A COMPOSITION PARADIGM FOR VIRTUAL OR CONSTRUCTIVE PLATFORM ENTITY DESIGN

Mark Biddle
SAIC
Orlando, Florida

Constance Perry
STRICOM
Orlando, Florida

Robert Franceschini
IST
Orlando, Florida

ABSTRACT

This paper describes an approach to implementing virtual or constructive platform entity design using a component based framework architecture. The approach is being implemented as part of a research effort to develop a framework that allows a user to compose and recompose applications for a distributed modeling and simulation (M&S) environment, and to further compose applications into a reusable packaged execution environment. The purpose of the project is not to develop virtual or constructive applications but to develop the composition framework. However, some virtual and constructive platform entities are being developed as test cases for the framework. This paper focuses on the use of this experimental framework to develop these virtual or constructive platform level entities for the synthetic battlespace. It examines alternative component design and composition methods, which can be used to support a wide variety of application domains. Examples of these application domains include distributed mission training, doctrinal policy analysis within a particular theater or tactical situation, or engineering design alternative analysis of a particular system or subsystem, to name but a few examples. The paper discusses issues related to component design, storage and retrieval of components from a repository, composition of components into a complete application, and interoperability issues related to executing a composed application in a distributed M&S environment. The purpose is to provide lessons learned and information exchange for the benefit of projects that are concerned with developing virtual and constructive entities.

BIOGRAPHICAL SKETCH

Mark Biddle is a senior systems engineer with over six years of experience in the distributed interactive simulation industry, and eight years of experience with in-service engineering support of mission critical systems. He has performed project management, requirements analysis, systems engineering, design, integration and testing in a program manager and project lead capacity for two years with SAIC, four years with the Naval Air Warfare Center Training Systems Division, and eight years with the Naval Undersea Warfare Center. Mr. Biddle received his Masters of Science degree in Engineering Management from Old Dominion University in Norfolk Virginia, and a Bachelor's degree in Electrical Engineering from the Pennsylvania State University.

Constance M. Perry is a Principal Investigator for the Engineering Technology Development Directorate at the U.S. Army Simulation, Training and Instrumentation Command (STRICOM). Ms. Perry has over 11 years experience in the simulation industry. She is the lead Principal Investigator on programs in the Advanced Distributed Simulation technology area. Ms. Perry received a Masters of Science degree in Industrial Engineering with a specialization in Modeling and Simulation Analysis from the University of Central Florida and a Bachelors of Science degree in Industrial Engineering from the University of South Florida.

Robert W. Franceschini is an Assistant Professor of Computer Engineering in the School of Electrical Engineering and Computer Science at the University of Central Florida. He earned B.S. and Ph.D. degrees in computer science from UCF. His research interests involve distributed modeling and simulation, including computer generated forces, interoperability, and multi-resolution simulation. He is the technical leader of a group of 23 UCF faculty and students currently involved in nine externally-funded research projects in these areas.

A COMPOSITION PARADIGM FOR VIRTUAL OR CONSTRUCTIVE PLATFORM ENTITY DESIGN

Mark Biddle
SAIC
Orlando, Florida

Constance Perry
STRICOM
Orlando, Florida

Robert Franceschini
IST
Orlando, Florida

INTRODUCTION

This paper outlines a new paradigm for platform level simulation application development and implementation. It seems appropriate, therefore, to begin with a quick definition of what is meant by "platform level" application. For the purpose of this discussion, let us turn to the Real Time Platform Reference Federation Object Model (RPR FOM). The RPR FOM is a widely used object model for implementation of training federations, and it defines the following subclasses for platform entities:

- Aircraft
- Amphibious Vehicle
- Ground Vehicle
- MultiDomain Platform
- Spacecraft
- Submersible Vehicle
- Surface Vessel

While the issues covered in this paper are not limited to this specific definition, it is a suitable one for the purpose of the discussion.

The paradigm to be discussed is based upon research that is being conducted by the U.S. Army Simulation, Training and Instrumentation Command (STRICOM), through the support of a collaborative development team, which includes SAIC, Oracle, and IST. The project is focused on building a framework to facilitate application composition and interoperability in a distributed M&S (Modeling and Simulation) environment. This research has provided insight that is applicable not only to platform application design, but to design of distributed M&S applications in general.

A recent survey paper characterizes simulation interoperability research as being either top-down or bottom-up [4]; however, no research has been able to connect the top-down and bottom-up approaches into a unified interoperability solution. We believe that the STRICOM composability framework has the potential to bridge between the top-down and bottom-up approaches and thus will become a long-term simulation interoperability solution.

The remainder of this paper discusses issues related to using this framework for synthetic battlespace platform level application development. This discussion is not meant to recommend specific solutions, but rather to highlight some issues that need to be considered by application developers and by the M&S community as a whole.

INTEROPERABILITY FRAMEWORK

Overview of ModISE

The framework mentioned above was developed by STRICOM to facilitate web-based composition and interoperability of distributed M&S applications (See figure 1). This framework is called the Modular Interoperable Synthetic Environment or ModISE. The discussion of platform level application design, as addressed by this paper, is centered around a prototype combat vehicle application that was built as a test application for the ModISE framework. It is important to note that the purpose of the framework is not to build virtual or constructive platform level simulations, but rather to provide a framework within which applications of any domain type may be composed. Even so, because the framework itself is an integral part of the design approach being explored, an overview of it is necessary to support the discussion of the specific issues of platform level application design.

The ModISE framework is designed to allow a user to graphically compose an application, via a web-based Graphical User Interface (GUI), from reusable components that are stored in a web-based repository. The composition is done in a pre-run time setting, and an XML (Extensible Markup Language) description of the composition is stored in the repository for retrieval at run time. This XML composition description is called an Execution Specification or ESpec.

At run time, a user executes an Interoperability Engine (IE) application, which is part of the ModISE framework. The IE puts the composed application together at run time and executes it. Via a pop-up window in the IE, the user selects an ESpec to execute. The IE downloads the ESpec from the

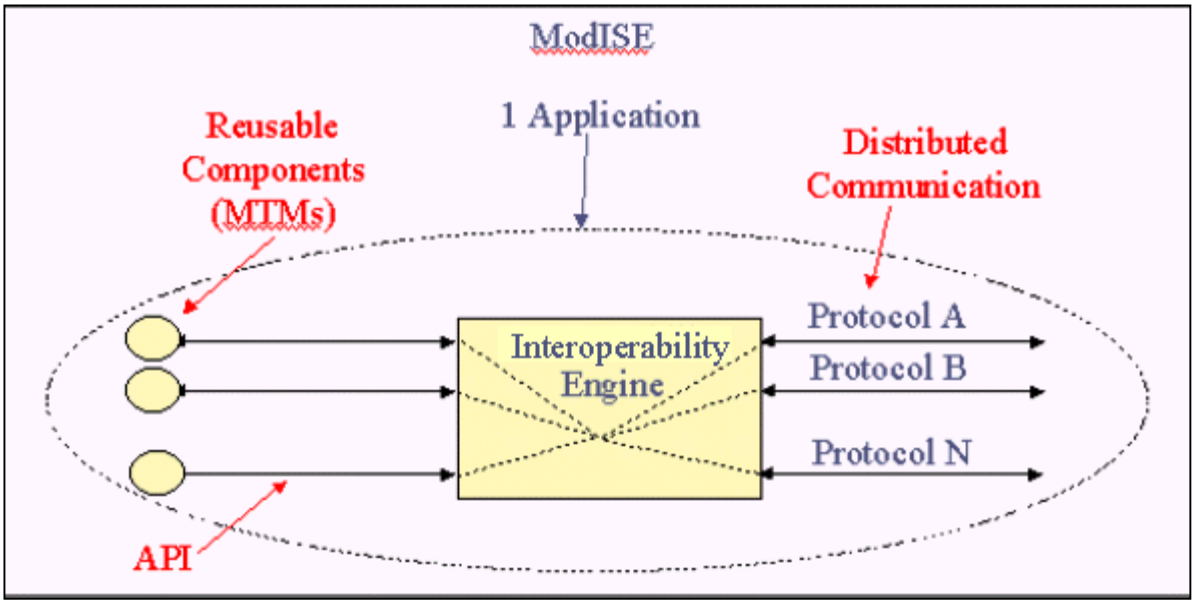

Figure 1: ModISE Concept

repository. Then instances of the components identified in the ESpec are acquired from the repository and connected into a single application. This application is then instantiated by the IE, and the run time execution begins.

Each instantiated application has its own dedicated IE, and components in the repository can be reused by any number of applications.

Reusable Components

The reusable components in the ModISE framework are called MTMs, which is an acronym for Mapper/Translator/Model. Each MTM could be something as simple as a Celsius to Fahrenheit temperature value converter, or as complex as an aircraft engine model. The MTMs are reusable and composable building blocks of functionality.

MTMs can provide functionality for applications that are composed and executed from within an instance of an IE. An example of this might be a radar model MTM that is integrated with other components into an aircraft entity application. MTMs can also provide supplemental or augmented capability for applications that are connected externally to an instance of an IE. An example of this might be an ordnance server MTM that is executed from within an instance of an IE to provide services to externally connected synthetic battlespace entity applications. Lastly, MTMs can provide functionality to facilitate interoperability between external applications connected to the IE, functioning as a third party translator in this case. An example of this might involve two MTMs, one that communicates using the DIS (Distributed Interactive Simulation) protocol, and one that communicates using RPR FOM HLA (High Level Architecture) protocol, composed within an instance of an IE to act as a DIS-to-HLA gateway.

The current ModISE framework design requires that MTMs be packaged as Java .jar files, and that they communicate via a specified API. This API is actually a subset of the IEEE 1516 Java API. In other words, the MTMs must be designed to act like HLA federates. In fact, to operate within the ModISE framework, MTMs must not only adhere to the selected subset of the IEEE 1516 HLA API, but they must also have an object model definition (.omd) file, as specified in the HLA specification, to go with them. A composition of MTMs within an instance of an IE forms a sort of mini internal HLA federation. The IE can then represent this composition as a single "federate" to the external environment. It should be noted here that the ModISE architecture is designed to be flexible enough to allow replacement of the HLA based API with another, if desired by the developer. The HLA was used in the first implementation of the ModISE because it is currently the DoD standard for modeling and simulation.

Implementing the MTMs as described above helps maximize their reuse potential and their flexibility. Communication with a particular MTM is done only through its interface, which is specified in its object model definition. There are no direct method calls

between MTMs. ModISE API calls (equivalent to a subset of the IEEE 1516 HLA API), are handled by MTM interface components within the IE, much the same way as the RTI (Run Time Infrastructure) handles HLA API calls from federates. Each MTM gets its own dedicated interface component. These components route data to other MTM interface components and to the external environment via a dispatcher component in the IE as necessary to facilitate the composition specified in the ESpec (See figure 2). The figure illustrates data being routed from MTM A to MTM n, then out to the external environment. This routing is set up dynamically by the framework to facilitate connectivity based upon the graphical composition created by the user.

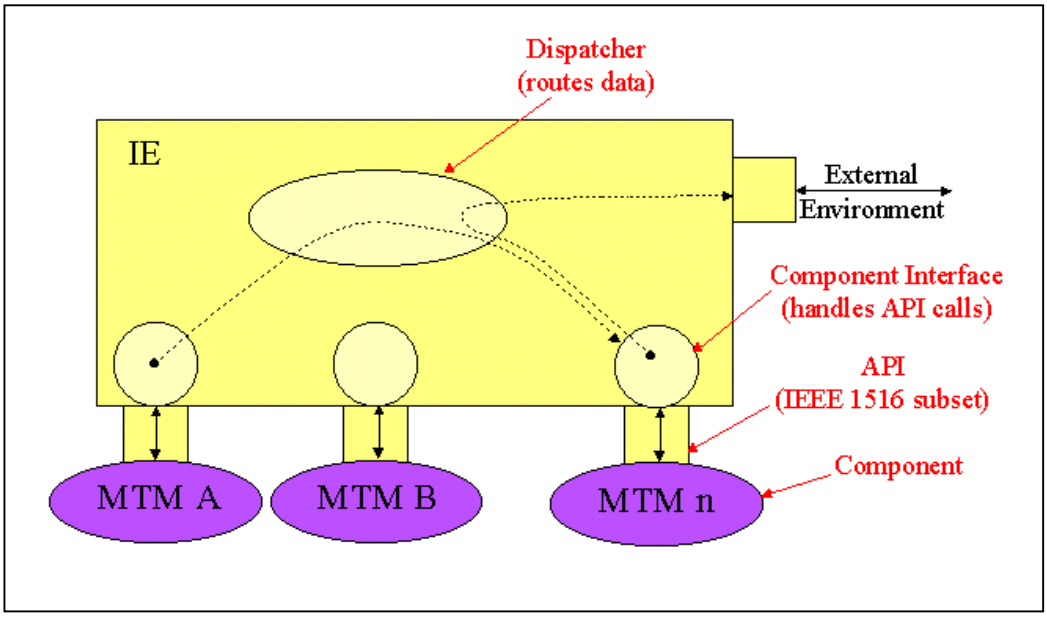

Figure 2: ModISE Component Connectivity

Verification of Component Compatibility

We can consider interoperability and compatibility verification as an instance of the more general problem of testing for software errors. In the software engineering literature, there are three approaches for dealing with errors: *avoidance* (keep errors from happening prior to run-time), *detection* (determine that an error has happened at run-time), and *tolerance* (gracefully handle a detected error at run-time). In the case of simulation interoperability, we can see that the traditional approaches to simulation interoperability (namely, those based on communicating network state between simulations) fall into the avoidance category. No significant work has been done on methods for interoperability error detection or tolerance in simulation. This is a potentially fruitful area of research because it would increase system flexibility and provide the first new techniques for approaching interoperability in a decade of research. The key to defining and applying interoperability error detection and tolerance is to build these mechanisms into a framework such as the ModISE.

When composing applications, the ModISE framework facilitates verification of compatibility among connected components in several ways.

First, the architecture is designed to allow a user to browse, search and filter components for selection based upon a variety of criteria. This might include fidelity level categorizations, SOM/FOM compatibility, I/O requirements, etc. The framework allows users to select individual components for closer inspection by providing access to descriptive information about each component. These capabilities provide for a human-in-the-loop verification of component compatibility.

Given a situation where the component repository contains a large number of heterogeneous components to choose from, human inspection may not be enough to ensure interoperability among selected components. There may be technical

characteristics, not available for review by the user, which keep some components from being compatible.

For this purpose, the ModISE also includes some automated interoperability verification mechanisms. The framework can compare the object model definitions of two components that the user selects to connect together. It can compare attributes and interactions that are specified in the corresponding object model definitions. The framework allows communication between components for any attributes or interactions whose definitions in the object model matches exactly in every detail, including spelling of the attribute/interaction names. This allows interoperability among components, even if they do not have the same SOM (Simulation Object Model, as defined in the HLA specifications), but only for those attributes/interactions that are common between them. The framework will also notify the user of instances where differences between component SOMs are discovered.

Finally, the ModISE framework requires that each MTM, in addition to having an object model definition, also have an XML based metadata description of functionality, interoperability requirements, fidelity level, and similar information. The purpose for this is to evaluate compatibility factors that are not captured in the object model. This part of the framework is still under development, and the solution is non-trivial. The goal is to provide metadata templates for applications, or perhaps classes of applications, that allow the framework to perform a normalized compatibility comparison of the specifications of selected components. While the framework can possibly detect potential incompatibilities, it cannot be expected to make a pass/fail determination. The ModISE framework research is focusing on enabling detection of potential compatibility mismatches and then informing the user. The user will make the final pass/fail decision.

As an example, consider a case where a user selects a radio system component as part of a combat battle tank platform application composition. The radio model metadata indicates that detailed information about the terrain needs to be provided by the environment in order to accurately calculate signal propagation. In this case, assume that the operator selects an environment database access component that cannot provide this information. From examining metadata information for the two components, the ModISE framework detects a potential compatibility mismatch. Now let us assume that the user only wants to use this composition for testing a trainee for vehicle maneuvering procedures, and radio signal propagation is not an issue. Hence the user can make the decision to accept the fidelity mismatch in this case and continue with the composition.

Framework Summary

The framework description illustrates a concept that can enable flexibility and extendibility in an application design, and it can do it in a way that is uniform and reusable across applications and across application domains. The ModISE framework facilitates component reuse, application configuration and reconfiguration, external protocol translation, component compatibility and application interoperability checking and verification. These qualities would not only improve platform level application designs, but other types of distributed M&S applications as well.

PLATFORM ENTITY DESIGN

The concept of developing an application around a framework like the ModISE is not specific to platform entity application design, but it is an important concept to consider for platform application design. This section will investigate other issues related to platform entity design, some of which will be somewhat specific to the platform domain, and some of which will have cross-domain applicability. The discussion assumes a framework-centric design approach, and the issues will focus around building a platform application using this paradigm.

System Of Systems

There are many ways to approach the design of a platform application. An entire entity could be designed as a single MTM (a helicopter simulation perhaps), but this may not be the most desirable approach. One of the goals of the ModISE framework is to enable an application developer to easily implement and take advantage of component-oriented design. The idea is to be able to reuse components and to use the framework to configure and reconfigure applications to fit changing needs.

The prototype combat vehicle that was developed for the ModISE research project was approached from a system of systems perspective. That is, the design focused on building MTMs to represent major subsystems of the vehicle, which when composed, forms the complete system (See fig. 3).

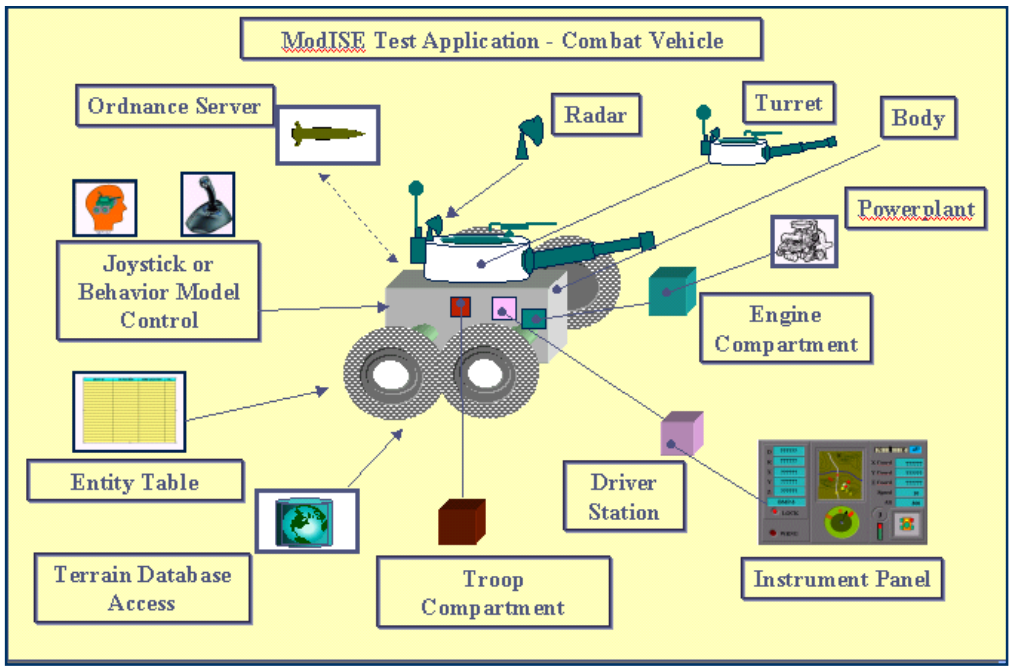

Figure 3: ModISE Test Application Composition (Combat Vehicle)

Consider a few scenarios for this type of design.

Perhaps the combat vehicle simulation is to be used to test a new engine/drive train design. This would require more of an engineering level type of composition, with modeling of such things as mechanical connectors and fittings, suspension, hydraulic and electrical systems, and compression ratios.

At some other point in time, perhaps the combat vehicle simulation is to be used to test out a theory for a new tactical maneuver by having a man-in-the-loop take the vehicle through a virtual scenario. This situation may require a different focus. So maybe the vehicle could be composed with simpler but less CPU-intensive drive train components to achieve a higher frame rate for operation in the synthetic battlespace. This application may, however, require a high fidelity radar model component to accurately represent the vehicle's tactical performance in the synthetic battlespace.

As a third example, assume we want to use the vehicle simulation to see how this new tactical maneuver might affect the outcome of a particular hypothetical battle scenario. So we want to use it in a constructive wargaming application in this case. For this use, we may want to compose the vehicle to operate without the need for human intervention, perhaps in a faster-than-real time mode.

These examples highlight the benefits of function and fidelity flexibility implemented through a reconfigurable component based design.

Taking the system of systems concept to the extreme, a design could be broken down to a very minute level of detail. Each nut, bolt, knob and switch could theoretically be modeled as individual MTMs. This is a challenging notion, given the state of the art in computer technology, but how absurd will it be in ten years, or perhaps fifty? Many systems being built today, as well as some that are already in service, will probably still be operational at that time and will require life cycle support and training for operators. Will the models and simulations that were built to support those systems be able to accommodate improvements in technology and support the system through its life cycle?

The point is that there is an advantage to taking an approach that allows reconfiguration, flexibility and extension to accommodate a range of fidelity levels. There is advantage in taking an approach that

maximizes reuse of components and interoperability standards to minimize mismatches of fidelity and reinventing of the wheel. There is advantage to implementing a design that is evolutionary in order to take advantage of improvements in technology.

For several years, the Department Of Defense (DoD) has been guiding an initiative for Simulation Based Acquisition (SBA). The goal of the SBA initiative is to improve acquisition of DoD products using M&S as an enabler. SBA requires consideration of the entire life cycle of a product, from cradle to grave.

SBA drives the need to consider design of platform level simulation applications as a system of systems in a way that maximizes potential for cross-domain interoperability and reuse. This is a non-trivial issue that warrants attention, because the system of systems approach is being used by application developers. Without coming to consensus in areas where standardization can be achieved, incompatibility will be the norm rather than the exception.

Control Mechanisms

An important factor in any platform level application is the control mechanism or mechanisms that guide its behavior in the synthetic battlespace. If we think about it, the main difference between a man-in-the-loop platform simulation and a SAF (Semi Autonomous Force) application is the control mechanism. A SAF entity requires behavior modeling and/or scripting capability because there is no human to operate it at run time. On the other hand, a man-in-the-loop simulation requires modeling of instrument panels and interactive controls to provide a human interface.

Thinking about this from a system of systems perspective, it would be beneficial if we could design a platform simulation in such a way that its control mechanisms are de-coupled from the rest of the application models.

The ModISE Combat Vehicle prototype, for example, was designed this way. The vehicle can respond to a variety of control mechanisms. These include a behavior model MTM for the vehicle to operate autonomously, a joystick controller MTM to allow control of the vehicle by a man-in-the-loop operator, and voice control MTMs to allow the vehicle to be controlled through audible commands from the operator or even third parties, such as instructors. A particular instance of the combat vehicle can be composed to accommodate control by any combination of these mechanisms without having to reconfigure the other components, such as the troop compartment or the engine/drive train.

Again addressing the issue of general platform simulation design, there seems to be a case for accommodation of a variety of control mechanisms, and for standardization of these control mechanisms with respect to interoperability.

A joystick controller component was mentioned above, and conventions do exist for joystick I/O. The joystick MTM developed for the ModISE Combat Vehicle will interface with any standard off-the-shelf joystick. The ModISE combat vehicle is controlled by a driver interface component. The driver interface can in turn be influenced by either the joystick component, a voice recognition component, a behavior model component, or any combination of these.

The interface between the joystick MTM component and the driver interface component is defined as a set of attributes and interactions in the object models of the two components. This interface specification is complimented with an "understanding" between the two components as to the meaning of these attributes and interactions and how they are to be used to modify the vehicle behavior.

Another control mechanism mentioned above was the voice control mechanism. This mechanism was designed using off-the-shelf voice recognition technology, an application to produce digitized voice as text strings in an HLA environment, and a set of voice context recognition MTMs to analyze the text string for specific command sets.

The vehicle behavior model itself was mentioned as a control mechanism. This is the component that allows an instance of the ModISE Combat Vehicle to operate on its own, as a SAF. This component is not mature, but the idea is to provide logic in the behavior model for the vehicle to operate autonomously according to a set of priorities and situational evaluations. The behavior model is currently designed so that the combat vehicle can avoid terrain obstacles when travelling in the synthetic battlespace. It does this by communicating with a terrain database access component, which can perform route planning. The behavior model component also interfaces with the driver interface component in much the same manner that the joystick does. Behavior modeling is

of course an area of study unto itself, and a detailed discussion of it is beyond the scope of this paper.

For the purpose of the ModISE Combat Vehicle, the control components focused mainly on facilitating maneuvering of the vehicle. In the general case of platform entity design, many other control mechanisms could also potentially be implemented for this purpose. Some of these might include keyboard I/O commands, virtual or hardware-based instrument panels, navigation computer control, or target tracking system control, to name a few.

Beyond the idea of platform maneuvering, there are many other areas where the development of standardized reusable control mechanism components could be applied or utilized in platform entity application design. Examples include control mechanisms for system fault insertion, system and subsystem operational settings, synthetic environment management (weather, dynamic terrain, etc.), simulation exercise management (start, stop, freeze, etc.), trainee/student performance monitoring, data collection, and many more. Using the system of systems design approach, and considering enablers like the ModISE framework, there may be benefit in designing control mechanisms as standardized reusable components.

Server vs. Dedicated Functionality

The concept of distributed simulation means that applications do not have to be co-located on the same hardware platform. For that matter, a single application may be distributed across multiple platforms. Perhaps there are advantages to allow distribution of subsystems or system functions. The idea behind a server application is that a particular system function can be managed uniformly from a single source instead of by individual entities.

The ModISE Combat Vehicle prototype design included the idea of implementing an ordnance server to facilitate handling of ordnance trajectories and missile fly-outs. This functionality is not yet complete, but the general idea is to implement an ordnance management MTM that can handle ballistic trajectories, guided missile management, or fire-and-forget missile management. This MTM could be coupled with individual MTMs that provide models for individual missile or ordnance types. The ordnance management MTM could act as an information broker between individual ordnance models and the synthetic battlespace environment. The ordnance management MTM would pass synthetic battlespace information to the individual ordnance MTMs (target positions, wind/air temp info, terrain masking info), and it would in turn get missile position/detonation information from the individual ordnance MTM(s) and pass it on to interested parties in the synthetic battlespace. The user could, via the ModISE GUI, compose the server application by selecting and connecting the ordnance manager MTM, all desired ordnance model MTMs, and any required protocol MTMs to allow the server to participate in the federation.

An alternative to the server application approach might be to include the ordnance server components as an embedded part of a particular platform entity application, to effectively be a dedicated ordnance server for that specific entity. The actual MTM components for these two different approaches would be very similar and, if designed properly, the same set of components could accommodate either approach.

This server-verses-dedicated approach could be applied to many components that are relevant to a platform entity simulation system of systems design. Some examples include component sets for sensor systems, communication systems, navigation systems, weapons systems, logistics models, and many others.

Some functions may be better served with a dedicated approach. Examples of these include instrument panels, or any type of component that does not have the potential to provide a common service.

Examples of components that might be better implemented in a server application include a weather model, or any type of component that is applicable to the synthetic battlespace as a whole.

In many cases, components can be designed to optionally function as a server or in a dedicated role. This approach facilitates a greater degree of flexibility and gives users more options to configure a synthetic battlespace to meet their needs.

De-Coupled Components

Implementing components as specified in the ModISE framework has the effect of de-coupling them, at least to some extent. In other words, MTMs do not communicate via direct method calls to other MTMs. This has several ramifications with respect to platform entity design.

Frame processing is one area of interest. Frame processing has to do with the repetitive tasks that an application needs to perform to interact in the synthetic environment. This includes sending and receiving data, processing models and operator input, and basically everything that the simulation does. The trick is to do all of these things fast enough to stay synchronized in time with other activities going on in the environment. An application that can't keep up with the minimum required frame rate is useless. The minimum frame rate is different for different types of applications (aircraft simulations verses ground vehicle simulations, for example), and for different simulation time constraints (real time verses faster-than-real-time). This is an area that is particularly challenging for the sort of component-based design paradigm that is being examined in this paper.

Traditional platform entity simulation applications are often designed to function in a single process space. Communication between object instances or between subroutines is via method calls. Though a bit of an over simplification, a typical frame might flow as follows:

- Receive updates from external environment (new entity positions, weapon fires and detonates, etc...)
- Update internal entity table (with new info from external environment).
- Perform dead reckoning on any entity positions for which updates have not been received by the external environment for this frame.
- Update the model for the local entity or entities (maneuver, weapon fires/detonates, sensors, damage evaluation, etc...)
- Update internal entity table (with local entity info that has changed during this frame) .
- Send local updates to the external environment.
- Repeat frame as quickly as possible or at least as fast as minimum required frame rate (60hz is a typical figure for aircraft simulations).

While this is by no means the only approach to frame processing, it does represent a general sort of approach used by many platform level simulation applications. The thing to point out here is that, when the frame processing is managed in a synchronized fashion, the various subsystems of the entity can count on well-correlated data. For example, an aircraft entity can use this approach to update its avionics, propulsion, airframe, and electronic warfare subsystems during any particular frame with confidence that the point in time from the perspective of the avionics system is the same point in time from the perspective of the electronic warfare systems.

The ModISE Combat Vehicle was implemented a little differently, using a de-coupled component approach. In this case, each component was designed to operate independently from the other components, at least with respect to timing synchronization. The engine/drive train component, for example, gets acceleration and braking inputs from the man-in-the-loop joystick component via a driver interface component, without making any assumptions as to when this data arrives. Similarly, it gets inputs from a terrain database access component to evaluate terrain conditions for traction and incline. It calculates engine RPM values, torque to wheels, etc., and sends these updated values out as quickly as possible to any other components that have expressed an interest in receiving this information by subscribing to it. There is no requirement for the acceleration setting, for example, to be received by the engine component at a specified rate or at a specified time. The engine operates autonomously, taking data as it comes. Because of this asynchronous mode of operation, the point in time form the perspective of the engine/drive train MTM may be *T ms*, while the point in time from the perspective of the joystick MTM may be *T+1ms,* and the point in time from the perspective of the database access MTM may be *T-1ms*. In the case of the ModISE engine component, this could mean that the engine, whose internal clock thinks the time is *Tms* is using an acceleration setting from time *T-2ms* along with terrain traction and incline information from time *T+3ms*. In other words, the engine is using an acceleration setting for 2 milliseconds in the past, and is applying it to a terrain polygon that, at least according to its local simulation time, the vehicle will not reach for another 3 milliseconds. This sounds odd, but it is the sort of thing that can happen when processes are not synchronized. Under normal conditions this situation will not pose a problem for the ModISE Combat Vehicle, but it could if the delta in reference times between components gets too large. If the engine/drive train component was designed as a high fidelity model to be used in an engineering simulation, then the asynchronous time delta effect would most likely be much more critical and could potentially create a serious problem.

The ModISE in no way prohibits a user from implementing an application as a single MTM, which would be able to regulate the interoperability among subsystem functions. This way the user could duplicate the synchronously controlled frame

approach outlined earlier. On the other hand, the benefit of implementing the application as a set of loosely coupled components is that it increases the potential for reuse of the components and increases the functional flexibility of the application. If synchronization among components for frame processing was critical, then perhaps a synchronization component could be implemented to handle this, while still allowing subsystem functions to be modeled in separate de-coupled MTMs.

As an illustration of the benefit of de-coupled components, consider the following example. Assume that a data window MTM has been designed as a test tool, to display current values of attributes and interactions for specific federation objects during run time. Assume that this data window MTM can be controlled by voice, so that an operator can select a federation object to be displayed and can also select the set of attributes/interactions to be displayed. This data window MTM can adapt to any SOM or FOM, but it can only display SOM-specific data if it can "see" inside of the federate for which the SOM applies. The reason for this is that a particular federate can by definition potentially produce any data that is in its SOM, but for the purpose of a particular federation exercise, it will only produce data that is specified in the FOM. This does not mean that the federate doesn't update SOM attributes and interactions internally. However, if the data window MTM had some means to get at those internal updates, then it could theoretically display that data for observation at the user's request. If the federate in question were designed as a set of composed MTMs within the ModISE framework, then the collection of composed MTMs forms a sort of mini internal federation. In this case, an instance of the data window MTM could be composed into the application and could then simply subscribe to the data selected for display.

There are many complex issues related to implementing a de-coupled component design. This approach imposes limitations, but also enables flexibility and extendibility.

Scalability

Always an issue of concern to the M&S community is the issue of scalability. No matter how many objects can be supported in a particular federation, there will always be a pending need for more. But at any given point in time, CPUs only have so much processing power, and networks can support only so much bandwidth. So beyond improving the physical infrastructure upon which federations execute, what can be done to help with this issue of scalability?

One answer may be in framework-based component-oriented design. One of the areas to be addressed with the ModISE framework is that of dynamic transfer of components during run time. In other words, we want the ModISE framework to be able to transfer execution of a particular MTM or set of MTMs to another computer during run time to decrease the work load on the computer that is currently executing these MTMs. It must do this in a way that is non-disruptive to the run time execution. Since the ModISE MTMs are for all intents and purposes equivalent to HLA federates, there is no reason why they cannot be run from different machines. In other words, a composed application could theoretically be distributed, with various components running on different physical computer platforms. Mechanisms could be designed into the framework to facilitate this and even manage a dynamic reconfiguration of components at run time to improve performance or to scale an application as necessary to deal with increased work load.

Consider the following example. In a particular application running on machine 1, MTM A is connected to MTM B within an instance of the ModISE IE. The IE at some point detects that performance is slowing down on machine 1 and realizes that the application will not be able to maintain a minimum frame rate. The IE decides to offload MTM A to another platform to reduce load. Through some control mechanism, the IE performs handshaking with another IE instance, which is running on machine 2, and negotiates the startup of a second instance of MTM A. Once the duplicate of MTM A is up and running, the IE on machine 2 initializes this new instance of MTM A so that its state matches that of the first instance of MTM A. Then the IE on machine 1 performs some internal switching to route data from the new MTM A on machine 2 to MTM B on machine 1 and at the same time kills the instance of MTM A on machine 1.

The example above is meant to illustrate the idea that handling dynamic run-time transfer of components via the ModISE framework could be done with little or no impact on application component design. It is functionality that has not yet been implemented by the framework, but it is a possibility for a future upgrade. Using this approach could facilitate scalability in an application design,

perhaps even dynamically to adjust for changing performance conditions.

Metadata

The topic of metadata was mentioned earlier as a mechanism to support verification of component compatibility. If metadata for different "types" of components could be specified in a uniform way, then perhaps a framework such as the ModISE could to some extent automatically determine compatibility between components and alert the user to any discrepancies. For this to work, metadata template standards would have to be developed for various component categories, which in turn requires a determination of component category distinctions. This is a challenging issue.

Consider an engine/drive train component for a combat vehicle simulation application. Should it be categorized as engineering, engagement, product, or by some other category? Are these categories too general? Could a component fall into more than one category? Are there situations where it might be desirable to compare matching template fields of components from different categories? There are many basic and difficult questions that need to be answered though research before metadata template standards and framework capabilities that can use these templates can be developed.

The ModISE framework research is attempting to address this issue, and it is hoped that more lessons learned will be available in the near future.

Metadata is the key to interoperability verification, but there is a lot of work to be done to better facilitate its impact on M&S application design.

Summary and Conclusions

Platform level simulation design is an issue that is very important to the M&S training community. Advancements in technology have enabled a framework-centric design paradigm for simulation applications in general, and the implications are significant for platform level simulation implementation.

A STRICOM research project developed an interoperability framework called ModISE. A platform level prototype application was also developed for the purpose of testing this framework. Many observations and lessons learned from this research have been collected and outlined in this paper, in order to provide ideas and issues for consideration by the M&S community.

A summary of the issues covered is as follows:

- Framework-centric design
  - Graphical composition
  - Component reuse
  - Distributed communication interoperability
  - Component compatibility verification
  - Application configuration & reconfiguration
  - Application flexibility and extendibility
- System of systems design
- Control mechanisms
- Server verses dedicated functionality
- De-coupled component design
- Scalable solutions
- Metadata

The ModISE is a continuing STRICOM effort, and the framework technology is enabling solutions for distributed M&S application interoperability that were not previously available. This research shows promise for advancing the state of the art in platform level simulation application design, and for other application domain areas as well.

REFERENCES

[1] SISO-STD-001.1-1999 (RPR FOM 1.0)

[2] 1516-2000 IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Framework and Rules

[3] M. Biddle, C. Perry (2000). An Architecture for Composable Interoperability Simulation Interoperability Workshop, Fall 2000. www.sisostds.org

[4] R.W. Franceschini, T. Clarke, B. Goldiez, A. Griffin, A. Huthmann, G. Schiavone, and B. Schricker. (2000). A survey of the theory and practice of simulation interoperability. Proceedings of the Spring 2000 Simulation Interoperability Workshop, Orlando, FL, March 26-31.