Modular HLA RTI Services: The GRIDS Approach

Simon J.E. Taylor
Rajeev Sudra
Centre for Applied Simulation Modelling
Department of Information Systems and Computing
Brunel University, Uxbridge UB8 3PH
UNITED KINGDOM
{simon.taylor, rajeev.sudra}@brunel.ac.uk

Abstract

The Generic Runtime Infrastructure for Distributed Simulation (GRIDS) has been developed to investigate modularity issues in distributed simulation. It could be argued that although the HLA RTI is a widespread solution to distributed simulation, it cannot include all possible services. This paper investigates an approach to extending the distributed simulation services available in the HLA RTI. One example of this is bridging support for HLA/DIS legacy integration. This paper therefore presents GRIDS, how GRIDS can be used to provide modular service support for the HLA RTI, and a case study on legacy integration to demonstrate our approach.

1. Introduction

The standardization of the HLA [1,2,3] has been an important step for the distributed simulation community. This has brought about a level of stability with regards to the API [2] as described by the interface specification and allowed developers to build federations with a degree of certainty regarding interoperability. Additionally, this has stimulated continued development in RTI to support distributed simulation.

Generally, current implementations of the HLA RTI do not support the integration of extra services. These new services could be driven through changing requirements or the need for a feature not directly supported by the RTI. Currently this is dealt with by provision of bespoke changes to the RTI middleware or to the federates. The work presented in this paper investigates an approach to an RTI based on modularity. The implementation is known as the Generic Runtime Infrastructure for Distributed Simulation (GRIDS). The feature which distinguishes GRIDS from other RTI is its support of modularity through its extensibility mechanism. This mechanism provides the facility to add additional services within the middleware that (for largely historical reasons) we call *thin agents*. Thin agents are service components used to realize new services. To demonstrate the feature of extensibility within an RTI, a legacy integration service in GRIDS is developed to show how our infrastructure could provide a standard “bridge” between HLA and legacy DIS federations.

This paper is structured as follows. In section 2 we briefly discuss the functionality provided by current RTIs and suggest possible benefits accrued from a modular RTI. GRIDS, our approach to this type of RTI is presented in section 3. Section 4 presents our case study. The paper ends with some conclusions in section 5.

2. Current RTI Functionality

The services described within the HLA address a range of simulation requirements. Examples of these include the integration of simulations based on different time schemes by the time management service group and relevance filtering as provided in part by the data distribution management group. The RTI implementations such as the pRTI 1516 [4] from Pitch AB or the DMSO RTI represent fully compliant middleware. However, their implementations do not provide the facility to add additional functionality by federation developers as it is not a requirement of the HLA standard or (at the time of the RTI development) not an end user requirement.

As a standard it is accepted that a HLA compliant RTI need only implement required services. The HLA API has been carefully designed to provide simulation specific functionality for distributed simulation. Furthermore, the HLA is seen as a generalized architecture to facilitate reuse and interoperability for a variety of simulations. Generally, the "variety" is seen to come from the defense community but the hope has been to encourage HLA technology into non-defense applications. A greater acceptance of the HLA by all communities creates experiences in best practice and an increase in exposure and familiarity to the HLA approach to distributed simulation.

The greatest barrier to using an architecture such as the HLA and the RTI is its complexity [5]. There has certainly been increased interest in non-defense related uses of HLA alongside the required uptake from the defense communities. However, the lack of flexibility within the RTI to support additional or alternative services is seen as further barrier to particular types of applications. A modular RTI could provide the following benefits:

- Vendors could develop RTI components to deliver the services provided by the six service groups. A federation developer could mix and match service components from various vendors providing a compliant yet optimized RTI. The developers could choose from components implementing superior algorithms for example.
- RTI developers could incrementally upgrade their software through releases of components while maintaining a stable and core basic RTI.
- Fundamental changes to the HLA standard will impact heavily on the RTI. However, through a component enabled RTI, new service components will replace the older ones. Furthermore, the new compliant RTI can be assembled very quickly from the components.

Leading on from this discussion, our approach to a modular, component-based RTI is now examined.

3. The Generic Runtime Infrastructure for Distributed Simulation

The Generic Runtime Infrastructure for Distributed Simulation is an extensible middleware created by staff at the Centre for Applied Simulation Modelling, Brunel University, for research into distributed simulation tools and techniques. Instead of the fixed functionality of the HLA RTI, GRIDS provides basic functionality for the interoperation of federates within a federation (interaction of models in a distributed simulation) and a mechanism to add extra functions where appropriate.

This extensibility is provided by *Thin Agents* that encapsulate additional services as and when required (dead reckoning [6], different time synchronization management algorithms [7,8,9], message filtering, security, translation, etc.). The basic functionality is a subset of the HLA RTI functionality. The extensibility is the principal difference between GRIDS and other approaches to distributed simulation. We address the integration of GRIDS and the HLA RTI in [10].

3.1 Basic architecture

GRIDS is based on message-oriented middleware. The middleware is composed of the following major elements:

- **Boot Server** The boot server is a single process used to coordinate the initialization, execution and termination of a distributed simulation. The boot server is started and serves as a central location for all the participants of the simulation to connect to. The boot server is also loaded with any configuration information required to be passed to the connecting simulations. It is equivalent to the Central Runtime Component (CRC).
- **Client** The GRIDS client is used by the federate to interact with the rest of the federation. The client initially is used to contact the GRIDS server and register its intention to participate in a simulation exercise. The client is loaded with configuration data where it is able to communicate directly with other clients during the execution of the federation. It is equivalent to the Local Runtime Component (LRC).
- **Thin Agent** A thin agent is the GRIDS term for a component service. Thin agents are coded to provide a service or range of services. Their specific function is entirely dependent on the requirements of the application and therefore can be application specific services or more general. The thin agents are loaded by the boot server and distributed to connecting clients prior to the execution of the simulation. The thin agents are then integrated into the client augmenting its deployed services.
- **Metadatabase** The metadatabase is the general data structure in GRIDS used to store information. The data stored commonly includes both infrastructural and application data but is capable of storing any conceivable data structure.

Figure 1 illustrates a typical GRIDS federation. Simulation objects/federates are connected to a GRIDS client via an interface. Thin agents are distributed to participating clients and instantiated to provide the extra services.

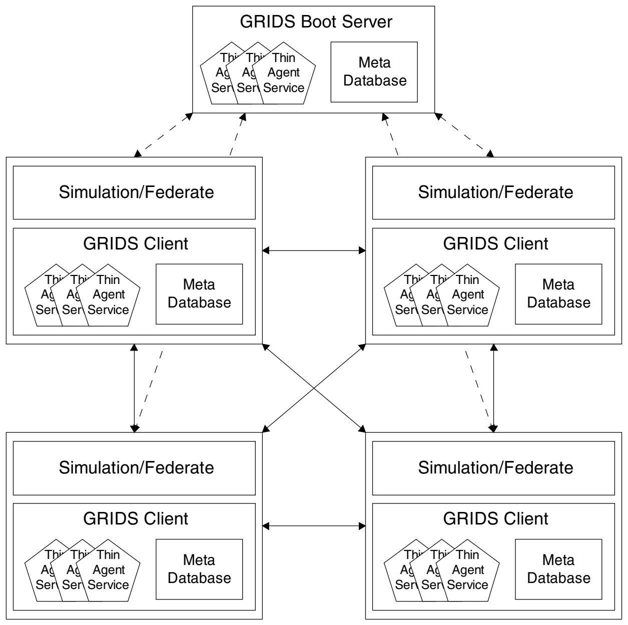


Figure 1. Typical GRIDS federation

A GRIDS session has five distinct stages of execution: *Initialization*, *Register, Broadcast, Runtime and Terminate.* Based on the stages, GRIDS can communicate in two modes, client-server (between the boot server and the client) and peer-to-peer (client to client). These stages of execution based on a typical GRIDS execution session are described.

- Stage 1: 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.

- Stage 2: 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 clients registering, 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.

- Stage 3: 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.

- Stage 4: 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 to other nodes in the simulation. The GRIDS client is responsible for synchronizing entity attributes between the local and remote nodes.

- Stage 5: Terminate

Once the simulations have completed executing, the clients register back with the boot server signaling that they are exiting gracefully from the federation.

The structure of a GRIDS node is summarized in Figure 2. Network access is controlled by the network interface manager. This is composed of several services that manage the distribution for the rest of the client. The thin agent manager is used to delegate jobs to thin agents. These are based on requests originating both from the network and the federate. Service conducted by the thin agent is specific to its implementation. The metadatabase is a general storage facility used by both managers to store relevant information. Finally, the thin agent manager is connected to the federate via an interface (described in the next section). Data is transmitted and received by the federate through this interface using the GRIDS middleware.

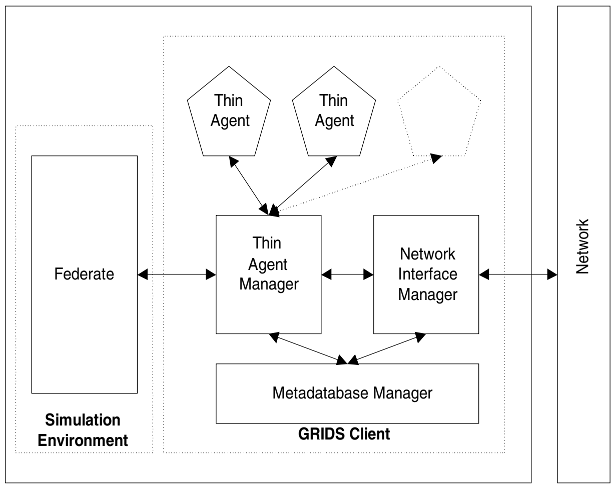


Figure 2. Composition of a GRIDS Client and Federate

3.2 Extensibility mechanism

The key feature that distinguishes GRIDS from other RTI is that it supports extensibility. This property allows the functionality of the infrastructure to be augmented. Extensibility is provided within GRIDS at several levels:

1. User-defined message types to transport data and for general communication.
2. API extensions via subclassing of the GRIDS interface.
3. Functionality extensions via thin agent components.
4. Internal data storage extensions via the MDB interface.
5. Modular internal structure based on runtime bindings.

Of the items above, those of interest are the first three in the context of this paper. As a message-oriented middleware, GRIDS relies heavily on message types to communicate. Messages represent a low level interaction primitive and therefore provide scope to abstract more complex forms of communication upon this foundation. GRIDS contains a series of internal message types. However, developers are able to add user-defined message types for their own applications.

The application interface within GRIDS is in the form of two basic object-oriented interfaces that must be implemented during an applications development. The SimInterface is equivalent to the federate ambassador. It defines a single method which is used to transfer data into the federate such as a callback. The GridsInterface is equivalent to the RTI ambassador. It defines a single method again used to transfer data into the middleware. A developer is expected to implement the SimInterface and provide all packing and unpacking of GRIDS messages. It must also provide an equivalent interface to match the calls made by the federate. This could be as simple as implementing just the single defined method or require a series of methods dealing with individual message types allowing a high degree of customization of the API presented by the middleware.

Thin agents encapsulate specific behavior and control providing each federation with a set of precise services (as defined by the component developer) during execution. As dynamically distributed and instantiated entities, thin agents provide a versatile and standard way for developers to provide specific services, without having to rewrite any code within the GRIDS RTI.

Developing a user-defined thin agent follows the process illustrated in Figure 3. Creation of every thin agent begins with an abstract service. This service is defined by the thin agent superclass interface. The thin agent is written as a service program based on the abstract service. This involves the implementation of the thin agent interface and superclass by a service program as source code. The service program is then compiled into a machine independent executable format known as Java bytecode. The service bytecode is ready to be distributed from the central GRIDS boot server to clients during an execution of a federation. The GRIDS clients receive the bytecode and instantiate it into a live service. The instantiation process may involve providing the service component with its own thread of execution, where it will have its own program loop. At this stage, the installed thin agent can be queried via invoking calls through its interface from the thin agent manager or through message interceptions as is now described.

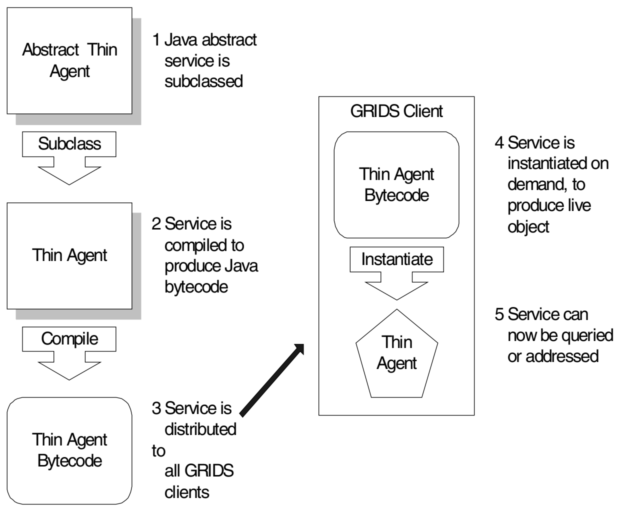

Abstract Thin Agent
1 Java abstract service is subclassed
Subclass
Thin Agent
2 Service is compiled to produce Java bytecode
Compile
Thin Agent Bytecode
3 Service is distributed to all GRIDS clients
GRIDS Client
Thin Agent Bytecode
Instantiate
Thin Agent
4 Service is instantiated on demand, to produce live object
5 Service can now be queried or addressed

Figure 3. Thin agent development lifecycle

Thin agents primarily perform their functions through the message interception approach. When a thin agent is installed into a GRIDS client, it first registers itself to receive particular message types. During execution the thin agent requires the GRIDS client to receive messages as illustrated in Figure 4. The TAM is used to route messages between the federate, the thin agents and the network. The TAM uses two queues to manage inbound and outbound messages. The TAM's thread of execution is used to check the inbound and outbound queues to direct messages through the GRIDS client. The TAM's circle in the figure represents the routing decision point. If the message type compares with a registered handling thin agent, the intercepted message is passed directly to the thin agent where it is processed. Otherwise the message continues either inbound or outbound according to its original path.

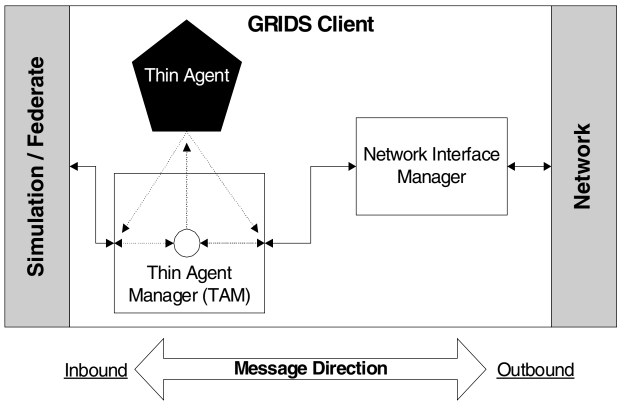


Figure 4. Thin agent message interception

4. Example Two: Legacy Support

The example describes an approach to supporting a legacy style simulation (DIS) within a HLA federation. The example attempts to interoperate a HLA federate (from example one) with a DIS federate as presented in a previous paper [6].

5.1 Introducing the DIS federate

The federate originally known as "tanksim" is based around a series of tanks autonomously moving within a two-dimensional environment. The environment is a fixed size with a square boundary. The tanks (each within its own federate) move within this environment via simple rules. A tank's movement pattern is broken into phases. A movement phase begins with the tank selecting a target waypoint. The tank moves towards the waypoint at a specified speed until the waypoint is achieved. Achievement of a waypoint is denoted by moving within a threshold distance from the waypoint (e.g. within 25 meters). The tank then selects a new random waypoint and begins another iteration of the movement phase.

In a DIS style simulation, each tanksim node is required to broadcast continuous updates to the rest of the distributed simulation. The scalability of DIS applications has always been a limiting factor due to bandwidth demands. Our previous work uses GRIDS to connect the tanksim federates together while providing relevance filtering through a dead reckoning thin agent (DR-TA) [6].

5.2 The bridging thin agent

This example discusses the key challenges faced in facilitating the interoperability of a DIS federate as described and an HLA federate from the first example. The approach to achieving the successful execution of the mixed federation using GRIDS is based on reusing as much of the federates and thin agents as possible. The use of a bridging thin agent to link the services provided by the existing DR-TA and the HLA-TA is used to achieve this. For clarity, the DIS federate is referred to as DISFed while the federate from example one is referred to as HLAFed.

The approach to interoperating the mixed mode federation is to bridge the communication between the HLA federate (HLAFed) and the DIS federate (DISFed) as in figure 5. It illustrates the position of the DIS-HLA Bridge Thin Agent (DHB-TA) when used within the GRIDS middleware. The new thin agent only resides within the node containing the DISFed and the DR-TA.

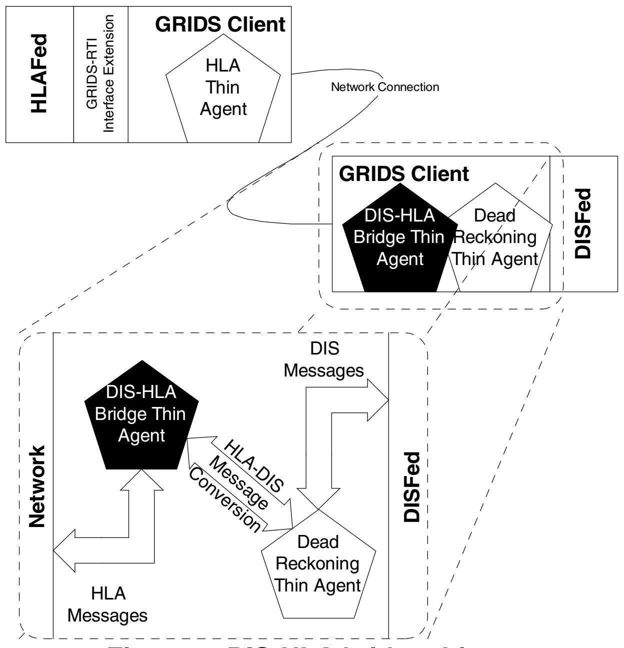

HLAFed
GRIDS-RTI Interface Extension
GRIDS Client
HLA Thin Agent
Network Connection
GRIDS Client
DIS-HLA Bridge Thin Agent
Dead Reckoning Thin Agent
DISFed
DIS Messages
DIS-HLA Bridge Thin Agent
HLA-DIS Message Conversion
Network
DISFed
Dead Reckoning Thin Agent
HLA Messages

Figure 5. DIS-HLA bridge thin agent

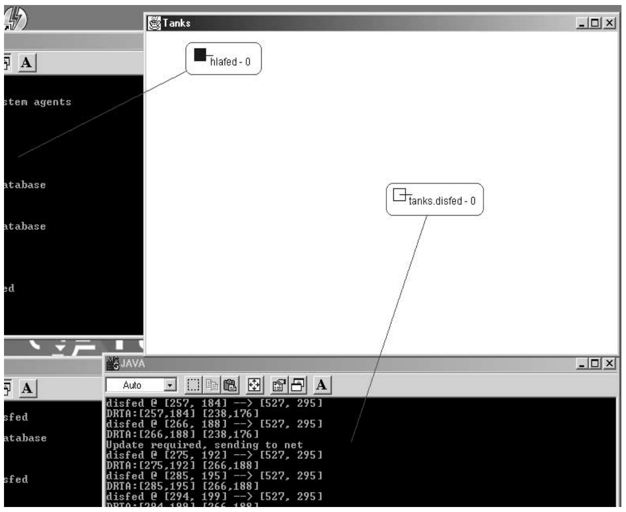
The function of the DHB-TA is further illustrated by the expanded view of the DISFed's GRIDS client in figure 5. It communicates HLA messages as specified in Table 1 with the rest of the federation. The DHB-TA translates any calls required to be sent to the federate into messages that the DR-TA can understand (entity state PDU's) and then passes them on. The DR-TA handles these messages as if they had been received from the network and operates accordingly. As the DISFed generates entity state updates destined for the federation, the DR-TA intercepts these according to normal execution. After processing, if the decision by the DR-TA is such that an update message is required to correct the low fidelity models of all the federation for the entity modeled locally, the update message is passed to the DHB-TA rather than the network. The DHB-TA translates this into the appropriate RTI call (such as an update attribute value call) for the rest of the federation (in this case, the HLAFed).

Table 1. RTI calls used by the federation

Service Group	RTI Ambassador	Federate Ambassador
Federation Management	Create Federation Execution(), Join Federation Execution()	
Declaration Management	Publish Object Class(), Subscribe Object Class Attributes()	
Object Management	Register Object Instance(), Update Attribute Values(), Request Object Attribute Value Update()	Discover Object Instance(), Reflect Attribute Values(), Provide Attribute Value Update()
RTI Support Services	Get Object Class Handle(), Get Attribute Handle()	

The DHB-TA does not need any associations to new message types. It is registered to intercept the existing DIS and HLA messages and implements the functionality to translate between the two federate types. Algorithmically, the DHB-TA is very similar to the HLA-TA. It must manage the handles for structures such as the class types of objects, the objects and the attributes. These data types are foreign to a DIS application. The following list details how the DHB-TA manages RTI calls that are not supported conceptually within a DIS application.

- **Object Creation** When the DHB-TA receives its first entity state update message from the DR-TA, it regards this as the object creation request. The object handle is generated and stored locally within the DHB-TA. The object discovery message is then propagated in the normal manner to the rest of the federation.
- **Object Discovery** If an object discovery message is received, the DHB-TA immediately requests an object attribute value update after storing the object handle.
- **Attribute Value Update** If an entity state update message is received by the DHB-TA from the DR-TA, the message is translated into an attribute value update and broadcast to the federation. A copy of the entities state is stored within the DHB-TA in case values are requests from other federates on demand.
- **Reflect Attribute Value** A received reflect attribute value message by the DHB-TA is translated into an entity state update message and passed to the DR-TA to deliver to the federate.
- **Request Object Attribute Value Update** If an object discovery message is received, a request object attribute value update message is sent to the owning federate of the object.
- **Provide Attribute Value Update** If the DHB-TA receives a provide attribute value update, the most current stored state values within the DHB-TA are used to generate an attribute value update.



```
disfed @ [257, 184] --> [527, 295]
DRTA:[257,184] [238,176]
disfed @ [266, 188] --> [527, 295]
DRTA:[266,188] [238,176]
Update required, sending to net
disfed @ [275, 192] --> [527, 295]
DRTA:[275,192] [266,188]
disfed @ [285, 195] --> [527, 295]
DRTA:[285,195] [266,188]
disfed @ [294, 199] --> [527, 295]
```

Figure 6. DIS and HLA mixed federation using GRIDS

Figure 6 is a screenshot of the mixed mode federation executing. To recap, the HLA federate from example one generates a user interface. This interface is a basic window which displays the position of the local tank as a solid square and all other tanks (updated from remote sources) as wire frame squares. The DIS federate uses a text-based interface directly to the command prompt. The solid square in the top left of the federate GUI is the HLAFed's tank. The wire frame square in the bottom right is the DISFed's tank.

6. Conclusions

This paper has demonstrated an approach to the implementation of an RTI based on extensibility. The paper has suggested that current implementations of the RTI do not provide support for the inclusion of additional functionality beyond that specified by the HLA standard. 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. The paper has presented one example to demonstrating our approach to extensibility. The example a thin agent service to provide support for a legacy DIS federate to interoperate with a HLA federate in a mixed federation.

The example demonstrates functionality which exists without the need for a modular RTI and can be delivered using traditional RTI. However, the examples demonstrate the flexibility provided by the extensibility mechanism and offer a vehicle to realize services not included by the HLA. Current work using the GRIDS RTI is continuing based on the following investigations:

- A comparison of implementations of a federation using an HLA compliant RTI and GRIDS. This work is in conjunction with Farshad Moradi from FOI (Swedish Defense Research Agency) and attempts to investigate the development time, best practice developing simulations using GRIDS and the performance of both RTIs.
- Time management to support distributed supply chain simulation. This is based on an increased interest in using distributed simulation technology for non-defense purposes. Work in this area has already attempted to use various middleware including the HLA [12], CORBA [13] and our approach using GRIDS and a thin agent implementing a conservative synchronization protocol (known as a CPADS-TA [8,9]).
- Commercial-of-the-shelf (COTS) simulation package integration to facilitate distributed simulation development using traditional simulation packages [12]. The challenge here is to use thin agents to gain access and control of the package and to provide mechanisms to relay event messages between packages.
- Full HLA RTI support through thin agents. A single component is used to encapsulate each service group providing the opportunity to upgrade incrementally and to select alternative algorithmic approaches to particular services.
- Thin agents as simulation components. This work is founded on the area of component-based simulation. The property of mobility of thin agents is seen as a method for load-balancing a distributed simulation through the transfer of entities between federates. Related to this, thin agents can also be used for object ownership transferal to other federates as with the HLA.

We hope to report on the progress of these research interests in future publications and welcome comments regarding these and alternative uses of components within RTI.

As a concluding remark, it is suggested that a modular RTI such as GRIDS may provide a vehicle for greater transfer of distributed simulation technology into non-defense areas. This is based on the view that GRIDS, for a given application is a skeleton infrastructure which can be fleshed with required. With the possibility to provide HLA support within the middleware, this type of functionality may generate interest and the flexibility required by a wider community while satisfying traditional users.

7. References

[1] IEEE Standard 1516-2000 IEEE standard for Modeling and Simulation (M&S) High Level Architecture (HLA) – Framework and Rules

[2] IEEE Standard 1516.1-2000 IEEE standard for Modeling and Simulation [M and S] High Level Architecture [HLA] – Federate Interface Specification

[3] IEEE Standard 1516.2-2000 IEEE standard for Modeling and Simulation (M&S) High Level Architecture (HLA) – Object Model Template

[4] M.Karlsson, and L. Olsson: "pRTI^{TM} 1516 – Rationale and Design". In Proceedings of the Fall 2001 Simulation Interoperability Workshop. Orlando, Florida. 01F-SIW-038. 2001.

[5] C. McLean, and F. Riddick: "The IMS Mission Architecture for Distributed Manufacturing Simulation." In Proceedings of

the 2000 Winter Simulation Conference. J. A. Joines, R. R. Barton, K. Kang, and P. A. Fishwick, eds, pp 1539-1548. Orlando, FL. 2000.

[6] S.J.E. Taylor, J. Saville., and R. Sudra: “Developing Interest Management Techniques in Distributed Interactive Simulation using JAVA”. In Proceedings of the 1999 Winter Simulation Conference. Phoenix, Arizona. 1999.

[7] R. Sudra, S.J.E. Taylor and T. Janahan: “Distributed Supply Chain Management in GRIDS”. In Proceedings of the 2000 Winter Simulation Conference. J. A. Joines, R. R. Barton, K. Kang, and P. A. Fishwick, eds, Orlando, Florida. 2000.

[8] R. Sudra., S.J.E. Taylor and T. Janahan: “GRIDS: A Novel Architecture for Distributed Supply Chain Management”. In Proceedings of the Fall 2000 Simulation Interoperability Workshop. Orlando, Florida. 00F-SIW-051. 2000.

[9] S.J.E. Taylor, R. Sudra., G. Tan, and J. Ladbrook: “Issues in Developing Distributed Supply Chain Simulation for the Automotive Industry” In Proceedings of the 2001 European Simulation Interoperability Workshop, pp 629-637 London, UK. 01E-SIW-097. 2001.

[10] R. Sudra and S.J.E. Taylor: “Extensibility: Modular HLA Services”. In Proceedings of the 2002 European Simulation Interoperability Workshop, *to appear*. London, UK.

[11] S.J.E. Taylor, R. Sudra., T. Janahan, G. Tan, and J. Ladbrook: “Towards COTS Distributed Simulation using GRIDS”. In Proceedings of the 2001 Winter Simulation Conference. B.A. Peters, J.S. Smith, D.J. Medeiros, and M.W. Rohrer, eds, pp 1372-1379. Arlington, Virginia. 2001.

[12] S.J. Turner, W. Cai and B.P. Gan: “Adapting a Supply Chain Simulation for HLA.” In Proceedings of the 4th International Workshop on Distributed Simulation and Real Time Applications, pp 71-78. IEEE Computer Society Press. San Francisco, California, USA. 2000

[13] B.P. Zeilger, D. Kim, and S.J. Buckley: “Distributed Supply Chain Simulation in a DEVS/CORBA Execution Environment.” In Proceedings of the 1999 Winter Simulation Conference, P.A.. Farrington, H.B. Nembhard, D.T. Sturrock, and G.W. Evans, eds, pp 1333-1340. Phoenix, AZ. 1999.