JEWEL – M&S Environment for the SAF

Thio Seng Joo, Kong Siew Theng, Tan Siew Fang, Yeo Loon Chew
Defence Science & Technology Agency
Singapore
tsengjoo@dsta.gov.sg, ksiewthe@dsta.gov.sg, tsiewfan@dsta.gov.sg,
yloonche@dsta.gov.sg

ABSTRACT

As a small nation, Singapore has limited human resources, land and airspace. The strategic use of M&S to help us overcome these constrains is therefore crucial. We thus formulated our simulation masterplan, called the Vision for SAF Simulations (VSS) back in the mid-90s, and JEWEL (Joint M&S Environment for Wargaming & Experimentation Labs) was conceived as the simulation environment in support of VSS. This ***enterprise-wide*** approach to simulation is very much analogous to what is happening in the business and C2 worlds.

Designed with reusability and interoperability as its primary precepts, JEWEL would be an ***open software environment*** that allows the incorporation of new technologies and standards from governmental, commercial and / or R&D bodies, and would be a launching platform from which new application needs can be satisfied accurately and quickly. To maintain openness and as a result future proof, DSTA believes that substantial attention must be devoted to its information architecture, both in terms of representation as well as content, as demonstrated in our adoption of HLA and XML, among other standards. JEWEL would support the SAF in training, analysis, experimentation and acquisition.

The first part of this paper is dedicated to JEWEL, its development rationale, philosophy and overall structure. Part two focuses on the key design considerations of the Distributed Simulation Engine (DSE), the core component of JEWEL. In the third part, the Joint Battle System (JBS), which is based on JEWEL and is being used by the SAF Centre for Military Experimentation (SCME), will be introduced to exemplify the key capabilities of JEWEL. Finally, some elaboration on potentially fruitful future directions will be attempted, based on known technological and application trends.

ABOUT THE AUTHORS

Thio Seng Joo is a Project Manager with the Defence Science & Technology Agency (DSTA). He graduated with a Bachelor degree in Computer Science and Master of Technology in Knowledge Engineering from the National University of Singapore (NUS) in 1996 and the Institute of Systems Science (ISS), NUS in 2001 respectively. Since 1996, he has led the development of complex simulation systems for training and experimentation, and worked with various research institutions in Singapore to develop future capabilities. JEWEL is one of his latest endeavours.

Kong Siew Theng is a Senior Engineer with DSTA. She has a Master in Computing from NUS in 1998 on distributed simulations. She is one of the key developers of simulation engine in JEWEL and led the development of the HLA component of JEWEL to support interoperability of JEWEL-based systems with external simulators. She is also involved in the design and development of the C2 simulation for JEWEL.

Tan Siew Fang is a Project Leader with DSTA. She graduated with a Bachelor degree in Engineering and Master of Science in Communication Software & Networks from the Nanyang Technological University of Singapore (NTU) in 2000 and 2005 respectively. Since 2000, she contributed to the development of complex simulation systems for training and experimentation, and worked with various research institutions in Singapore to develop future capabilities. JEWEL is one of her latest endeavours.

Yeo Loon Chew is a Project Leader with DSTA. He graduated with a Bachelor degree in Electrical Engineer from NUS in 2003. Since then, he has contributed to Computer Generated Forces (CGF) development in DSTA, with a primary focus on Air CGFs. He is currently also looking into the future enhancement of JEWEL.

JEWEL – M&S Environment for the SAF

Thio Seng Joo, Kong Siew Theng, Tan Siew Fang, Yeo Loon Chew
Defence Science & Technology Agency
Singapore
tsengjoo@dsta.gov.sg, ksiewthe@dsta.gov.sg, tsiewfan@dsta.gov.sg,
yloonche@dsta.gov.sg

INTRODUCTION

As a small nation, Singapore has limited human resources, land and airspace. The strategic use of M&S to help us overcome these constrains is therefore crucial. We thus formulated our simulation masterplan, called the Vision for SAF Simulations (VSS) back in the mid-90s, and JEWEL (Joint M&S Environment for Wargaming & Experimentation Labs) was conceived as the simulation environment in support of VSS. This ***enterprise-wide*** approach to simulation is very much analogous to what is happening in the business and C2 worlds.

Designed with reusability and interoperability as its primary precepts, JEWEL would be an ***open software environment*** that allows the incorporation of new technologies and standards from governmental, commercial and / or R&D bodies, and would be a launching platform from which new application needs can be satisfied accurately and quickly. To maintain openness and as a result future proof, DSTA believes that substantial attention must be devoted to its information architecture, both in terms of representation as well as content, as demonstrated in our adoption of HLA and XML, among other standards. JEWEL would support the SAF in training, analysis, experimentation and acquisition.

The first part of this paper is dedicated to JEWEL, its development rationale, philosophy and overall structure. Part two focuses on the key design considerations of the Distributed Simulation Engine (DSE), the core component of JEWEL. In the third part, the Joint Battle System (JBS), which is based on JEWEL and is being used by the SAF Centre for Military Experimentation (SCME), will be introduced to exemplify the key capabilities of JEWEL. Finally, some elaboration on potentially fruitful future directions will be attempted, based on known technological and application trends.

JEWEL

Motivation

The primary motivation behind JEWEL is the anticipated increase in demand for M&S in the SAF to support experimentation, training and operations. We can no longer afford to rely on the traditional approach of developing stovepipe simulation systems for each application. A more radical approach that involves the development of a common, interoperable and shared M&S environment will be required, with these long-term benefits:

- **Reduced Cost, Shorter Time-to-Deploy, Reduced Risk.** Through reuse of common components and models, the cost to develop new M&S applications would be greatly reduced, as we need only develop the deltas to meet the specific user requirements. This, in turn, translates to shorter time to deploy. Project risks would also be reduced, as we are reusing tested components from similar projects.

- **Meeting User Requirements "On-Demand".** With composability built into the environment, the systems deployed will be more flexible and re-configurable to meet different experimentation requirements, on-demand.

- **Greater C4I-Sim and Sim-Sim Interoperability.** The focus on shared conceptual and data models, common interoperability standards and components will enhance interoperability among simulations, and with the C4I systems.

- **Improved Consistency.** Standardization facilitates the verification and validation of models. Reuse of these validated models will enhance the consistency of outcome across our M&S systems, which is key if M&S were to aid decision-making.

- **Extended Systems Shelf Life.** The adoption of an enterprise architectural approach rather than the stovepipe system approach will lend greater manageability to continuously upgrade our systems in order to keep up with new technology and standards, thereby extending the shelf life of these systems.

Anatomy of JEWEL

JEWEL is a collection of data & interface specification standards, frameworks & tools, and composable models & databases, as shown in Table 1. The rows of the table is explained as follows:

- **Repositories of Composable Models & Databases.** This layer refers to the physical set of reusable models and databases that are developed or acquired to perform specific simulation functions.

- **Frameworks & Tools.** This layer consists of the technical frameworks that provide the glue essential for the different simulation tools to integrate seamlessly with one another. Some of these tools are COTS while others are custom-developed for the JEWEL environment.

- **Data and Interface Specification Standards.** In order to enhance the level of interoperability among simulation systems, and between simulation and C4I systems, it is important that these systems adopt consistent data and interface specification standards. This would minimize the problems associated with interpretation and mapping of data as a result of representation differences.

Across Table 1, we account for all activities during the M&S lifecycle. The columns are explained as follows:

- **Modelling.** Model development poses great challenges to system developers, as it requires both the technical expertise of programmers and the domain knowledge of subject matter experts. Maintaining models as technology changes is also an issue. The direction is therefore towards a formal development approach whereby models are captured at the conceptual level, independent of the underlying platforms and technologies. The generation of platform-specific codes should be automated as far as possible.

- **Composition & Integration.** BattleSpace composition adopts the "lego brick" paradigm for M&S system development. It stems from the desire to re-configure and reuse systems through plug and play. It is often a stated system objective among the more recent M&S developments such as OneSAF and is a result of the advancements made in component-based technologies.

- **Deployment & System Configuration.** M&S systems today do not normally operate in a standalone manner. They are often embedded within C4I systems, and operate within a federation comprising of other simulation systems. HLA is the current technology for integration among simulation systems. The technology is still being enhanced to address some of its current limitations. Integration between M&S and C4I systems is traditionally achieved through the use of dedicated gateways. Increasingly, web technologies are being considered as the enabler for greater Sim-Ops integration.

- **Simulation.** This is the focal point of all simulation environments as it is where the actual simulation is being executed. All the other stages exist to support the runtime simulation. The key issues that characterise this stage are mainly performance-related, such as the scalability of the specific runtime simulation architecture and the level of interactivity. A simulation engine is usually at the heart of the runtime architecture, which in turn is supported by other simulation components such as behavioural or CGF engines.

- **Analysis.** Analyses are conducted in order to maximise the values of the simulation runs. Standard after-action review features include the ability to perform record & playback. More advanced capabilities would include the ability to use COTS tools for statistical analysis as well as data mining of data captured during simulation runs.

We devote the entire next section to DSE, as it is the core component of JEWEL, and plays pivotal role in our endeavor to achieve reuse and interoperability. DSE also embodies high performing design and implementation decisions, in order to satisfy the disparate needs of various M&S communities (e.g. experimentation, training).

We aspire to share on the other JEWEL components in the near future.

Table 1. Anatomy of JEWEL

	Activities				
	Modeling	**Composition & Integration**	**Deployment & System Configuration**	**Simulation**	**Analysis**
Repositories of Composable Models & Databases	❑ Models (& Specifications) Repository ❑ Sim-C4I Interface Maps Repository				
	❑	❑ Frameworks & Tools Repository ❑ RPR FOM ❑ Battlespace Repository			
	❑		❑ System Configuration Repository		
	❑			❑ Simulation State Repository	
Frameworks & Tools	❑ Integrated Model Development Environment ❑ Terrain Development Tools	❑ On-Demand Synthetic Battlespace Composer	❑ System Configuration Editor	❑ Scenario Editor ❑ ***Distributed Simulation Engine*** ❑ CGF Framework ❑ Injects Framework ❑ Assortment of GUIs	❑ Data Analysis Console ❑ After Action Review Tools ❑ Data Analysis Tools
Data & Interface Specifications Standards	❑ Models Specification ❑ Sim-C4I Interface Maps Specification				
	❑	❑ Tools Interface Specification ❑ HLA FOM Specification ❑ Battlespace Specification			
	❑		❑ System Configuration Specification		
	❑			❑ Simulation State Specification	

DISTRIBUTED SIMULATION ENGINE (DSE)

Introduction

DSE is a software framework that provides the following:

- A set of base classes that represent the common concepts. These classes define by default a set of APIs that different components of applications may use to interact with each other (see later sections). These base classes can be used to define application specific concept types.
- A simulation kernel that keeps track of time and orchestrates execution of all application components.
- Communication capability that handles networking issues between processes built on DSE.
- Distributed database services for storing runtime instances of all concepts.
- Avenues for attaching commonly used repositories or libraries of functions. Examples are assets definition database, scenario loader/writer, simulation state record/playback module, environment database, etc.

Key Concepts Defined

We defined four main concepts that represent the software equivalent of the real-life battlefield:

- Entity. This refers to any object that has the capability to act and react, which can include brigades, squadrons, tanks, soldiers, task forces, missiles, radar, etc. Entity contains attributes that define its characteristics, such as the amount of fuel, number of missiles, travel velocity, level of casualty, physical shape and size, etc.
- Model. Whereas entities represent the objects that can act and react, models define how they perform such action and reaction. For instance, an action of one entity is to fire a missile, and the reaction of another entity is perhaps to compute the damage sustained. This concept includes the following sub-concepts:

- Behavioral Model. Analogous to the human brain, this defines the decision-making processes of each entity. For instance, an aircraft performing CAP may have to deliberate between engaging an approaching enemy fighter or not.

- One-time Action Model. Action models implement the exact physical level activities that are performed by the entities. Distinction is made between activities that must be performed throughout the lifetime of the entities and those otherwise. One-time Action Model refers to the latter.

- Repetitive Action Model. This defines activities that must be performed throughout the lifetime of entities.

In addition to representing entities' behavior, the model concept also extends to modules that display GUI that interacts with users and modules that handle the networking aspects of DSE.

Models in DSE are classified into two categories: Global or Entity-based. Entity-based models contain codes specific to individual entity (e.g. models that implements the action of an entity), whereas Global models maintains visibility of all entities (e.g. User Interface models that provides avenues for users to control the simulation).

- Event. Events represent the "happenings" in the battlefield. Events are created when something happens that may be of significance to other entities. The models that perform sensor operations usually receive these events.

- Command. Commands represent instructions or orders given to the entities. For example, an order to attack an enemy position will include the identification information of that position, contained in a command object. This concept includes the following sub-concepts:

  - Behavioral Command. Commands targeted for Behavioral Models.

  - Action Command. Commands targeted for Action Models.

  Command is also a convenient mechanism for status reporting. That is, the recipient of a command can use that same command to convey the status of task completion back to the command issuer.

DSE was written in C++, and was designed under the premise that all simulations can, and should be built by simply 'specializing' the base classes ('class inheritance' in C++ nomenclature) of these concepts. Thus, creating a new model class by inheriting from the Model base class (defined in DSE) and inserting additional attributes and logic (by means of overriding virtual functions) that are peculiar to how an F16 aircraft maneuvers would create an F16 aircraft motion-dynamic model.

Design Considerations for Scalability

Scalability Defined. Scalability refers to the ability of a simulation system to support larger scenario. Larger scenario encompasses increase in one or more of the following: number of simulated objects and size of battlefield. In reality, no software scales infinitely, due to the limits imposed by the Operating Systems, processing hardware and network infrastructure.

Objects Count. DSE was designed to represent up to 2^63 objects during each simulation, although the processor speed and network bandwidth will further limit the number of supportable models / processes.

Process Distribution. DSE enables the splitting of a simulation application into processes, each simulating different objects or parts of objects. Such splits can be attained without adding on to the load of the inter-simulation network (typically HLA-RTI network), because DSE implements its own intra-simulation network. The provision of intra-simulation network has these additional benefits:

- Allows more intimate relations between models within the same application, thereby increasing modeling efficiency.

- Able to scale without relying on HLA. The intra-simulation network can also be enhanced independently, should new and better networking technologies surface.

- Enable scalability without affecting HLA, especially during scenario creation. HLA supports runtime co-ordination among applications well, but does not contribute to convenience during scenario creation.

Design for Extensibility

Extensibility Defined. Extensibility refers to the ability of a simulation application to support new kinds of objects. In a sense all applications are extensible, just that extending some requires greater effort than the rest.

General Techniques. DSE is an Object-Oriented software that is highly modularized, with well-defined interfaces that connects to all application components (the concepts, defined earlier). Applications built on DSE must package their models into DLLs (dynamic loadable libraries), so that they can be loaded/unloaded dynamically during runtime. Another benefit of using DLLs is that models can be added without requiring re-compilation/re-linking of any software component.

Extensible Parts. All concepts (defined earlier) are extensible, with DSE. All extensions to applications built on DSE can be performed without resulting in code compilation, except new models (in which case coding and compilation is necessary). This effectively means that non-technical end-users may extend a simulation by adding new entity types (together with their characteristics) without looking at a single line of code. Apart from models, entities, events and commands, databases for scenario and environment representation can also be plugged into DSE easily, so long as the pre-defined APIs are adhered to.

Design for Efficiency

Efficiency Defined. Efficiency is the measure of the throughput of a simulation against time. Basically, the more objects that can be simulated within a fixed time, the more efficient the software is.

General Techniques. Multithreading and parallel execution (by process distribution) are two general performance-enhancing techniques employed by DSE.

Localized Memory Allocation/Reallocation. The time consumed by the operating system during new memory allocation is not predictable, and the delta can range from just a few microseconds to a few milliseconds. To overcome this potential performance letdown, most simulation software reserves a large pool of memory at start up and implement their own localized memory allocation/reallocation routines. DSE went one step further, to also re-use memory lots previously allocated to data of the same type. This further shortened the time needed to allocate/reallocate memory, and paved the way for more efficient logging/networking functions.

Runtime Model. The two well-established runtime models adopted by simulation systems are time-stepped and event-driven models. Conventionally, applications that require high level of user-interactivity adopt the time-stepped model, while non-realtime and non-interactive applications take advantage of the event-driven model for efficiency reasons.

DSE is time-stepped driven with adjustable frame size and rate. One new simulation state is computed every frame. If the frame size is x ms and rate is r, DSE will invoke all application models at x ms interval and the simulation clock will advance by x*r ms, as illustrated in Figure 1.

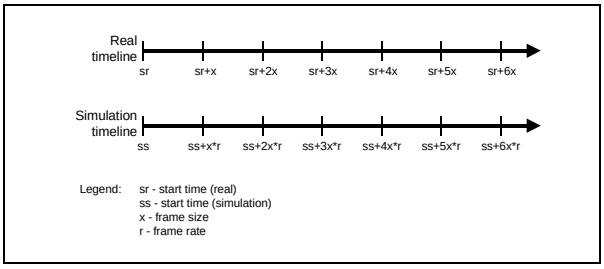

Figure 1. Effects of Frame Size and Rate

Figure 2 depicts the perfect scenario, in which the computation time consumed to calculate each state falls within x milliseconds. For DSE to handle exceptional scenarios, three timing modes were implemented:

- Race Mode. Simulation states are computed one right after another. This is useful for applications that do not require user interaction, and only the final state is of importance.

- Disregard Real Time Mode. If the time to compute state n exceeds the frame interval, computation of state n+1 commences immediately after state n is computed. However, if the time to compute state n is shorter than the frame interval, the kernel will wait until the frame interval elapses completely, before computing the next state. In both cases, the simulation time advances by the same amount – the frame interval. Applications may assume that

each computation computes the next state of objects for fixed time advancement.

- Follow Real Time Mode. Advancement of simulation time is tied to real time, at the specified rate. Computation of one state may be skipped if the previous state requires more than the frame size to compute. Applications must be built to compute states of objects for variable time advancement.

An enhancement to the standard time-stepped model is to implement a hybrid runtime model. This will be elaborated next.

Execution Scheduling Policy. It is important to note that the battlefield is best represented if all models execute in parallel and continuously. However, this is not possible, as it would require one CPU for each model. As such, some scheme of scheduling the execution of the models within each process is necessary.

On top of the basic runtime model, DSE implements an efficient scheduling algorithm that supports advanced scheduling options, as follows:

- Time Schedule Order. Much like event driven simulation, this feature will enable models to schedule their own execution time. However, unlike event driven simulation, execution is carried out only at the nearest later frame time, rather than immediately. Entities, models and in fact any application function can be scheduled at a time decided by the application.

- Variable Frequency. Useful in handling models of differing updating resolution. For instance, the position of a tank on the ground can be updated at one second interval, while an aircraft requires at least 100 ms.

The advanced scheduling algorithm is able to improve the overall efficiency of simulations because of its ability to spread out the invocation of low frequency entities and models. For instance, in a scenario with 2 aircraft and 10 tanks, only 1 aircraft and 2 tanks need to be invoked for computation each frame if the update resolution of aircraft is 200ms, that of each tank is 1000ms, and the simulation frame interval is 100ms.

Design for Reliability

Reliability Defined. Reliability refers first to how infrequent downtimes occur for an application. The lower the frequency, the higher the reliability. Ideally, there should be no downtime at all. As DSE is an extensible software framework (i.e. it will execute together with non-native codes), it is unrealistic to aim for the highest level of reliability (i.e. no downtime). Therefore, DSE provides a mechanism to ensure quick recovery from such downtimes.

Data Replication. There are generally four ways for data to be shared among the processes of distributed applications and each has its pros and cons, as described below. DSE applies the second method:

- Centralized Database. This is the traditional approach, in which the entire simulation database is hosted on one server. All processes connect remotely to access the database. Although slow in access speed, data consistency is maintained with ease. Adopting mirroring technology will remove the single point of failure.

- Replicated Database. Replicating the database across to all processes. While utility of network goes up and maintenance of data consistency is tough, this method incurs low data access latency and allows flexibility in terms of data and scenario size.

- Shared Memory. Using a combination of hardware and software, this solution allows processes across multiple machines to share the same memory spaces. One major drawback of this method is its limited address space and the fact that the failure of any single component will result in complete system failure, although the lowest data access latency can be expected.

- Scalable Parallel / MPI. Housing all processes of one application in one SMP machine that also hosts the database, this method often produces the highest performance for specially designed applications. The drawback is the proprietary status of both hardware and OS, leading to high cost in deployment and maintenance.

Replication technique provides the best compromise in access speed and scalability. With an up-to-date local database, data access by each process can be immediate. Besides, since each process holds a

replicated database, there is no worry of unexpected data loss due to faults affecting just a few processes. However, the main disadvantage of this technique is that redundant data may be sent to a process that does not require them. For instance, an MMI process that handles logistics functions need not know the radar range of an aircraft. Also, updates need not necessarily constitute value changes, and that means additional possibility of network optimization.

While PCs and networking technologies are advancing at such pace that these issues might disappear altogether in due course, the project team also evaluated a few techniques that could reduce the network bandwidth required. These techniques can roughly be categorized into two categories, as follows:

- Filtration-Based. Techniques under this category sought to cut down network bandwidth required based on exploring the 'need' to send.
    - Spatial Vicinity. Adopted by HLA Data Distribution Management (DDM), this technique reduces the data sent to each federate by considering the spatial vicinities of entities. That means, if Entity 1 is beyond the range of interest of all entities in Federate A, updates for Entity 1 will not be sent to Federate A. This technique assumes that there is no need for all federates to store data of all entities, since the area of operations of each federate is small, compared to the entire battle scene.
    - Declaration of Production and Needs. At models/entities level, each object class declares the data they produce and require. Based on these declarations, only data needed by a process is sent to that process.
    - Ownership Migration. By migrating the entities between processes, we can create clusters of entities that require frequent data exchange and run them in the same process, to minimize data transmission across network.
    - Direct Declaration of Change. This method demands that applications make specific declarations each time one or more attributes are changed. Therefore only changed data need to be transmitted to the network.
    - Double Buffering. Maintaining a copy of all data, a comparison can be made each time a decision has to be made on whether a data needs to be sent. Both data buffers are synchronized at the end of each frame.
- Compression-Based. Compression algorithms could be used to compress data at the sending point, and de-compress the same data at the receiving point. This has the effect of reducing the actual bits and bytes transmitted across network.

Subject to the availability, in future, of an algorithm that is both efficient and effective, a compression-based technique can be applied in addition to other techniques. DSE implemented a combination of the 4th and 5th method of the filtration-based techniques. The choice was made based on elimination. The 1st method is well suited to federate-to-federate rather than process-to-process communication, while method 2 demands substantial processing just to organize the subscription information.

Crash Recovery. The simulation state record/playback module provided by DSE is able to log down simulation states at a frequency specified by the user.

Design for Reusability

Reusability Defined. Reusability defines how easy an application, software module or component may be reused.

Reusable Parts. DSE defined interfaces to which the following application components can be plug-n-played:

- Models. These include behavioral and physical models, and their associated parameters, entities, events and commands.
- Databases. Definition of assets, system/application configurations, scenario (including environment and orbat), etc. are all databases that may be replaced easily, thereby enhancing their prospect of reuse.

Design for Flexibility

Flexibility Defined. Flexibility is a measure of how easy an application can be configured and reconfigured.

Assets Definition Database (AssetDB). Recognizing that the greatest stumbling block faced by the users would be to have to wait for a system to be reconfigured to suit their needs by technical personnel, DSE includes a flexible asset definition mechanism that allows new simulation entities to be created by the end-users themselves.

All DSE objects, except models, can be defined in the asset definition database, which is XML based. These objects are then instantiated at run time by DSE.

The AssetDB provides clarity, as it provides a single file at which simulation objects are defined, instead of having them hard-coded as classes in multiple files within the simulation software. Supporting inheritance, the AssetDB makes objects creation versatile by allowing new objects to take on all parent attributes and have additional ones specific to themselves. In addition, specifying the parent object name in the model code can cause retrieval of all of its inherited objects.

Design for Interoperability

Interoperability Defined. Interoperability refers to how well two or more simulation applications can contribute to a common scenario.

HLA Compliance. DSE supports HLA via a gateway that marshals data between the HLA network and DSE's intra-simulation network. The gateway is designed as a model that can be plugged in to DSE easily. In a similar manner, DSE can support connectivity to any new protocols and standards.

CASE STUDY

Joint Battle System (JBS)

Joint Battle System (JBS) is a distributed simulation system used in the SAF as an experimentation testbed and training platform. It is built upon the JEWEL environment, and thus manifests the latter's development rationale and philosophy. It is designed to be flexible, scalable and highly configurable to serve SAF's varied and demanding experimentation needs.

JBS Architecture

JBS is composed of any number of Player Stations (PLS), an Exercise Control Station (ECS) and an optional Data Analysis Station (DAS). These stations are linked together via HLA.

A PLS consists of four PCs joined together by the DSE provided intra-simulation network. The four PCs serves the role of:

1. HLA Gateway – To communicate and exchange simulation updates among the PLS.

2. C2 – To provide a mock-up of the console of a generic C2 system. These mockup consoles allow the players to exchange C2 messages via a dedicated C2 LAN.

3. Simulation – For executing models of the entities present in the PLS. The models would include platform motion models, sensor models, CGF behavioral models, weapon models, etc.

4. 3D visualization – Each PLS can provide an Out-of-Window view of the simulation environment, to allow the players to immerse in and interact with the simulation.

The PLS can be configured to support different roles, such as CGF controller, platform operator (plane cockpit, ship bridge, vehicle commander's seat), or tactical commander station.

Figure 2 shows the JBS system architecture.

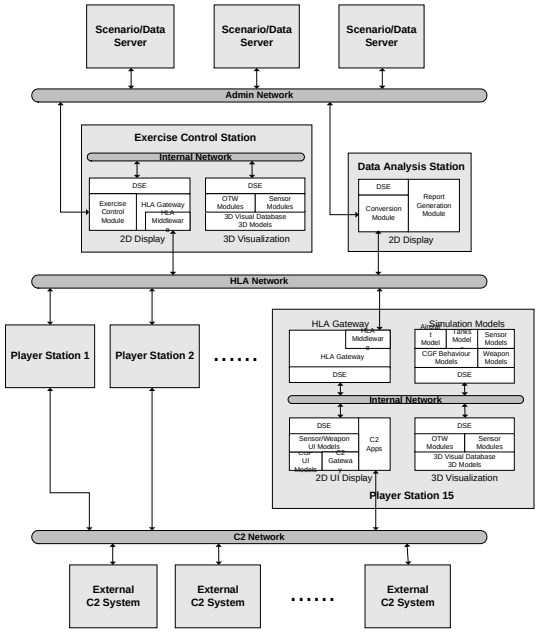

Figure 2. JBS System Architecture

Lessons Learned

In adopting JEWEL as the enterprise M&S framework, and implementing JBS and other systems using JEWEL, some insights were gleaned through the experiences so far.

Model Reuse

JBS has been the base system from which new simulation testbeds were created. These new setups often require enhancements in the form of adding new or modifying existing models. New versions of the models can also be plugged back into JBS. These activities to co-evolve reusable models illuminated one critical limitation of JEWEL – that for models to be plugged-n-played, additional efforts are needed to synchronize the way models make use of the data they exchange with other models.

Our study revealed that apart from disparities caused by differences in simulation framework (issues aptly resolved by adopting the same framework, such as DSE), we need to be mindful of the following factors:

- **Different Application Needs**. Simulations have been used in training, experimentation, analysis and acquisition. Differences in use imposed different requirements on how simulations are designed and implemented. These affect their data I/O requirements, with some requiring (or produce) more data than others. Non-functional requirements such as execution frequency and execution condition add to their differences. All these differences affect the models' packaging and interfaces. Furthermore, there is no standard way to define what models should contain. For example, a model in one application can consist of sensor, decision-making, movement dynamics and weapon system routines of an aircraft, while in others each of the routines constitute a model. The reason for the former is efficiency, since the modules are now able to access functions/data produced by other modules directly, and this is often the adopted design for virtual applications.

- **Different Fidelity, Resolution and Granularity**. Related to the above, simulations for different uses also differ in fidelity, resolution and granularity of their models. For commanders' training using constructive systems, which requires decision making, command and control of huge forces, low to medium granularity models (of aggregated forces) are common place to lighten the processing load. On the other hand, training of operators using virtual systems demand entity level granularity for realism. Virtual systems often require high-resolution models as well to give realistic look and behavior of the simulated entities. Simulations for experimentation require higher fidelity model compared to simulations for training to ensure that the results are credible. Even higher fidelity is necessary for acquisition purposes since assessment of equipment would not be accurate otherwise.

To overcome these challenges, we need to look at ways to capture important properties of models, perhaps to the extent of capturing the algorithmic processes encapsulated within, and harmonize all data required and produced by all interacting models. We are beginning to examine the Conceptual Models of the Mission Space (CMMS), and ideas such as the Levels of Conceptual Interoperability Model (LCIM), Base Object Models (BOMs) and Conceptual Models, and we aspire to align JEWEL along these international initiatives.

AssetDB

The goal of AssetDB is to provide a way for the non-technically trained users to create and configure new object types to be simulated, by mixing and matching existing models. The current implementation, however, can result in system crashes or unrealistic simulation results if the users mixed incompatible models together. For instance, a user could attach a ground radar on an aircraft without knowing that the ground radar model was not designed with the automatic behavior to scan areas below the platform.

There should therefore be a way for model developers to specify the constrains of their models.

Use of HLA & RPR FOM 1.0

HLA was chosen as the communication protocol between the PLS. This design decision was made to create a dual-layer network (i.e. HLA plus DSE's intra-simulation network) for data localization and bandwidth optimization. We also hoped that this implementation would enable integration with other HLA & RPR FOM compliant simulation systems easily, as though it is just another PLS.

While this design allowed for larger scenarios to be enacted, the constant struggle between extending the HLA FOM and living with limited interoperability among simulated entities demanded a re-look into our design philosophy. This is due to these conflicting ideals:

- We should conform to RPR FOM or any internationally recognized FOM, so that integration with similarly compliant systems would be a bliss.

- We should allow simulated entities running in different PLS to exchange data, so that more algorithmic optimization can be performed.

As a result, we decided to forgo the first ideal (assuming that a FOMs mapping gateway would be used when connecting to external systems), and accepted that while RPR FOM is suitable for inter-system data exchanges, it is not suitable for intra-system purposes that requires more "private" data of entities to be published.

Distributed & Delta Logging

For the DSE logging mechanism to function in JBS, every distributed process within JBS must transmit their simulation state over to a selected process that runs the logger. This resulted in an increase in network bandwidth utilization, and the single logger process could not record the complete simulation state at 33Hz frame rate.

To overcome this problem, we modified our logging process to allow distributed logging. Individual processes perform recording of their own state, which will be merged after simulation completes. With this solution, we managed to avoid unnecessary consumption of network bandwidth.

To reduce the amount of data to be recorded, we further enhanced the logger with delta logging capability, which allows snapshot interval to be chosen and only changes in between snapshots are recorded. Resource utilization, in this case, was improved and requirement for large storage media is reduced.

Fault Discovery & Rectification

During development of JBS, the lack of a simple front-end for injecting user commands (for testing purposes) proofed to be painful. Adhoc codes were created, as part of a model, to generate such interactions, which has to be recompiled frequently due to adjustments to test scenarios, however minor they might be.

In addition, the distributed nature of DSE prevents the same simulation state to be recreated, resulting in the difficulty to re-enact the same erroneous state for debugging purposes.

These limitations resulted in long hours of integration testing and debugging. JEWEL should address these limitations.

FUTURE PLANS

We intend to explore MDA-based translation approach to component-level reuse. To overcome the difficulties in testing and debugging of a distributed simulation system, we plan to look into ways to ensure data consistency. We have also begun, and will continue to explore ways to embrace Web-based Technologies and Service-Oriented Architecture (SOA) for M&S.

To better manage our expanding repertoire of reusable M&S components and the increasing number of deployment sites for JEWEL-based simulation, we will setup a JEWEL Knowledge Bank to enable greater collaboration among the developers and promote reuse.

CONCLUSION

In this paper, the authors attempted to provide a complete picture of the rationale and vision of JEWEL. In contrast, only selected development efforts were captured, with the good intention of focusing this paper on the fundamentals – the core component of JEWEL. It is envisaged that additional papers that deal with the other components of JEWEL will be written and shared in the near future.

REFERENCES

Andreas Tolk (2004), Composable Mission Spaces and M&S Repositories Applicability of Open Standards

Andrea Tolk, Michael R. Hieb (2004), Web Enabled M&S and the Global Information Grid

Andreas Tolk, James A. Muguira (2003), The Levels of Conceptual Interoperability Model

Barham, P. T., Barker, R. E., Metzger, J. L. (1999), Dynamic Simulation Environment

Bernard P. Zeigler (1998), DEVS Theory of Quantized Systems

Bernard P. Zeigler, Hessam S. Sarjoughian (2000), Creating Distributed Simulation Using DEVS M&S Environmetns

Don Brutzman, Michael Zyda, J.Mark Pullen, Katherine L. Morse (2002), Extensible Modeling and Simulation Framework (XMSF) Challenes for Web-Based Modeling and Simulation

Gilbert Chen, Boleslaw K. Szymanski (2001), Component-Oriented Simulation Architecture: Toward Interoperability and Interchangeability

Judith S. Dahmann, Richard M. Fujimoto, Richard M. Weatherly (1998), The DoD High Level Architecture – An Update

Katherine L. Morse, Mikel D. Petty, Paul F., Reynolds Jr., William F. Waite, Philomena M. Zimmerman (2004), Findings and Recommendations from 2003 Composable Mission Space Environments Workshop

Paul K.Davis, Robert H.Anderson (2004), Improving the Composability of Department of Defense Models and Simulations

Pope, A. R., Schaffer, R., L. (1991), The SIMNET Network and Protocols

Richard M. Fujimoto (2001), Parallel and Distributed Simulation Systems

Stephen J. Mellor, Marc J. Balcer (2000), Executable UML – A Foundation for Model-Driven Architecture

Steven W. Reichanthal (2002), SRML - Simulation Markup Reference Language

Thomas J. Schriber, Daniel T. Brunner (2001), Inside Discrete Event Simulation Software: How it works and why it matters

Wei Tze Ng, Seng Joo Thio, Cheng Hong Teo (2004), A MDA-based Translation Approach to Component-level Reuse