COMMON MODELING SYSTEM FOR DIGITAL SIMULATION

Capt Rick Painter, USAF
USAF Wright Laboratory/Avionics Directorate
Wright Patterson AFB OH 45433

ABSTRACT

The Joint Modeling and Simulation System is a tri-service investigation into a common modeling framework for the development of digital models. The basis for the success of this framework is a X-window-based, open systems architecture, object-based/oriented methodology, standard interface approach to digital model construction, configuration, execution, and post processing.

For years Department of Defense (DoD) agencies have produced various weapon systems/technologies, and typically digital representations of those. These digital representations (models) have also been developed for other reasons such as studies and analysis, Cost Effectiveness Analysis (COEA) tradeoffs, etc.

Unfortunately, there have been no Modeling and Simulation (M&S) standards, guidelines, or efforts towards commonality in DoD M&S. The typical scenario is an organization hires a contractor to build hardware, and in doing so a digital model may be constructed. Until recently, this model was not even obtained by the organization. Even if it was procured, it was on a unique platform, in a unique language, with unique interfaces, and, with the result being UNIQUE maintenance required. Additionally, the constructors of the model expended MORE effort in writing the "infrastructure" of the model/simulation (e.g. user interface, database/database management system, data journalizing/archiving, graphical presentations, environment characteristics, other components in the simulation, etc.) than in producing the model of the desired system. Other side effects include: duplication of efforts; varying assumptions; lack of credibility/validation, decentralization both in policy AND execution, and various others. J-MASS provides the infrastructure, standards, toolset, and architecture to permit M&S developers and analysts to concentrate on the their area of interest.

J-MASS ARCHITECTURE and STANDARDS LAYERS

J-MASS has several architectural and standardization layers. This paper describes J-MASS in terms of the Tool Interconnect Backplane (IBP) layer, referred to as the Simulation Support Environment (SSE IBP) the Simulation Runtime Agent (SRA) IBP layer, and the Model Component/Object Standards layer.

MODEL COMPONENT/OBJECT STANDARDS

Each model component (or object) in J-MASS is structured compliant with our Software Structural Model (SSM). The SSM evolved from the Software Engineering Institute (SEI) work on the Object Connection Update (OCU) model. Both the C-17 and B-2 weapon systems trainers use a similar methodology for their object definition. The SSM, also described in a document, enforces software structure and interface standards for all levels of object decomposition. In this way, ANY objects in the system can be syntactically "connected" with any other objects in the system with guaranteed success. Semantically, the connection may have no realistic "meaning", but syntactically they can be connected ("Assembled", see discussion in Develop and Assemble Modes under Tool Interconnect Backplane). J-MASS objects are described in three layers: "Players", "Assemblies", and "Elements". Players are the "top" level objects responsible for synchronization with the simulation runtime engine and comply the software interface is standard to all objects at that level. Additionally, the interface between the "player", and its subcomponents, "assemblies" and "elements", is also standard. This interface is similar to but NOT exactly like the player to runtime engine interface. Figure 1 represents the J-MASS SSM implementation.

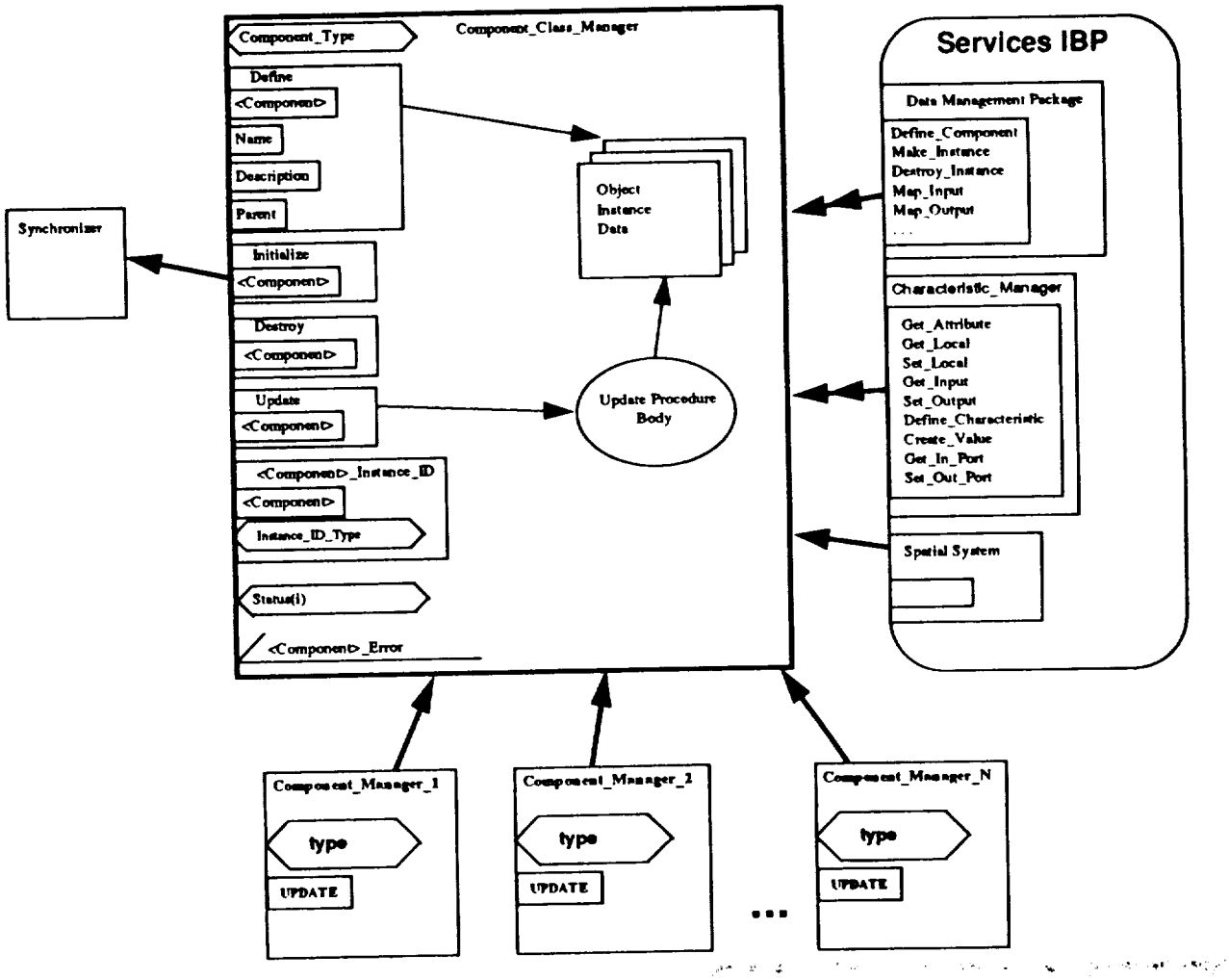


FIGURE 1

OPEN ARCHITECTURE - TOOL INTERCONNECT BACKPLANE

At the Tool Interconnect Backplane (IBP) layer, known in J-MASS as the Simulation Support Environment (SSE) IBP, several backplane methodologies were considered, including the HP Softbench, IEEE P-1175 "Toaster Model", the Atherton Backplane, and, significantly upon the Common Object Request Broker Architecture (CORBA) from the Object Management Group (OMG). In J-MASS terminology, a cul-du-sac model is employed, where each cul-du-sac represents a tool, or potentially a collection of tools or capabilities, referred to as "agents". Each "tool" or "agent" (a software capability), can register as a client/server with the backplane, indicating the service/message traffic of interest. The backplane maintains the knowledge of the other tools that have registered that can either provide the service, or will request the service. This concept is knows as message brokering and is powerful for de-coupling the tools from knowledge of other tools on the system. J-MASS has implemented a prototype of its design for this backplane in C on Unix workstations, currently SUN Sparc series, and Silicon graphics. Other platforms in progress include the IBM RS6000, and HP 9000 series, with DEC Alpha, and VAXstations in the plans. Reference Figure 2 for a graphical depiction of this concept.

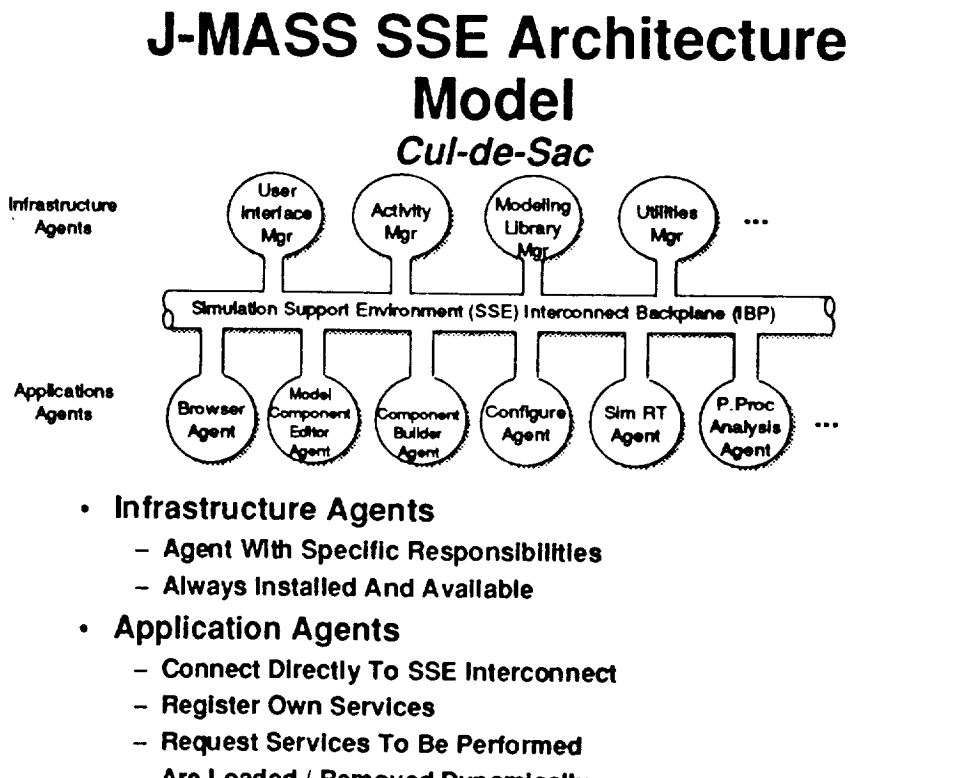


FIGURE 2

User Modes.

J-MASS has five conceptual "user modes" associated with it. These are "functionally" oriented modes, namely: Develop; Assemble; Configure; Execute; and Post-Process. Each represents a capability that a model developer and/or simulation analyst requires to build, configure, execute, and analyze simulations. Each of these modes can be viewed as an instance of the cul-du-sac methodology. The next series of charts (2 thru 6) depict an instance of the backplane at the tool interconnect layer for each of the J-MASS modes.

Develop Mode/Assemble Mode.

Develop Mode and Assemble Mode provide the model developer with visual mechanisms for constructing model objects/object hierarchies, with data flows represented. Control flows (not currently implemented) will also be depicted so that model developers can separate control/activation of objects from data flow. The graphical information is then translated to ascii "dot" notation, referred to as .DSC (description) files. These .DSC files are then read by an automatic code generator, which generates source code compliant with the Software Structural Model (SSM) in various languages (currently Ada, C++). The SSM is discussed further in the Model Component/Object Standards section. At this point, the algorithms for the lowest level objects in the decomposition must still be described (currently, in the native language thru an editor). The code can then be compiled, linked, loaded and executed. A semantic tool, or "template" editor, is provided to build the semantic "template" information that describes "normal" assembly of the model components, which is done in "Assemble" Mode. Here in develop, the template semantics are generated. See Figures 3 and 4 for a graphical depiction of Develop Mode. Assemble mode permits the connection of the model objects built in Develop Mode visually. The "templates" are populated with actual object instance selections. All of the model components are stored in the modeling library, an object oriented storage mechanism which makes the information about the objects in J-MASS available to all other agents on the backplane.

Application Agents Supporting Develop

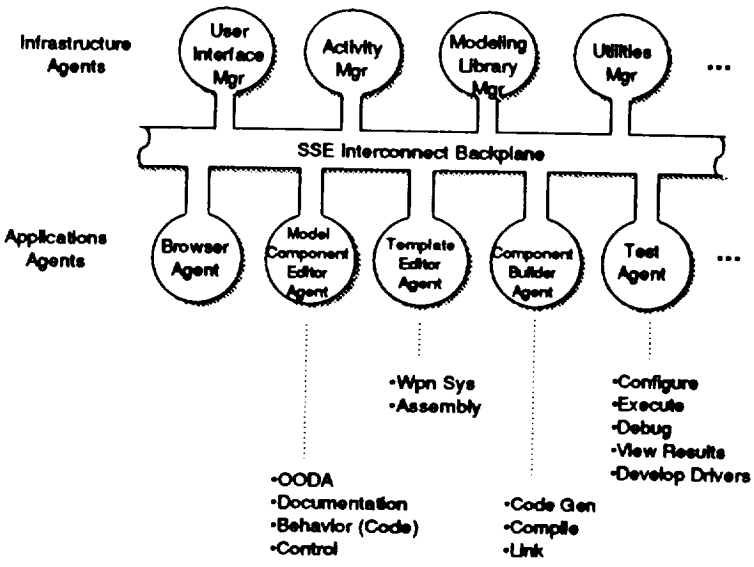


FIGURE 3

Application Agents Supporting Assemble

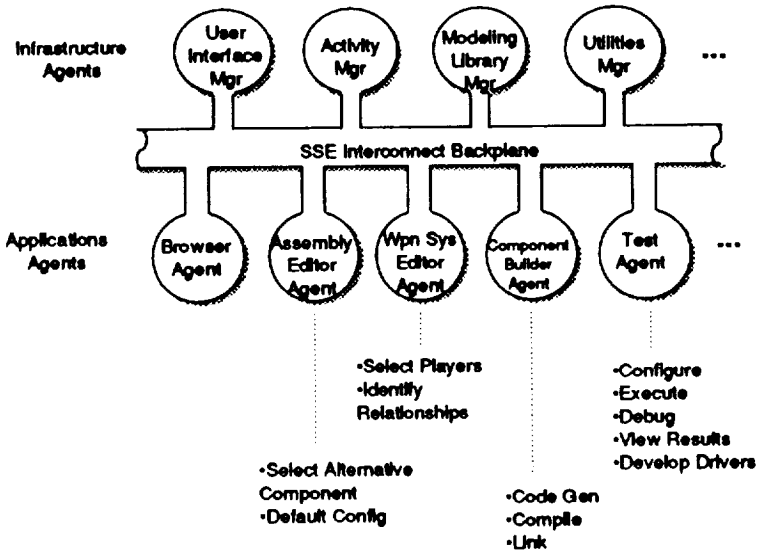


FIGURE 4

Configure Mode.

Configure Mode permits the M&S developer/analyst with the capability to determine simulation characteristics. Model component objects attribute values are populated with values thru a graphical configure tool. Additionally, geographical laydowns, raster maps, etc. are made available to set up the scenarios of the model objects stored in the model library. J-MASS "teams" are formed, whereby player classes are defined, and actual player instances are populated for the teams. This "distribution strategy" is totally configurable by the user. If "legacy" simulations exist, the configure mode will permit the modeler/analyst with the capability to catalogue those models/simulations, and have data passed back in forth sequentially. Eventually, real time synchronized communication between J-MASS compliant and legacy simulations will be achieved. Additionally, if a Distributive Interactive Simulation (DIS) Protocol Data Unit (PDU) generation is desired, the user is able to configure a J-MASS team (collection of players into a single executable). The entire team will then generate PDUs, and the J-MASS spatial system will create "objects" for the incoming DIS entities. The software that provides this capability is the DIS_manager software, and is de-coupled from the standard J-MASS objects, so as not to perturb that interface. Figure 5 depicts the architecture backplane instance for Configure Mode.

Application Agents Supporting Configure

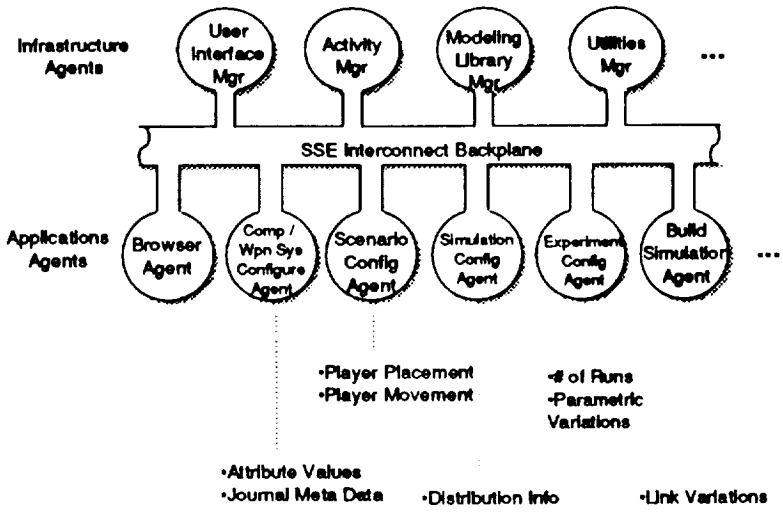


FIGURE 5

Execute Mode.

Execute Mode simply executes the selected simulation. Currently, visualization is accomplished in the Post Process Mode. If the DIS manager software was invoked due to configuration selection, then using "magic carpet" software, the PDUs can be displayed in real time. In work is a real time display of the simulation as it occurs. Figure 6 depicts the architecture instance for the Execute Mode.

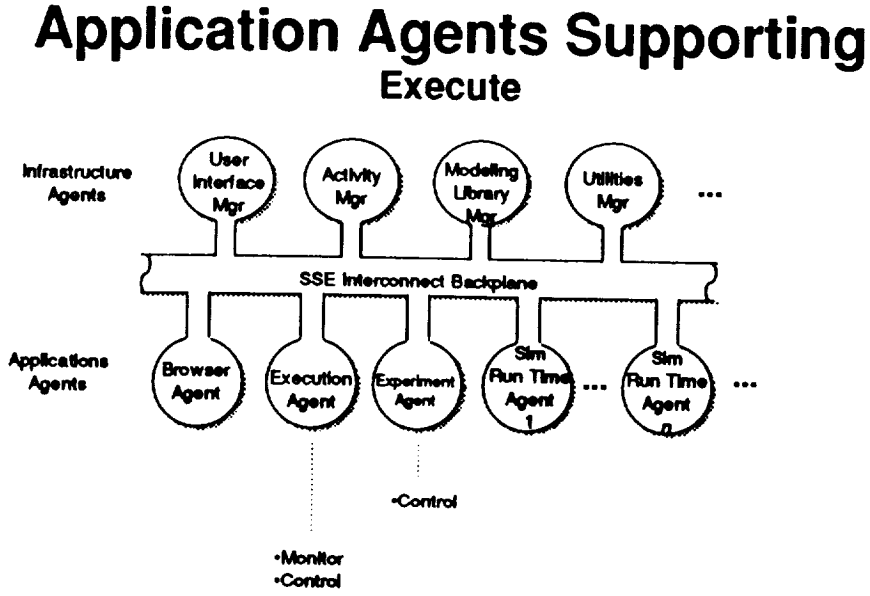


FIGURE 6

Post Process Mode

Post Process Mode is a visualization, both static and dynamic, of the information of interest to the user. This mode includes graphical plotting tools, and animated playback capability. The extraction tool converts the binary journalized data into ascii information. The filter mechanism then prepares it in the appropriate format for the display tool requested. Figure 7 describes the backplane instance for the post process mode of J-MASS.

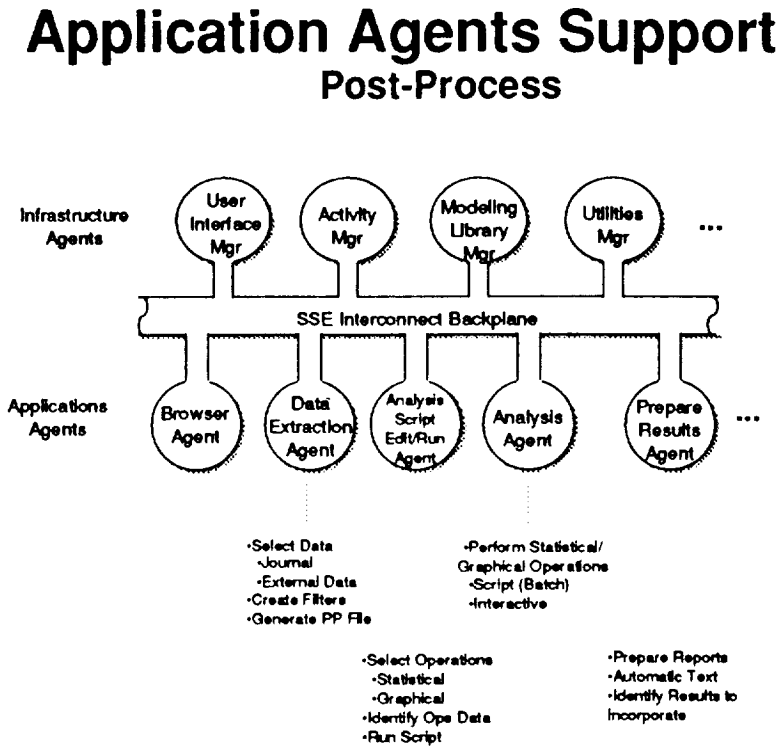


FIGURE 7

SIMULATION RUNTIME AGENT (SRA) ARCHITECTURE

The J-MASS Simulation Runtime Agent (SRA) architecture is depicted in Figure 6. The SRA is "expanded" in this view to show its own architecture. In fact, any agent on the SSE backplane may in fact be another recursive instance of the SSE level. Notice the SRA has its own backplane. The SSE level and SRA level backplanes could in fact be the same. In our current implementation, they are not, but both are distributed in nature using standard Unix (TCP/IP) socket message passing mechanisms. What is important to note in the SRA is the encapsulation of the spatial object, synchronization object, data management object, journalization object, and others away from the model objects. Thus, a true "plug and play" architecture is achieved because any given object may be replaced in the architecture without perturbing the other objects. In the SRA, each team is a single executable using a shared memory implementation, providing significantly faster communication than "inter-team" communication, which uses Unix sockets. Just as the SSE level architecture is distributable, so too are the "teams" within any given SRA. A J-MASS system may in fact have more than one SRA, each communicating over the SSE level backplane. In fact, we plan to demonstrate an Ada SRA with Ada model objects communicating with C++ SRA and C++ model objects over the SSE IBP mechanism. "Players" communicate with each other by placing information on each others "ports" facilities. Players do NOT require apriori knowledge of what team the other player is on, the team synchronizers work with the SRA synchronizer to "locate" the appropriate port. Again, the model objects remain "un-perturbed" with this approach. Journalization of output is accomplished by the journalization object, using state information maintained in the Data Management Package (DMP). In this way, non-intrusive journalizaing occurs. Figure 8 represents the expanded view of the SRA.

J-MASS Architecture
Simulation Runtime Agent
(SRA) Detail

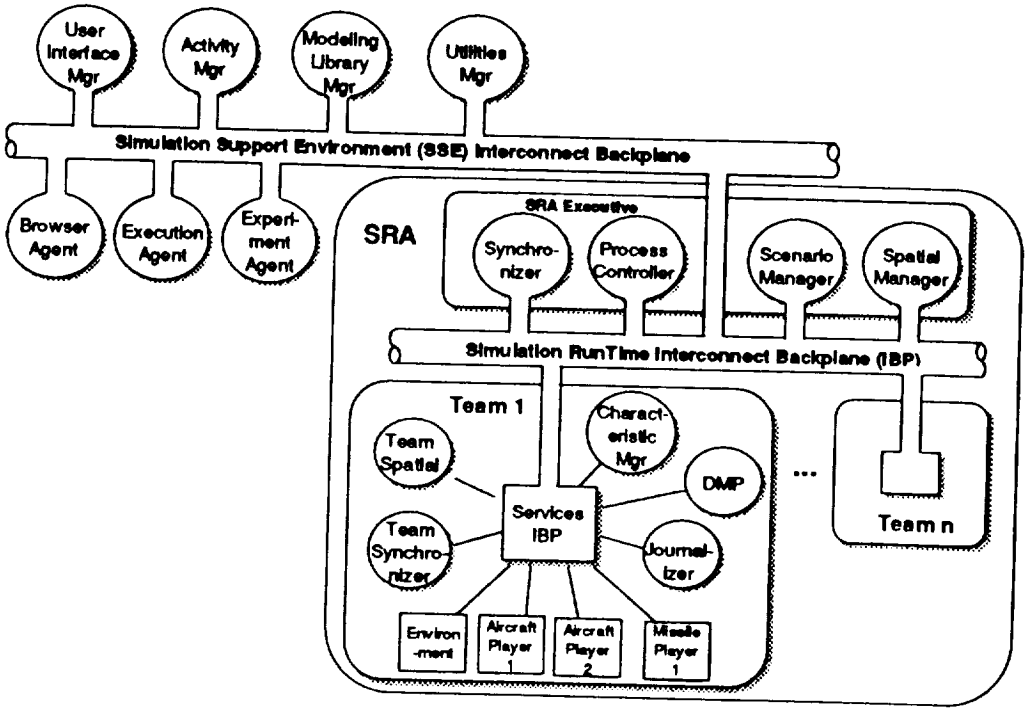


FIGURE 8

COMMERCIAL POTENTIAL

The J-MASS concepts and philosophies are not entirely original. The backplane methodology, message brokering mechanisms have been espoused by OMG and others. However, J-MASS has applied these concepts to a generalized Modeling and Simulation System.

J-MASS brings the idea of standards for digital simulations, both in structure and interface. This guarantees "plug and play" philosophies, both from model components and architecture components point of view. J-MASS espouses the idea of "plug and play" throughout the architecture to include tools, objects (model components), etc.

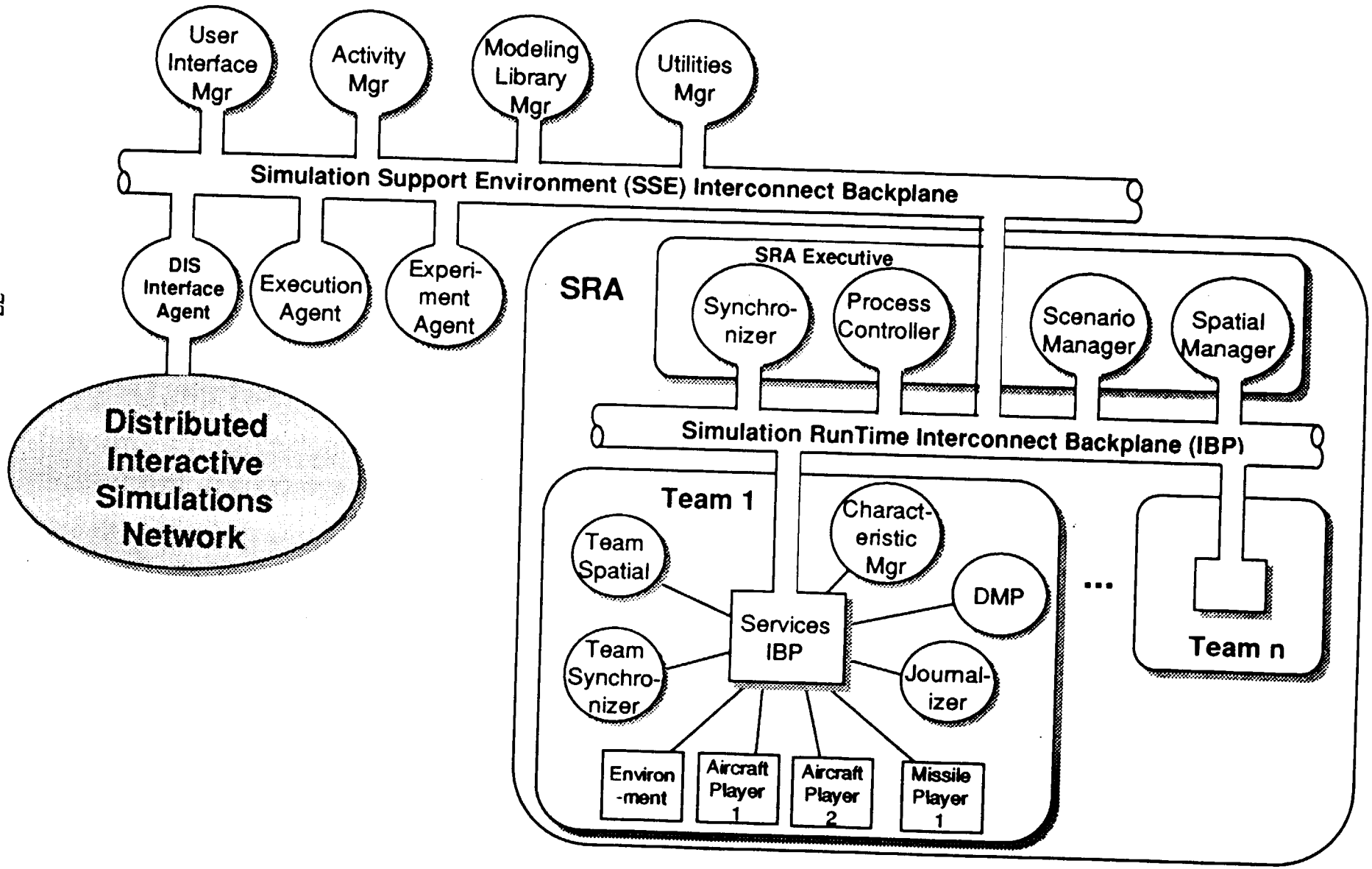
The J-MASS notion of graphical tool environment coincides with standard commercial technology as well. Expanding that concept which permits (automated) standard compliance with specified standard structures is another potential benefit to the commercial world.

J-MASS itself does NOT prescribe what objects or systems are modelled with its architecture. For example, the object repositories could represent traffic objects, manufacturing objects, weather objects, organizational objects, utility objects, etc. The system is designed so that the M&S communities build object hierarchies and behavior appropriate for the particular domain.

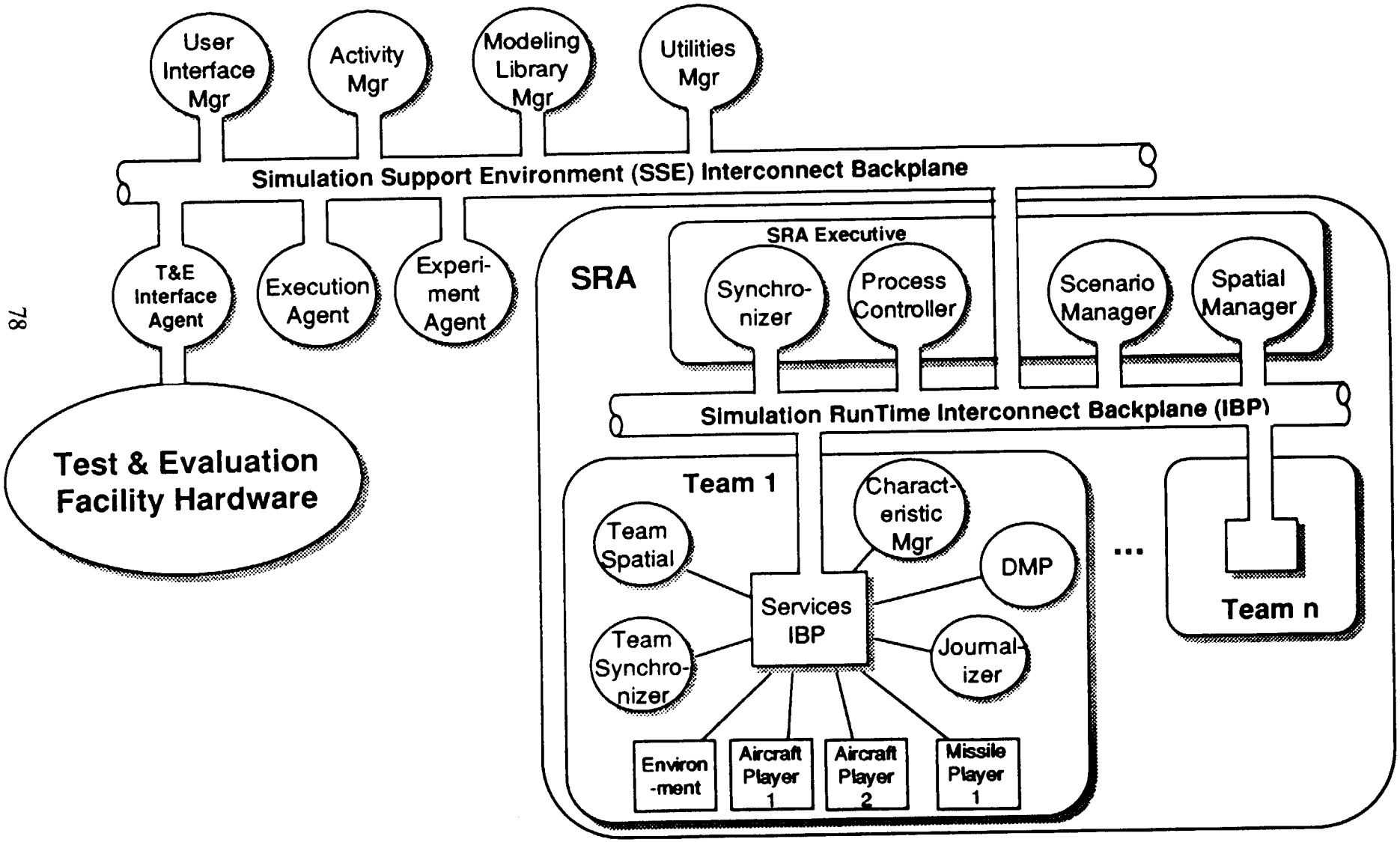
J-MASS / Windows Comparison

J-MASS	Windows
J-MASS Development Team	**Microsoft Corp.**
J-MASS Software	**Windows Operating System**
Tool Interconnect Standard	**Windows Interface Standard**
User Interface Mgr, Activity Mgr, Modeling Library Mgr, Utilities Mgr	Program Mgr, File Mgr, Print Mgr, etc.
Browse Agent, Test Agent, Simulation Runtime Agent (SRA)	Write, Paintbrush, Terminal, etc.
Specialized Post Processing Tools, Unique SRA, etc.	Word for Windows, Excel, PowerPoint, etc.
Model Interconnect Standard	**Word File Format**
J-MASS Model Component	Word Document

J-MASS Architecture
Extendible to Training



J-MASS Architecture Extendible to Weapon Systems Acquisition



J-MASS Architecture
Extendible to Wargaming