Proceedings of the 1997 Winter Simulation Conference
ed. S. Andradóttir, K. J. Healy, D. H. Withers, and B. L. Nelson

A FRAMEWORK FOR DEVELOPING AND MANAGING OBJECTS IN A COMPLEX SIMULATION SYSTEM

James D. Barrett

NYMA, Inc.
Engineering Services Division
4027 Colonel Glenn Hwy., Suite 445
Dayton, OH 45431-1672, U.S.A.
e-mail: j-d-barrett@worldnet.att.net

ABSTRACT

The Integrated Supportability Analysis and Cost System (ISACS+) Operation and Support (O&S) Simulator is a flexible, object-oriented software tool that simulates the operation and support environment for a fleet of aircraft and engines. The model is written in C++ and runs from pre-processed flat files generated from any SQL database, making it both portable and database-independent.

Originally developed for General Electric Aircraft Engines (GEAE), ISACS+ is currently being rehosted and extended under a Space Act Agreement between GEAE and NASA Lewis Research Center.

This paper describes a process that begins with objects of interest to a simulation model and imposes order on those objects by defining interactions between the objects and services required to support those interactions. The result of this process is an object framework. An object framework can itself become an object within a greater object framework. For example, a simulation module is an object framework imposing order on the objects it controls. This module can, in turn, be treated as an object in a larger application that provides not just simulation capability, but other capabilities as well (e.g., reporting, database management). Based on the ISACS+ development team's experience, object frameworks provide a useful means of managing complexity in a large simulation model.

1 INTRODUCTION

The Integrated Supportability Analysis and Cost System (ISACS+) Operation and Support (O&S) Simulator was developed for General Electric Aircraft Engines (GEAE) to perform operation and support analysis for the U. S. Air Force F-16 engine program. The object-oriented model is written in C++ and runs from pre-processed flat files generated from any SQL database engine, making it both portable and database-independent.

The original implementation, delivered early in 1994, consists of four subsystems: the Graphical User Interface (GUI), the O&S Simulation Module, the Report Generator and the Parameter Database. The original system is hosted on a UNIX workstation, using Ingres as the RDBMS, X Window/Motif and Ingres Windows/4GL for GUI generation, and native C++ for implementation of the O&S Simulation Module. The subsystems which comprise the ISACS+ functional architecture are shown in Figure 1.

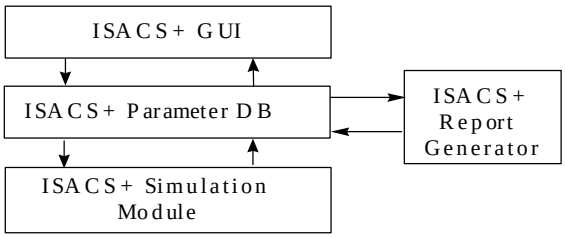

Figure 1 - ISACS+ Functional Architecture

In 1996, GEAE and NASA Lewis Research Center entered into a Space Act Agreement to extend the capabilities of ISACS+ to address the needs of the commercial airline fleet. This effort began by porting the ISACS+ Simulation Module and the Parameter Database table definitions from UNIX to run in a 32-bit Windows client/server environment, using Borland's C++5.0 Development Suite as the compiler and Oracle 7.2 Workgroup Server as the SQL engine.

This paper details the process that transformed the functional architecture of Figure 1 into an object-oriented application framework. The process began by examining and documenting the ISACS+ object model (OM) and placing the OM into an object framework by

describing interactions between objects and services required to support those interactions. Once the O&S Simulator module was defined in this way, it became an object itself. This object is then placed into the ISACS+ application framework which, in turn, defines how the O&S Simulator interacts with the other subsystems (i.e., objects) contained within ISACS+.

The development of the ISACS+ OM and object framework is based on sound practices of object-oriented analysis, object-oriented design, and object-oriented programming which the development team has used successfully many times in the past.

2 OBJECTS AS MODELS OF REALITY

2.1 Objects and Object Models Defined

As Rumbaugh et al. (1991) note, simulation models lend themselves well to object-oriented methodology because both involve discrete updates to individual items, rather than a massive update to the system as a whole. This means it is easy to define objects to be modeled (classes) and to ascribe attributes (member data or class data) and behaviors (member functions or object methods) of interest to those objects.

The object to be modeled, then, consists of data of interest, operations on that data, and messages that allow others to request services or information from that object (Pressman, 1992). If correctly designed, it is possible to extend the functionality of an object over time without adversely impacting existing applications by either simply adding new member functions or by adding both new data and new member functions to an existing object. This information about objects, when documented, constitutes the OM.

It is worth noting, however, that objects in the OM are not allowed to simply proliferate unattended, passing "messages" about freely and without regard for the context implied by the state of simulation model. Object frameworks, then, prescribe the manner in which the objects in a simulation model interact with one another and impose order on the OM.

2.2 The ISACS+ Object Model

The porting process allowed the development team to evaluate and document the ISACS+ OM in detail, resulting in a list of class-&-object (Coad and Yourdon, 1991) inputs that are of interest to the ISACS+ O&S Simulation Module. It turns out that the inputs can be categorized into three general areas of interest:

1) Basing Structure,
2) Hardware, and
3) Fielded System Configurations.

Tables 1-3 identify these class-&-object inputs categorized by area of interest. Indentation in the tables documents relationships among objects.

Table 1 - Basing Concept Inputs to the O&S Simulator

Classes & Objects
Basing Concepts:
Levels of Maintenance
Facilities
Facility Locations
Facility Changes
Facility Statistics
Locations
Transportation Times
Squadrons
Squadron Facilities
Sortie Schedule
Squadron Changes
Missions
Mission Factors
Aircraft Delivery Schedule
Aircraft Movement Schedule
Engine Retirement Schedule
Spares Delivery Schedule
Spares Movement Schedule

Table 2 - Hardware Inputs to the O&S Simulator

Classes & Objects
Hardware:
Part Data
Durability Failure Modes(Weibull Parameters etc.)
Service Limit Factors
Probability of Detection
Secondary Damage
Reliability Failure Modes
Probability of Detection
Secondary Damage
Fleet Time
Scheduled Removal Requirements
Windows for Scheduled Removals
Time to Removals
System Level Failure Modes (Performance Degradation)
Durability Failure Modes for Performance Degradation
Time Compliance Technical Orders
Retrofits
Replacement Part Configurations
Pre/Post/Pilot Squawk Inspections
Inspection Requirements
Scheduled Flight Line Inspections
Inspection Requirements
Task Data
Indentured Parts
Inspection Requirements
Configuration Templates
Exposed Parts
Aircraft Templates
Warranty Requirements

Table 3 - Fielded Configuration Inputs to the O&S Simulator

Classes & Objects
Fielded Systems:
Aircraft Identifiers
Aircraft Configurations
Installed Engine Configurations
Initial Time on Parts
Initial Time on Failure Modes
Time to Next Scheduled Removal
Time to Next Scheduled Inspection
Engines Initialized in the Shop
In-Shop Configurations
Initial Time on Parts
Initial Time on Failure Modes
Time to Next Scheduled Removal
Time to Next Scheduled Inspection
Spare Hardware
Spare Hardware Configuration
Initial Time on Parts
Initial Time on Failure Modes
Time to Next Scheduled Removal
Time to Next Scheduled Inspection

To develop a useable OM for ISACS+, it was not enough to simply add attributes and member functions to the objects in Tables 1-3. It was necessary to describe how the objects interact with one another in the model. For example, parts have failure modes which accumulate time as aircraft assigned to squadrons fly sorties. The completed OM for the ISACS+ problem domain, then, must identify

1) application program interface (API) signatures through which objects communicate, such as how to obtain an aircraft to fly a given sortie, and
2) the needs of the O&S Simulator that must be satisfied by other subsystems of ISACS+, such as the requirement that a process separate from the O&S Simulator download the Parameter Database into the input flat file format.

3 FRAMEWORKS: CONTROL OF OBJECT INTERACTION

3.1 Building An Application With Objects

While it may seem that an object-oriented application is nothing more than objects and their message-passing interfaces (Lewis, et al., 1995), it is important to note that the actual design of an application (rather than the implementation of the individual objects) depends on defining a problem domain and describing how the objects within that domain interact with one another. In fact, ISACS+, or any simulation model, only has validity in the context of the problem domain within which it is designed to operate. This being so, it is important that an organizational framework be defined which provides consistency and stability for the model over time (Coad and Yourdon, 1991).

3.1 Object Frameworks

The organizational framework, in this context, contains the knowledge required to build an application out of individual objects. Such knowledge includes not only the interactions between objects within the simulation model, but the interactions between the simulation model and the runtime environment as well. Once this knowledge is catalogued, it is possible to develop a model of object interaction that is uniquely representative of the problem domain under study, and is also generic enough to serve as a reusable application object. Such a model is called the object-oriented framework, or simply the object framework. A well-defined framework is more, then, than objects and message passing, it is a miniature application (Lewis et al., 1995) specialized toward a narrow range of problems.

4 BUILDING THE ISACS+ OBJECT FRAMEWORK

4.1 The ISACS+ Problem Domain

The ISACS+ OM defines a problem domain in which aircraft may be assigned anywhere in the world and fly any route to any legitimate end-point, accumulating time on parts and increasing the likelihood of a failure (random or durability). Once a failure event occurs, the offending component (piece-part, line-replaceable unit (LRU), assembly, or end item) is removed and repair tasks and other maintenance events (including inspections which might reveal other failures) are initiated.

Once repaired, a part is either returned to the assembly from which it was removed or placed into the spares pool. By serializing and tracking parts, it is possible to watch the distribution of parts within the maintenance environment, giving a measure of the "mixing" of fleet reparables over time (in addition to the capturing of the primary event data and related repair, packaging, and transportation events.) Figure 2 provides a simplified object model view of location objects in a typical ISACS+ model.

4.2 Controlling the Proliferation of Objects

While Figure 2 may appear at first glance to be simply a class hierarchy, it is actually an interconnection of smaller object models that represent, in effect, a subsystem (Islam, 1996) of the O&S Simulator. For instance, each location being simulated may have zero or more repair facilities, spares pools (warehousing), or squadrons located there. In turn, each squadron may

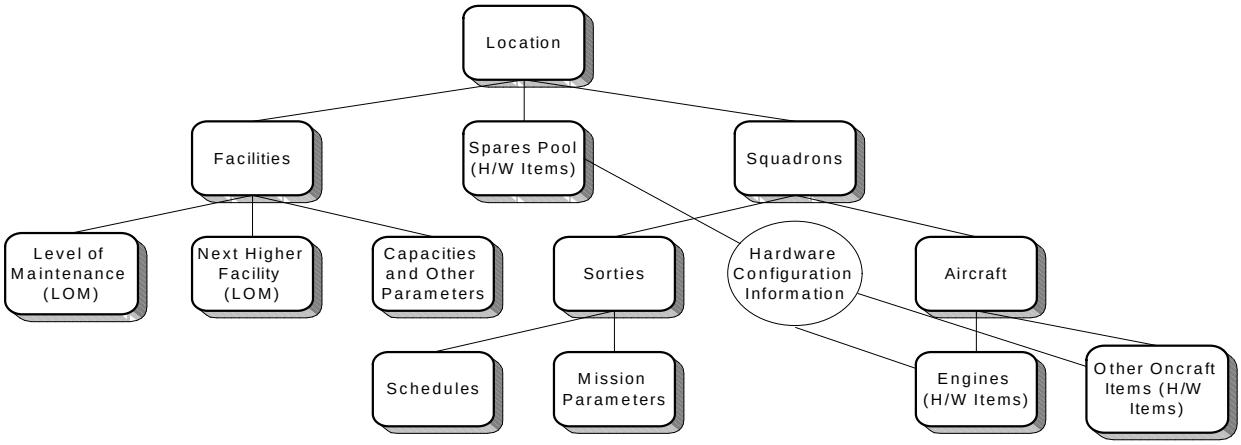

Figure 2 - Simplified Object Model of ISACS+ Location Objects

have zero or more aircraft assigned with which to fly zero or more sorties, etc. (Note the tree-like pattern of relationships. That pattern is exploited for its very efficient memory storage and retrieval capability - see Weiss, 1993 for a discussion of memory models and computational complexity.) In a typical ISACS+ run, there are literally tens of thousands of such objects communicating simultaneously at any instant in time. The ability to create such large numbers of objects is draining on computing resources and must be managed and used effectively.

4.3 The ISACS+ Object Framework

To manage the objects and the connections between objects requires that rules of interaction and documented application program interfaces (APIs) be defined to allow objects to determine the "reasonableness" of messages received from or sent to other simulation objects. The process for defining such rules is akin to the process of top-down software design, in that a procedural interaction sequence is established. Sequencing is not only allowable under the rules of object-orientation, it is a natural reflection of the world being modeled. The sequence of events within ISACS+ is controlled by the passage of time in user-defined units and levels of granularity. This sequence imposes order on the ISACS+ OM and defines the ISACS+ O&S Simulator Object Framework, shown in Figure 3. In this context, the object framework equates to a software module.

Figure 3 exhibits the "procedural" logic that is used to establish an object framework for the ISACS+ O&S Simulator. Each time the O&S Simulator is run, it performs data initialization wherein comma-delimited flat files produced from the ISACS+ Parameter Database are loaded into memory to create a map of the simulation model. Once that map is created, aircraft are flown, time is accumulated, failures detected, work performed, and spares placed back into service, all while keeping detailed records of every occurrence in the simulation.

5 THE NEED FOR AN APPLICATION FRAMEWORK

5.1 Using Knowledge About the Application

The knowledge that the O&S Simulator expects another process to create flat files for upload into its memory map is part of the object framework which defines the rules that apply to the O&S Simulator OM. The coordination required to generate the flat files and then

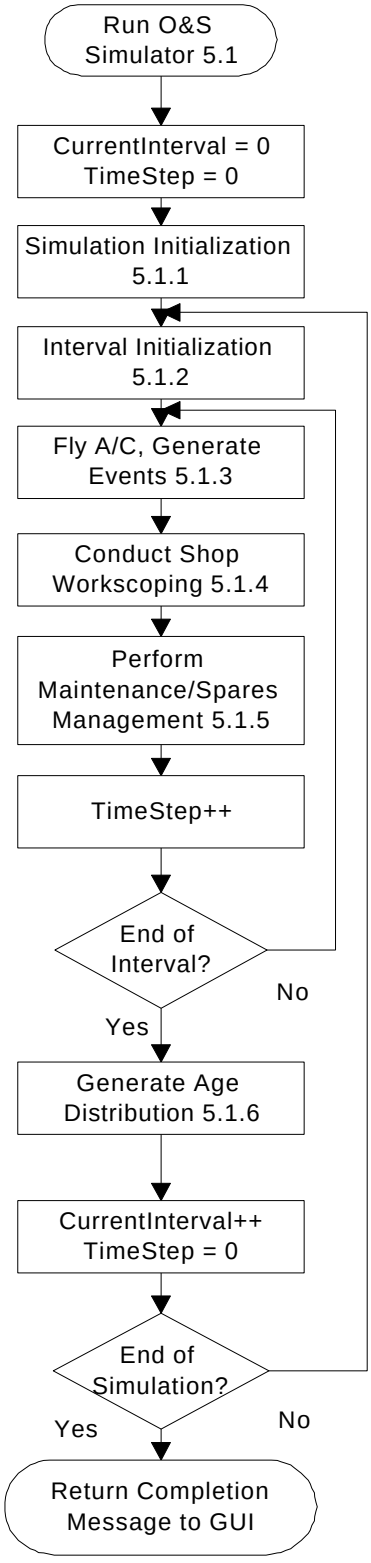

Figure 3 - ISACS+ O&S Simulator Object Framework

run the O&S Simulator represents knowledge that must be stored in a higher-level application object framework, or simply application framework. In this case, the object frameworks that represent the O&S Simulator, the Report Generator, the Parameter Database, and the graphical user interface become objects themselves.

The knowledge contained in the ISACS+ object framework had to be complete enough and robust enough to enable:

1) redesign of the Parameter Database to make best use of newer database technologies such as triggers and stored procedures;
2) redesign and rewrite of the Graphical User Interface (which was not portable from the original Ingres environment); and
3) design of a Report Generator Database, data preprocessor, and graphical user interface (also not portable from the original Ingres environment).

5.2 Applications as Objects

In order to develop an application, it is necessary to first define an application framework composed of lower-level object frameworks that interact (Islam, 1996). In particular, if each subsystem, such as the ISACS+ O&S Simulator, the Report Generator Process, or the Parameter Database GUI is treated as an object, an application framework can be established that defines

1) the objects of interest within the framework;
2) the responsibilities of each object in the framework; and
3) the interactions between objects in the framework.

The ISACS+ Application Framework Model exhibited in Figure 4 represents the object frameworks that compose ISACS+.

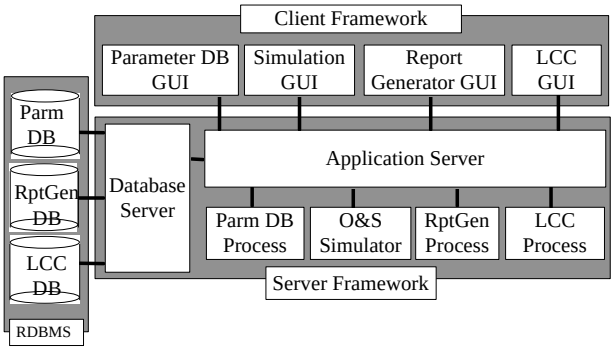

Figure 4 - The ISACS+ Application Framework Model

5.3 Connecting Application Objects in a Framework

Each of the object frameworks in Figure 4 represents a complete application in and of itself. Each works with a known input to generate a known output. However, the output of one object framework (e.g., Parameter DB process) may serve as input to another object framework (e.g., O&S Simulator). In this context, it is easy to see how each object framework serves as an object that fits within a larger application framework. This larger application framework defines the relationships between individual object frameworks. These relationships can be captured and customized to define an application suite, as has been done in the case of ISACS+.

6 CONCLUSIONS

This paper has demonstrated a recursive approach to object modeling whereby objects are defined and rules are applied to how those objects interact with one another. This approach yields an object framework. Object frameworks may, in turn, be treated as simple objects that interact with other object frameworks in an application framework. In this manner, it is possible to start with a very simple object and build up a very complex application. This approach was applied successfully to ISACS+.

REFERENCES

Coad, Peter and Edward Yourdon. 1991. *Object-Oriented Design.* Prentice-Hall, Englewood Cliffs, NJ.

Islam, Nayeem. 1996. *Distributed Objects: Methodologies for Customizing Systems Software.* IEEE Computer Society Press, Inc., Los Alamitos, CA.

Lewis, Ted, Glenn Andert, Paul Calder, Erich Gamma, Wolfgang Pree, Larry Rosenstein, Kurt Schmucker, John Vlissides, Andre Weinand. 1995. *Object Oriented Application Frameworks.* Manning Publications Co., Greenwich, CT.

Pressman, Roger S. 1992. *Software Engineering: A Practitioner's Approach.* McGraw-Hill, Inc., New York, NY.

Rumbaugh, James, Michael Blaha, Frederick Eddy, William Lorensen, William Premerlani. 1991. *Object-Oriented Modeling and Design.* Prentice-Hall, Inc., Englewood Cliffs, NJ.

Weiss, Mark Allen. 1993. *Data Structures and Algorithm Analysis in C.* Benjamin/Cummings Publishing Company, Inc., Redwood City, CA.

AUTHOR BIOGRAPHY

JAMES D. BARRETT is an engineering specialist with NYMA, Inc., in Dayton, Ohio. He is responsible for selecting and installing the ISACS+ development environment and for development of the Simulation GUI, the Parameter Database GUI, the Main Menu GUI, and all C++ development. In addition to ISACS+, he has developed simulation models for factory and laboratory facilities planning. He is currently a member of the IEEE, IEEE Computer Society, the Association for Computing Machinery, and the Society for Computer Simulation.