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

THE IMS MISSION ARCHITECTURE FOR DISTRIBUTED MANUFACTURING SIMULATION

Charles McLean
Frank Riddick

National Institute of Standards and Technology (NIST)
Gaithersburg, MD 20899, U.S.A.

ABSTRACT

This paper presents an overview of a neutral reference architecture for integrating distributed manufacturing simulation systems with each other, with other manufacturing software applications, and with manufacturing data repositories. Other manufacturing software applications include, but are not limited to systems used to: 1) design products, 2) specify processes, 3) engineer manufacturing systems, and 4) manage production. The architecture identifies the software building blocks and interfaces that will facilitate the integration of distributed simulation systems and enable the integration of those systems with other manufacturing software applications. The architecture is being developed as part of the international Intelligent Manufacturing Systems (IMS) MISSION project.

1 INTRODUCTION

Scientists and engineers within the NIST Manufacturing Systems Integration Division of the Manufacturing Engineering Laboratory are developing an architecture for distributed manufacturing simulation in collaboration with representatives from a number of outside organizations. The organizations are principally participants in the IMS MISSION Project (MISSION Consortium 1998). MISSION is just one of many international, collaborative projects that are currently underway as part of the IMS Program.

"The goal of MISSION is to integrate and utilize new, knowledge-aware technologies of distributed persistent data management, as well as conventional methods and tools, in various enterprise domains, to meet the needs of globally distributed enterprise modelling and simulation. This will make available methodologies and tools to support the definition of appropriate manufacturing strategies and the design of appropriate organizations and business processes. This goal will be achieved by establishing a modelling platform incorporating engineering knowledge and project information that supports space-wise and control-wise design, evaluation and implementation over the complete enterprise life cycle. This will be the foundation stone for an architecture to support engineering co-operation across the value chain of the entire extended enterprise." (MISSION Consortium 1998)

NIST is currently serving as the U.S. Regional Coordinator for the IMS MISSION project. For further information on the overall IMS Program, see the IMS Web page at <www.ims.org>.

2 DISTRIBUTED MANUFACTURING SIMULATION

This document takes a broad view of distributed manufacturing simulation (DMS). Normally a DMS may be thought of as a manufacturing simulation that is comprised of multiple software processes that are independently executing and interacting with each other. Together, these simulation software processes may model something as large as a manufacturing supply chain down to something as small as an individual piece of industrial machinery. Different software vendors may have developed the basic underlying simulation software. The modules may run on different computer systems in geographically dispersed locations. The simulation may be distributed to take advantage of the functionality of specific vendor's products, protect proprietary information associated with individual system models, and/or improve the overall execution speed of the simulation through the use of parallel computer processors.

DMS may also refer to a distributed computing environment where non-simulation manufacturing software applications are running and interacting with one or more simulation systems. Engineering systems may interact with simulation systems through service requests. That is, they submit data to a simulator for evaluation. For example, a computer-aided manufacturing application that has generated a control program for a machine tool may

submit that program to a simulator to verify that it is correct.

Another view of DMS is a computer environment comprised of multiple, functional modules that together form what today is commonly a single simulation system. Such an environment may include model building tools, simulation engines, display systems, and output analysis software.

2.1 Why Build Distributed Manufacturing Simulation Systems?

A distributed approach increases the functionality of simulation. For example, it could be used to

- model supply chains across multiple businesses where some of the information about the inner workings of each organization may be hidden from other supply chain members
- simulate multiple levels of manufacturing systems at different degrees of resolution such that lower level simulations generate information that feeds into higher levels
- model multiple systems in a single factory with different simulation requirements such that an individual simulation-vendor's product does not provide the capabilities to model all areas of interest
- allow a vendor to hide the internal workings of a simulation system through the creation of run-time simulators with limited functionality
- create an array of low-cost, run-time, simulation models that can be integrated into larger models
- take advantage of additional computing power, specific operating systems, or peripheral devices (e.g., virtual reality interfaces) afforded by distributing across multiple computer processors
- provide simultaneous access to executing simulation models for users in different locations (collaborative work environments)
- offer different types and numbers of software licenses for different functions supporting simulation activities (model building, visualization, execution, analysis).

The next section outlines the role that software architectures will play in enabling the development of distributed manufacturing simulations.

3 SOFTWARE ARCHITECTURE

In their book, Software Architecture: Perspectives on an Emerging Discipline, Mary Shaw and David Garlan, explain the significance of software architectures:

"As the size and complexity of software systems increase, the design and specification of overall system structure become more significant issues than the choice of algorithms and data structures of computation. Structural issues include the organization of a system as a composition of components; global control structures; the protocols for communication, synchronization, and data access; the assignment of functionality to design elements; the composition of design elements; physical distribution; scaling and performance; dimensions of evolution; and selection among design alternatives. This is the software architecture level of design."(Shaw and Garlan 1996)

A distributed manufacturing simulation architecture is needed to address the integration problems that are currently faced by software vendors and industrial users of simulation technology. Neutral simulation interfaces would help reduce the cost of data importation and model sharing, and thus would make simulation technology more affordable to users. The definition of a neutral architecture for distributed manufacturing simulation is the first step towards identifying the information models, interfaces, and protocols for integrating these systems.

This step can be achieved by decomposing the distributed manufacturing simulation architecture into three major functional views: *Distributed Computing Systems, Simulation Systems, and Manufacturing Systems.* Each architectural view defines a set of system elements, data models, and interface specifications for integrating distributed manufacturing simulations. Aspects of each view are interrelated to and interconnected with aspects of the other views. The views can be thought of as three sides of a cube.

3.1 Distributed Computing Systems View

This architectural view is concerned primarily with simulation as a set of computers and software processes that are simultaneous executing and communicating with each other across a computer network. This view also addresses issues pertaining to the general management and integration of the software applications that are used to generate models and data for the simulations. The fact that the software processes are simulations or simulation-related is not particularly critical in this view. This view is not concerned with simulation or manufacturing data content.

This view provides the infrastructure that allows us to implement simulation development and execution environments as distributed systems. Elements of this view include: hardware computing platforms; operating systems, distributed computer processes, integration infrastructures, process initialization and synchronization, software development environments (including but not limited to editors, compilers, system build utilities, debuggers, source code, general subroutine and header libraries, run-time modules, and system test data), communications systems, information models and data

dictionaries, work flow management systems, database management systems and databases, product data management systems, version control and configuration management, computer file systems and files, system installation and maintenance utilities, computer security and data protection services, license verification systems, and World Wide Web access mechanisms. It also includes various input and output peripheral devices such as digital cameras, scanners, monitors, projection displays, printers, and virtual reality interfaces.

There are five major clusters of information systems that are relevant to the distributed manufacturing simulation problem: 1) software development systems; 2) design, engineering, production planning, and simulation model development systems; 3) distributed manufacturing simulation execution systems; 4) manufacturing management, control, production, support systems, and 5) distributed manufacturing data repository systems.

Figure 1 groups these systems into four computing environments and a shared, common data repository. The figure presents a logical grouping of system elements. Undoubtedly each implementation of this architecture will be based on different information systems and physical configurations. The major elements of the figure are described briefly below.

The Software Development Environment is used to develop simulation engines, visualization systems, integrating infrastructures, and other software applications. It is not the central focus of the architecture and will not be addressed in this paper. The Design, Engineering, Production Planning, and Simulation Model Development Environment contains the systems that generate models and data used by simulation and manufacturing itself. It is described in further detail below. The Distributed Manufacturing Simulation Execution Environment contains simulation engines executing models, visualization systems, and infrastructure systems to manage and integrate those simulations. The Manufacturing Management, Control, Production and Support Systems Environment is made up of the "real" systems that are used to run and perform the manufacturing operations.

There are five component elements of the Design, Engineering, Production Planning, and Simulation Model Development Environment: 1) product design applications and tool kits; 2) manufacturing engineering applications and tool kits; 3) production management applications and tool kits; 4) simulation model development applications and tool kits, and 5) work flow management systems. In this environment, the work flow management system provides the integrating infrastructure. It manages and sequences activities within the applications and tool kits that generate manufacturing models and data. Tool kits are tightly coupled suites of applications that work together to perform a related set of functions. Tool kits may be manually driven or more automated expert systems.

Product design applications may include conceptual and detailed design, solid modeling, bill of materials generation, design handbooks, parts catalogs, and various analysis tools. Some manufacturing engineering applications may include process planning and process specification, plant layout, machine tool programming, time standards development, line balancing, and tool and fixture design. Production management applications may include manufacturing resource planning, batch and lot sizing, and scheduling applications. Simulation model development tools include functions such as flowcharting, diagramming, model definition, and user level programming.

A communications network connects environments with each other and the Manufacturing Data Repository. The Repository is a consolidation of the various data stores and management systems that are used by the various information systems environments. It logically integrates the file systems,

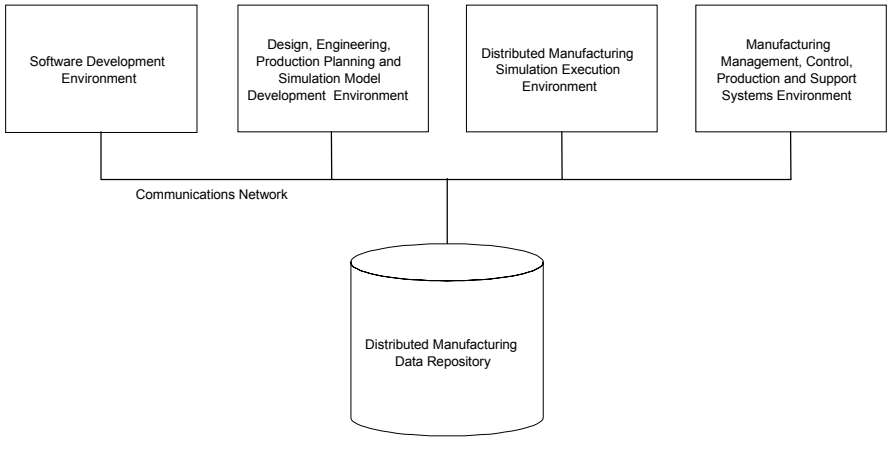

Figure 1: Relationships Between the Major Elements of the DMS Architecture

Web pages, data bases, and libraries used for the storage of data by design, engineering, production planning, real manufacturing systems, simulation model development, and executing distributed manufacturing simulations. In different implementations of the architecture, the repository may reside on a single computer system, a file server, or be geographically distributed across a network.

The Distributed Manufacturing Data Repository may include the following types of data stores and management systems: computer file systems, Web pages and files, object-oriented database management systems, relational database management systems, special-purpose library management systems, and source-code control systems for software. A common data access interface mechanism will be used to simplify access to the data repository by all software environments and applications within those environments. References to documents in the data repository may be specified as Uniform Resource Locators (URLs) see (Berners-Lee et al. 1998). This will allow the identification of documents, both remotely and locally stored using the well-established, standard, World Wide Web access mechanism.

Figure 2 shows a decomposition of the Distributed Manufacturing Data Repository into its component elements. All of the types of data stores indicated in the figure do not necessarily have to be included in an implementation of the architecture. In the future, additional data management schemes and data stores may be added to the repository structure. From this point forward in this document, the Distributed Manufacturing Data Repository and Common Data Access Mechanism will be treated and represented as a single module.

3.2 Simulation Systems View

This architectural view is concerned with the specifics of building, initializing, running, observing, interacting with, and analyzing simulations. In this view, simulation systems, tools, and supporting applications should be viewed generically; i.e., independent of the manufacturing domain. The same system elements could be used for simulating other problem domains. Major elements of this view include: simulation coordination and management, visualization systems, manufacturing data preparation and model development tools, simulation models, discrete event and process simulation engines, component module and template libraries, mathematical and analytical models, input distributions, timing and event calendars, and output analysis tools.

Figure 3 illustrates the relationship between the various elements of the distributed manufacturing simulation execution environment. The integration infrastructure for this environment, the Run Time Infrastructure (RTI), is based on the U.S. Department of Defense High Level Architecture (HLA) developed by the Defense Modeling and Simulation Office (DMSO) (Kuhl et al. 1999). The HLA was developed by DMSO to provide a consistent approach for integrating distributed, defense simulations. Several implementations of the HLA RTI software are currently available from different sources. There is, however, no interoperability across RTI implementations. A distributed simulation running on different computer systems across a network must use the same RTI software as an integration infrastructure.

An HLA-based distributed simulation is called a federation. Each simulator, visualization system, real production system, or output analysis system that is integrated by the HLA RTI is called a federate. One common data definition is created for domain data that is shared across the entire federation. It is called the federation object model (FOM). Each federate has a simulation object model that defines the elements of the FOM that it implements.

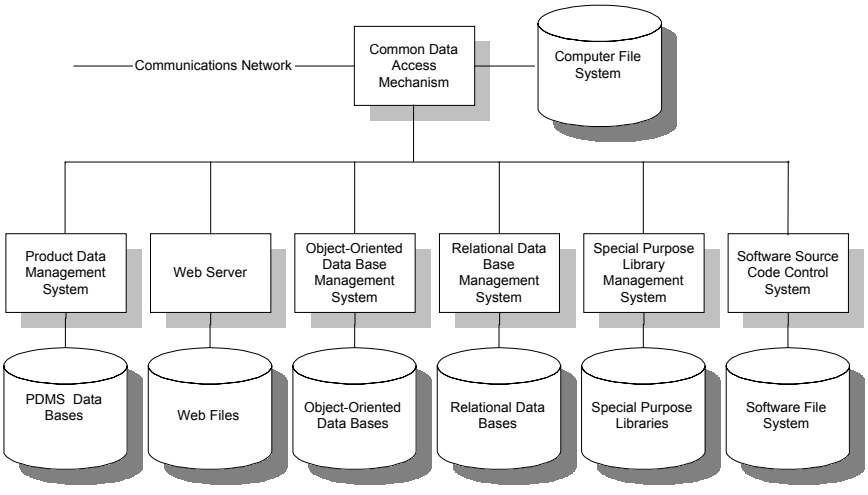

Figure 2: Decomposition of the Distributed Manufacturing Data Repository

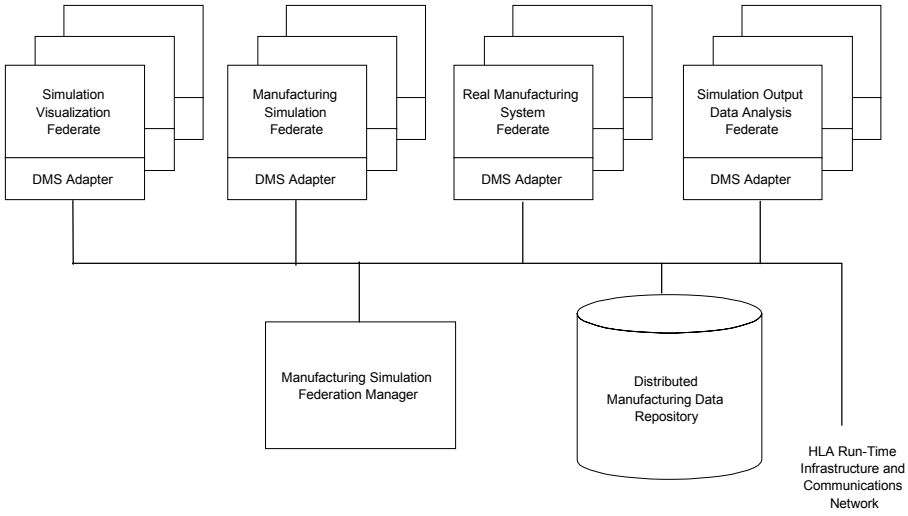

Figure 3: Distributed Manufacturing Simulation Environment Elements Integrated by the HLA Run Time Infrastructure

A DMS Adapter Module is incorporated into each DMS federate. The DMS Adapter will handle the transmission, receipt, and internal updates to all FOM objects used by a federate. The DMS Adapter Module will contain a subroutine interface and data definition file that will facilitate its use as an integration mechanism by software developers. The goal of the DMS adapter is to ease the development of distributed manufacturing simulations by reusing implementations for some of the necessary housekeeping and administrative work. The DMS adapter provides a simplified time management interface, automatic storage for local object instances, management of lists of remote object instances of interest, management and logging for interactions of interest, and simplified object and interaction filtering.

Several functions may be needed for the proper operation of a distributed simulation that are logically outside of any one simulation federate. In the distributed manufacturing simulation environment, the Manufacturing Simulation Federation Manager is the architectural element that provides these functions. Its may implement functionality to execute initialization scripts that launch federates, to provide initialization data to federates, to assist in federation time management, and to provide a user interface so that users can monitor and manipulate the federation and invoke federation services.

3.3 Manufacturing Systems View

This architectural view is concerned with modeling the behavior and data of specific manufacturing organizations and systems, from the supply chain down to individual machines on the factory floor. Major elements of this view include, but are not limited to

- manufacturing organizational templates and structures, business process and organizational models
- supply chain systems - refineries, mills, factories, warehouses, distributors, transportation systems, wholesalers, retailers, customers, and so on
- manufacturing facility departments, areas, and subsystems - design, engineering, procurement, finance, production shops, work cells, production lines, workstations, inventory storage areas, shipping and receiving, and so on
- production resources and support equipment - machine tools, inspection equipment, material handling systems, storage buffers, robots, workers,
- tools and materials - cutting tools, hand tools, jigs and fixtures, consumables, components, part blanks, sheet and bar stock, work-in-process inventory, and so on
- manufacturing information systems - design, engineering, production planning and scheduling, tool management, shop floor data collection systems, and
- manufacturing documents and data - work flow patterns, orders, jobs, product data, part designs, process plans, production calendars, schedules, layouts, and other reference data (machinability data, statistical distributions).

Different manufacturers will create different supply chain organizations and arrangements of systems within each organization. The DMS architecture must be flexible enough to allow these different system configurations, but still enable increased integration. As such, the architecture does not mandate a particular manufacturing organization. It does require the development and specification of one DMS FOM.

Many objects in the FOM may reference documents containing more detailed information that are stored in a file system, PDM system, or database. An example of such a document might be a part design file or a process plan. The Extensible Markup Language (XML) can be used to define new document types (Goldfarb and Prescod 2000). XML allows for the definition data that has semantic information in addition to the data values. XML data-type-definitions (DTD) may be used to define new document formats. Advantages of this approach include:

- the set of supported document types can be easily extended
- each individual document format can be easily modified
- COTS tools are available to implement creation, parsing, interpreting, and displaying the documents
- XML documents from other sources can easily be supported
- different instances of file structures may be created to convey the same semantic information
- XML-enabled browsers can intelligently display the data
- semantic validation of the files is possible.

Even without the DTD, XML files are often both human and machine readable because of the semantic information that is included.

There are potentially many document types that will be stored as distributed manufacturing simulation data. Some of these document types have widely-accepted or standardized formats. Examples of these include the many kinds of CAD files (DXF, IGES, etc.), image files (GIF, TIFF, BMP, etc), and executables (EXE, com, bat, dll, etc.). However, many manufacturing documents do not have standardized format. Schedules, BOMs, and process plans are examples of such documents. While it is easy to come up with acceptable representations for such data that are appropriate for short-term use, it is highly likely that these representations will need modifications, possibly major modifications, over time. A mechanism is needed to allow the definition of extensible formats for new document types without adversely affecting the rest of the DMS architecture or interfaces. XML can be that mechanism. XML DTD's must be stored in and uniquely accessible from the DMS data repository. An initial set of document formats should be developed and allowed to expand over time as the need arises.

4 INTEGRATION VIA DMS ADAPTER AND THE HLA/RTI

In the discussion and diagrams below, the changes necessary for integrating a legacy simulation into a distributed simulation using the HLA and the DMS Adapter will be discussed. The term legacy simulation is used to indicate a manufacturing-oriented or general-purpose discrete-event simulation tool that does not have native support for the HLA or DMS Adapter technologies.

4.1 Simplified Simulation Execution Architecture

In Figure 4, a simplified view of a non-distributed legacy simulation application is shown. It consists of a simulation execution system executing a simulation model. The simulation model is a behavior-oriented description of the logical system that is to be simulated. Simulation execution systems often support the visualization of the executing model and statistical reporting of the simulated events that are generated during execution. Data that are needed as input to or that are generated by the executing simulation are maintained in the persistent data store.

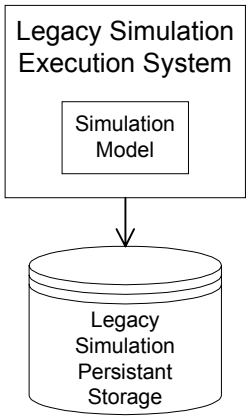

Figure 4: Simplified View of a Typical Legacy Simulation System

4.2 Integration using the HLA/RTI

Figure 5 shows the architecture of a legacy simulation that has been integrated into a distributed simulation using the HLA. On the right side of the diagram, a simplified view of the HLA architecture is presented (constructs or concepts that are beyond the scope of this presentation have been left out for brevity). The Federate Object Model (FOM) is a description of the data that can be exchanged between federates. The FOM is usually different for each distributed simulation that is developed. The RTI Ambassador implements the interface through which federates send information to the RTI. This interface contains over 120 methods that provide the capability to manage federation creation, manage object class definitions, manage information exchange using objects and interactions, and manage the advancement of time for the federation.

While the RTI Ambassador provides the mechanism for sending information to the RTI, an implementation of

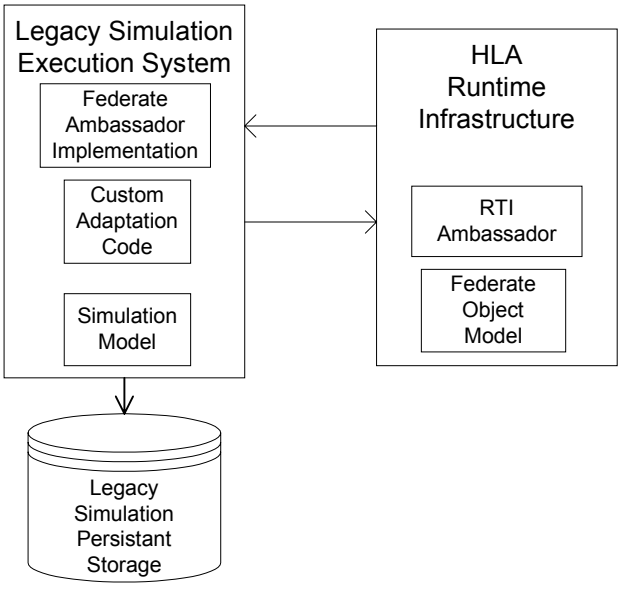

Figure 5: Legacy Simulation Integration Using the HLA/ RTI

the Federate Ambassador interface is necessary to be able to receive information from the RTI. The Federate Ambassador is an interface that contains around 40 methods that define how the RTI sends information to a federate asynchronously in response changes in the state of the federation. These state changes may be in response to calls to the methods on the RTI Ambassador interface made by any federate in the federation. An implementation of the Federate Ambassador interface is not provided with the RTI software. The rules of the HLA require that an implementation of the Federate Ambassador be provided by the legacy simulation. Furthermore, this implementation must be consistent with the information defined in the FOM that is being used in this federation.

4.2.1 HLA/RTI Integration Issues

Since legacy simulation systems are not designed to be used with the HLA/RTI, code must be developed to adapt the legacy simulation system for such purposes. Normally this code is complex. In addition, although some of the code can be reused, a significant amount of code will need to be added or modified for each distributed simulation that is developed. In the following sections, some of the important issues related to the complexity and reusability of the adaptation code are discussed.

4.2.1.1 RTI Interface Complexity

There are roughly 120 methods in the RTI Ambassador interface and 40 methods in the Federate Ambassador interface. Depending on the current state of the RTI, the federation, and the data that is defined in the FOM, invoking a method can cause vastly different outcomes to occur. While the richness of the RTI's interfaces provide for an extremely flexible simulation integration approach, a side effect is that the learning curve for understanding these interfaces is quite high.

4.2.1.2 The RTI's Implicit Invocation Architecture

The architecture of the RTI is based on what is called an "implicit invocation architecture." In this approach, a federate can modify the state of the federation by invoking methods of the RTI Ambassador interface. Information relating to changes in the state of the federation is passed back as asynchronous callbacks to methods in the Federate Ambassador that was implemented by the federate. While this is an efficient and flexible approach, it makes adapting legacy simulation difficult because legacy simulations usually provide only procedurally oriented mechanisms for integration.

4.2.1.3 Inadequate Integration Mechanisms are Provided by the Legacy Simulations

To use the interfaces of the RTI, some adaptation code must be written using a language supported by one of the RTI language mappings. Mappings currently exist for languages such as C, C++, Java, and CORBA IDL (Ben-Natan 1995). While some simulation systems provide mechanisms to call functions written in such languages natively, many do not. Integrating those legacy simulations usually requires a combination of proprietary-language code, file input/output, and socket programming, depending on which mechanisms are provided. This situation increases the complexity of developing and maintaining the adaptation code.

4.2.1.4 Cooperative Time Management

In distributed simulations in which federates must cooperatively manage the advancement of time, the legacy simulation must be modified to cede some of the control over the advancement of time where previously it had complete control. Because the RTI provides multiple mechanisms for coordinating time advancement, choosing the appropriate mechanism and properly implementing the adaptation code to support it can require significant forethought and development.

4.2.1.5 Storage and Maintenance for Instances of FOM Objects

Many legacy simulations have internal representations for entities such as parts or machines, and these simulations can maintain the information about such entities as they are

created during a simulation execution. The definitions of these entities will differ between different legacy simulations. To enable the exchange of data relating to these entities, neutral representations of these entities are usually defined in the FOM as object classes and associated attributes. However, the HLA/RTI provides no mechanism for storing object class instances. It only provides for storage of information related to the owner of a particular object instance, the class of the instance, and the attributes that are associated with an instance. Therefore, storage for instances of FOM objects must be provided by the legacy simulation. This is in addition to whatever storage has been set aside to maintain the legacy simulation's internal representation of an object. Adaptation code to maintain FOM object storage and to coordinate state changes between the internal representations and the FOM representations of objects must be developed.

4.3 Integration using the DMS Adapter

In the previous section, some of the issues that are related to integrating legacy simulations using just the facilities of the HLA were discussed. It shows that developing the Federate Ambassador and adaptation code can be a significant undertaking when developing a distributed simulation, and that this effort must be repeated for each legacy simulation that is to be integrated.

Figure 6 shows the architecture of a legacy simulation that has been integrated into a distributed simulation using the DMS Adapter. Instead of having legacy simulations integrated directly with the HLA/RTI, those simulations will interact with the interface of the adapter. The goal of the adapter is to provide a simplified method for integrating legacy simulations into distributed simulations while also providing as much of the capabilities of the HLA/RTI as possible. The reader should note that simplified does not imply simple. Adaptation code must still be developed to integrate a legacy simulation system with the DMS Adapter. However, by reducing the complexity of the interface to which the legacy simulation is being integrated, the level of effort for performing the integration should be greatly reduced.

4.3.1 Architectural Goals for the DMS Adapter

What follows is a list of design goals for the architecture of the DMS Adapter. If met, implementing distributed simulations using the DMS Adapter should be simpler than when using the approach that was depicted in Figure 5.

4.3.1.1 Reduce Interface Complexity

The interface of the adapter will have approximately 35 methods instead of the 120 methods with 40 callbacks defined by the RTI and Federate Ambassadors.

4.3.1.2 Remove Federate Ambassador Implementation Issues from the Legacy Simulation

Legacy simulations will not have to develop Federate Ambassador implementations. The adapter will implement a federate ambassador and use it to receive information from the RTI.

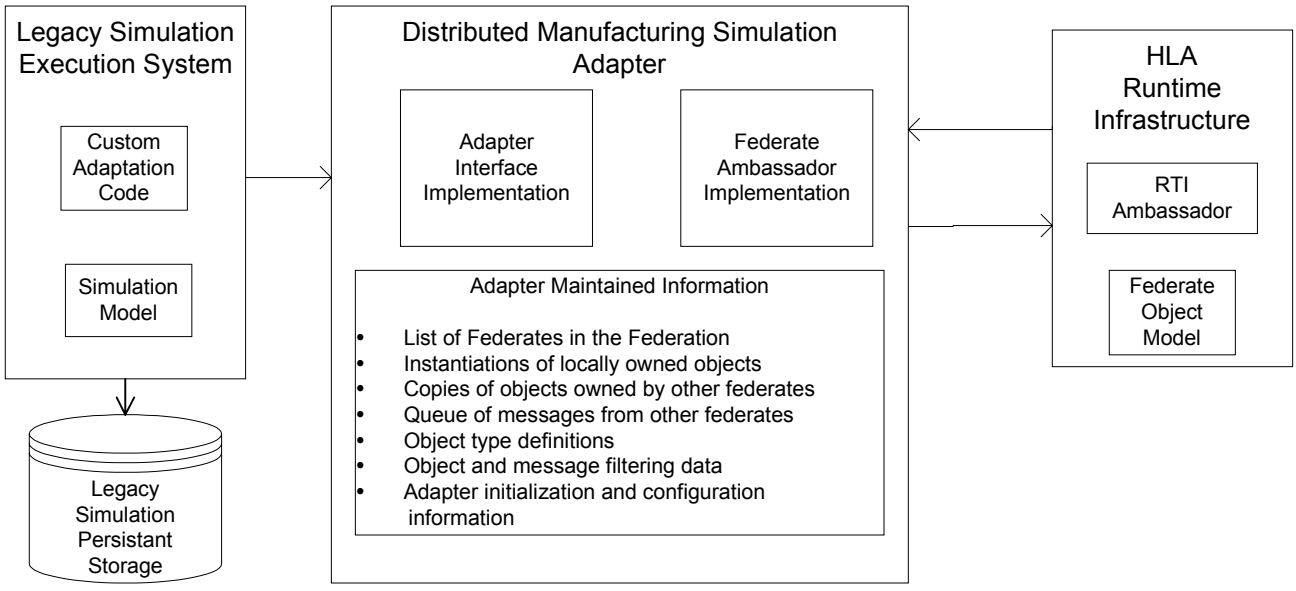

Figure 6: Legacy Simulation Integration Using the DMS Adapter

4.3.1.3 Define an Interface that Facilitates Integration with Procedurally Oriented Legacy Simulations

The results of invoking most of the methods in the adapter's interface will be returned immediately to the legacy simulation. Information that must be passed back asynchronously to a federate will be stored in a message queue in the adapter associated with that federate. This includes information that is generated by the activities of other federates in the federation. The adapter will provide the storage for this information and provide methods to access this information upon request from the legacy simulation.

4.3.1.4 Minimize the Impact of Changes to the Information Model through the Development of Generic FOM Objects that Contain XML Stings

To overcome the problem of having to develop different FOM's for each distributed simulation configuration, the information about the classes and attributes for the objects that are to be exchanged will not be defined in the FOM. A generic object will be defined in the FOM and this object will be exchanged between federates. This generic FOM object will contain an XML string that contains the semantic content for the object. The XML string is the information that will be passed to the legacy simulation. The generic FOM object will also contain information about the type of data contained in the XML string. This will facilitate filtering and routing of object updates by the RTI. There are five major benefits to this approach:

1. Only one FOM needs to be developed for use with the DMS Adapter.
2. Only one implementation of the Federate Ambassador needs to be developed for use with the DMS Adapter.
3. The DMS Adapter does not have to be modified and recompiled for each distributed simulation configuration.
4. The information model (the definition of the entities, attributes and messages that will be exchanged between simulations) can be changed without changing the FOM, Federate Ambassador implementation, or the DMS Adapter Implementation.
5. Implementations of mechanisms for manipulating XML data are widely available and can be used in the development of both the DMS Adapter and the adaptation code for legacy simulations.

4.3.1.5 Maintain Storage for the Objects that are to be Exchanged between Simulations

As discussed in a previous section, the definition of an object (class and associated attributes) that is to be exchanged between simulations will differ from the internal definition that each simulation supports for that object. Since each legacy simulation only provides storage for its internal objects and the RTI provides no mechanism for the storage of objects, storage and maintenance for the objects that are to be exchanged must be provided. The DMS adapter will provide this capability.

Each adapter will provide methods that allow a legacy simulation to create, modify, and delete objects that can be shared with other federates in the federation. Objects will have "owners", and ownership will be granted initially to the adapter (and associated legacy simulation) that created it. Ownership is required for modification or deletion operations on an object to succeed. Storage for "owned" objects will be provided by the DMS Adapter that owns the object. In addition, storage for copies of objects owned by other DMS Adapters will be provided. Each DMS Adapter will use the services of the RTI to distribute object update information for the objects it owns, and will incorporate object update information it receives about objects owned by other DMS Adapters. In this way, the DMS Adapters in the federation can work cooperatively to maintain updated information about all the objects in the federation, without the direct intervention of their associated legacy simulations.

4.3.1.6 Simplify Time Coordination

The RTI provides a multitude of time synchronization methods that are extremely flexible and powerful but are also quite complicated. The adapter implements a "time-stepped" synchronization approach. DMS Adapter methods are provided to declare that the associated legacy simulation wishes to advance to a certain simulation time, and to check if it is ok to advance to this time. When the DMS Adapter indicates to the legacy simulation that it is ok to advance, the legacy simulation can then "simulate" from its current simulation time to the new simulation time that it requested. It can then use the other methods in the DMS Adapter interface to get information about what was going in the rest of the federation while it was executing its "simulation step." When all of the simulations use this method, the functionality of the RTI's time management services ensures that the collective advancement of all of the simulations proceeds properly.

5 CONCLUSIONS

This document has provided a brief overview of the distributed manufacturing simulation architecture that is being developed as a part of the IMS MISSION Project.

The approach taken in the architecture is to facilitate integration of existing commercial systems with minimal new development work. The architecture also should enable experimentation with research systems that are based on evolving technology. The architecture describes the major system modules, data elements or objects, and interfaces between those modules. It uses the DOD High Level Architecture and Run Time Infrastructure as an integrating infrastructure. Detailed specifications will have to be prepared for the key interfaces identified in this document. Prototypes of each of these systems are being developed, tested, and integrated with commercial simulation systems, modeling tools, and other related manufacturing software applications as part of the IMS MISSION Project.

ACKNOWLEDGMENTS

Work described in this paper was sponsored by the NIST Systems Integration for Manufacturing Applications (SIMA) Program. No approval or endorsement of any commercial product by the National Institute of Standards and Technology is intended or implied. The work described was funded by the United States Government and is not subject to copyright.

REFERENCES

MISSION Consortium. 1998. Intelligent Manufacturing System (IMS) Project Proposal: Modelling and Simulation Environments for Design, Planning and Operation of Globally Distributed Enterprises (MISSION), Version 3.3. Shimuzu Corporation, Tokyo, Japan

Shaw, M., and D. Garlan. 1996. Software Architecture: Perspectives on an Emerging Discipline. Prentice-Hall: Saddle River, NJ

Berners-Lee, T., R. Fielding, and L. Masinter. 1998. Uniform Resource Identifiers (URI): Generic Syntax (RFC 2396). Internet Engineering Task Force

Kuhl, F., R. Weatherly, and J. Dahmann. 1999. Creating Computer Simulations: An Introduction to the High Level Architecture. Prentice Hall: Upper Saddle River, NJ

Goldfarb, C., and P. Prescod. 2000. The XML Handbook, Prentice Hall: Upper Saddle River, NJ

Ben-Natan, R., 1995. CORBA: A Guide To The Common Object Request Broker Architecture. McGraw-Hill: New York, NY

AUTHOR BIOGRAPHIES

CHUCK MCLEAN is Leader of the Manufacturing Systems Engineering Group in the U.S. National Institute of Standards and Technology (NIST) Manufacturing Systems Integration Division. He has managed research programs in manufacturing simulation, engineering tool integration, product data standards, and manufacturing automation at NIST since 1982. He has authored more than 50 papers on topics in these areas. He holds a Master's Degree in Information Engineering from University of Illinois at Chicago and Bachelor's Degree from Cornell University.

FRANK RIDDICK is a staff member in the Manufacturing Systems Engineering Group in the U.S. National Institute of Standards and Technology (NIST) Manufacturing Systems Integration Division. He has participated in research and authored several papers relating to manufacturing simulation integration and product data modeling. He holds a Master's Degree in Mathematics from Purdue University.