GRIDS Client Interface

In this section, we describe how a federate within the coupled simulation model would connect to the GRIDS client infrastructure. The node would need to implement the GridsRunnable interface in order to allow GRIDS to call appropriate methods within the execution loop and process incoming messages.

```
GridsClient {
     void setValue (string name, object Value)
           throws GridsException;
     object getValue (string name) throws
           GridsException;
     void registerServer (GridsServer Server,
           string EntityName, gridsRunnable
           callback) throws GridsException;
     void registerAgent (string TAClassname)
           throws GridsException;
     void sendMessage (string Entity, GridsMessage
           message) throws GridsException;
     void timeAdvance (double Time) throws
           GridsException;
     double timeElapsed ()throws GridsException;
}

interface GridsRunnable {
     void processIncomingMessage (GridsMessage
           message) throws GridsException;
     void processExecutionLoop () throws
           GridsException;
}
```

The published methods getValue and setValue provide the API calls to modify and retrieve published entity state properties visible across the coupled model. Where a value exists on a different node, the GRIDS infrastructure will transparently either retrieve the value from the relevant node, or via a supplied thin agent. The registerServer method registers the node onto the supplied "Boot Server" and also registers within the GRIDS client infrastructure, the object class responsible for processing incoming messages and implementing the execution loop. Thin agents are registered by the node using the registerAgent method. The thin agents will be transparently sent to the Boot Server during the Registration Phase. Sending of messages is provided by the sendMessage method. Messages are posted direct to the receiving client where it will be placed within an internal mailbox for processing by the GRIDS client. Time synchronization control of the nodes is provided by the timeAdvance and timeElapsed methods.

The interface GridsRunnable is implemented by the application to provide GRIDS with two handlers for processing the application loop, and processing incoming messages. The processExecutionLoop is called by GRIDS on a set interval to process a single simulation loop. The processIncomingMessage method would be called by GRIDS whenever a message is received by a node.

3.3 Thin Agent Interface Model

This section describes the interface that needs to be adopted by designers of thin agents for use within a GRIDS session.

```
abstract ThinAgent {
     void initialise (GridsClient) throws
           GridsException;
     object getAttributeValue (string Name) throws
           GridsException;
     void setAttributeValue (string Name, object
           value) throws GridsException;
     void startLoop () throws GridsException;
     void endLoop () throws GridsException;
     void sendMessage (GRIDSMessage m);
     void receiveMessage (GRIDSMessage m)
}
```

A thin agent service needs to implement the above-published interface in order to be used by the GRIDS client. Providers of thin agent services need not implement all the above methods, but rather only implement a subclass of the generic thin agent class, and override the relevant methods allowing development of a taxonomy of thin agent types. Figure 3 describes the process taken to deploy a thin agent service from sub-classing through to instantiation on to the target client.

The initialize method is called by GRIDS when instantiating a thin agent for use by a client. Thin agents can register themselves within the MetaDatabase to handle specific simulation attributes. The GRIDS client calls the getAttributeValue and setAttributeValue methods whenever the registered data value owned by the thin agent is accessed by the simulation application. The startLoop and endLoop methods are called at the start and end of every execution loop to allow designers of thin agents to be able to perform any processing required at these stages. sendMessage is provided to allow the agent to communicate with a remote node.

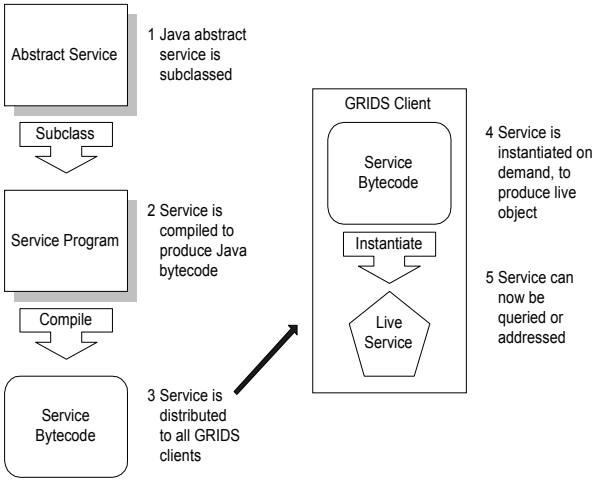

Figure 3: Thin Agent Mechanism

4 CASE STUDY

The case study discussed here focuses on the ability of thin agents to assist the coupling of a distributed model. Specifically, thin agents are used to coordinate the time synchronization of the coupled model.

Our scenario is loosely based on the Global Food Manufacturing case study described by Archibald et al. (1999). The model is composed of supply chain simulations that require to be connected into a linked model. GRIDS is employed as the middleware infrastructure. The infrastructure's primary function is to provide execution context and simulation data services.

An infrastructure to support a supply chain simulation would need to support the following requirements:

- An event list.
- A simulation clock that is advanced to the time of the next scheduled event on each iteration of the simulation.
- State variables containing the state of the simulation.

In order to implement simulation support of the supply chain simulations within the GRIDS infrastructure, we extended the interface to the GRIDS client to provide methods for supporting Discrete Event Simulation. This resulted in a specific client interface designed for DES in a similar fashion to proprietary client development. This section describes the synchronization role of the thin agent and extending the client interface to work with a SCS coupled model.

4.1 Synchronization Thin Agent

One of the main tasks of the thin agent is in the synchronization of events across the coupled model. To exemplify this approach we use conservative parallel simulation. Fujimoto (1999a, 1999b) provides a detailed discussion of this field.

Synchronization thin agents that are distributed to all participating simulations coordinate consistent time advancement of the coupled model. The thin agents control access to the two types of event queues, incoming and outgoing. Events are always sent with non-decreasing time stamps ensuring arrivals are queued in the order that they are sent. The thin agent is able to send event messages both locally as well as remotely. This process ensures that locally generated events are also processed in the correct order avoiding local causality errors.

Figure 4 illustrates how the model is arranged. The thin agents control the passing of events to the simulation by taking the next event with the lowest timestamp from its connected incoming queues and internal queue in line with the conservative approach. The local clock is advanced to the timestamp of that event and the simulation processes it. This processing may generate an additional event (either local or remote) which is handed back to the thin agent to be scheduled. If the generated event is local, it is redirected back into its own event link queue, otherwise it is sent to the remote client's thin agent link queue. Outgoing events are scheduled with time stamps avoiding the generation of causality errors for the target simulation.

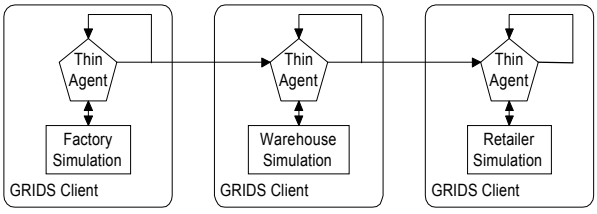

Figure 4: Event Queues

Figure 5 shows a more detailed view of the thin agent. Although only one incoming queue is shown here, there can actually be more with each one holding time ordered events from the same or different source simulation. Fujimoto's description of null messages is used to avoid deadlock. This algorithm is implemented in the thin agent. Time stamps in the null messages are used to advance the simulations. The null message algorithm introduces a property known as lookahead that is utilized by virtually all conservative synchronization algorithms. Lookahead is used to generate the time stamps of null messages scheduled in the future and is further discussed by Fujimoto (1999b).

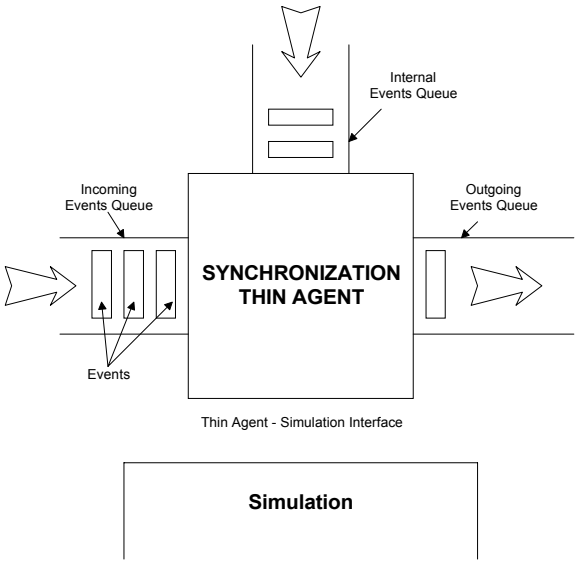

Figure 5: Synchronization Thin Agent

4.2 GRIDS Proprietary Interface to support SCS

To implement the synchronization thin agent, we needed to extend the GRIDS client interface to add support for requirements of SCS. A description of the additional methods is presented.

```
gridsSCSClient extends gridsClient{
     void registerServer
           (gridsServer Server,
           string EntityName, gridsSCSRunnable
           callback) throws GridsException;
     void scheduleEvent
           (string Entity, SCSEvent event, double
           scheduleTime) throws GridsException;
}

interface gridsSCSRunnable {
     void processEvent (SCSEvent event) throws
           GridsException;
}

class SCSEvent {
     Long EventUID; //unique number for the event
     String targetName;
     double scheduleTime
     String Value;
}

SCSAgent extends ThinAgent{
     void scheduleEvent
           (string Entity, SCSEvent event, double
           scheduleTime) throws GridsException;
}
```

The registerServer() method is modified to only allow a callback class that implements the gridsSCSRunnable Java interface. The method scheduleEvent() within the GRIDS interface is a proxy method used to invoke the SCSAgent, a specialized version of our ThinAgent class. Events dispatched by this thin agent increment a local counter. These counters can be matched globally to check if there are any events that have not arrived into event queues, these are known as transient messages (Fujimoto 1999a). When an event needs to be processed the infrastructure calls the processEvent() method in the application with the event as the parameter.

This description of the extended interface is actually a move towards a proprietary supply chain simulation client. This involves taking the generic GRIDS client and adding the required behaviors for the given simulation type. It can be seen that several different types of interface can be generated and specifically applied to a range of simulation types. This feature allows connectivity of simulations without compromising internal structure and allows for additional services not directly catered for in the core GRIDS distribution. A major goal of GRIDS is extensibility and this is provided in part by using thin agents and also by subclassing the GRIDS client.

5 CONCLUSIONS

This paper has reviewed the need for supply chain simulations to aid the decision processes involved in its management. The distribution of simulations as a result of mutually exclusive development has created a demand for building connecting infrastructure. This replaces the need to rebuild simulations and rather relies on the use of existing models. Additionally, model reuse is promoted reducing the effort of redevelopment. We have introduced a novel paradigm for implementing the middleware technology in GRIDS and provided another example of its extensible services architecture using mobile objects. The potential exists for organizations to publish proprietary interfaces, partly to protect models and also to allow easier integration of stakeholder's simulations with other models.

This contribution primarily demonstrates an alternative approach to distributed simulation. It serves as an exemplar of GRIDS support for a range of simulation types. The works continued use of Java as the implementation vehicle serves to further evaluate its capabilities in the distributed simulation area. Issues surrounding speed, usability and reliability are the focus here and feedback from our research group is available on request. More so, it is our intention to prototype possible architectures. If this kind of architecture was to meet with success then it is possible that a high speed implementation could be implemented in C++ or possibly future versions of Java (with optimization). This work is continuing through further development of the infrastructure and its applicability to problem domains other than DES (web-based simulation and our continued efforts in DIS). Examples of the GRIDS implementation are available from <www.brunel.ac.uk/ research/casm>.

REFERENCES

Archibald, G., N. Karabakal, and P. Karlsson. 1999. Supply chain vs. supply chain: Using simulation to compete beyond the four walls. In *Proceedings of the 1999 Winter Simulation Conference*. P.A.. Farrington, H.B. Nembhard, D.T. Sturrock, and G.W. Evans, eds, pp 1207-1214. Phoenix, AZ

Arnold, K. and J. Gosling. 1996. *The java programming language*. Addison-Wesley, USA.

Fujimoto, R.M. 1999a. Parallel and distributed simulation. In *Proceedings of the 1999 Winter Simulation Conference*. ed, P.A. Farrington, H.B. Nembhard, D.T. Sturrock, and G.W. Evans, 122-131.

Fujimoto, R.M. 1999b. *Parallel and distributed simulation systems (Wiley series on parallel and distributed computing)*. John Wiley & Sons, New York, NY.

Saville, J. and S.J.E Taylor. 1997. Interest management: dynamic group multicasting using mobile java policies. In *Proceedings of the Fall 1997 Simulation Interoperability Workshop*. Orlando, FL, USA.

Taylor, S.J.E., J. Saville, and R. Sudra. 1999. Developing interest management techniques in distributed interactive simulation using java. In *Proceedings of the 1999 Winter Simulation Conference*. ed, P.A. Farrington, H.B. Nembhard, D.T. Sturrock, and G.W. Evans, 518-523.

Zeilger, B.P., D. Kim, and S.J. Buckley. 1999. Distributed supply chain simulation in a DEVS/CORBA execution environment. In *Proceedings of the 1999 Winter Simulation Conference*. ed, P.A. Farrington, H.B. Nembhard, D.T. Sturrock, and G.W. Evans, 1333-1340.

AUTHOR BIOGRAPHIES

RAJEEV SUDRA is a Ph.D. candidate in the Department of Information Systems and Computing at Brunel University, UK. He received his B.Sc. in Computer Science and Economics also from Brunel University. He has gained much experience working in industry ranging from distributed systems software development to designing and deploying large-scale computer networks. His research focuses on interoperability issues with simulation and agent-based systems.

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 Modelling, both at Brunel University, UK. He was previously part of the Centre for Parallel Computing at the University of Westminster. He has an undergraduate degree in Industrial Studies (Sheffield Hallam), a M.Sc. in Computing Studies (Sheffield Hallam) and a Ph.D. in Parallel and Distributed Simulation (Leeds Metropolitan). His main research interests are distributed simulation and applications of simulation health care. He has also been known to occasionally tread the boards.

THARUMASEGARAM JANAHAN is a Ph.D. candidate in the Department of Information Systems and Computing at Brunel University, UK having completed a B.Sc. there. His research has focused on Distributed Interactive Simulation and Parallel and Distributed Simulation. He has previous industrial experience in Defense, Banking and Engineering.