Proceedings of the 1995 Winter Simulation Conference
ed. C. Alexopoulos, K. Kang, W. R. Lilegdon, and D. Goldsman

OBJECT-ORIENTED MILITARY SIMULATION DEVELOPMENT AND APPLICATION

Ronald D. Painter

CACI Products Company
1600 Wilson Boulevard
Suite 1300
Arlington, Virginia 22209, U.S.A

ABSTRACT

The Department of Defense (DoD) is currently embracing object-oriented programming as a possible solution to problems that beset current simulation tools. The object paradigm, while extremely powerful, has associated costs that must be considered. To realize object-oriented programming benefits such as object reuse (sharing objects between simulations), data hiding (encapsulation), and code reuse and extension (polymorphism), the DoD modeling and simulation community must build objects to a set design standard. The immediate need facing the military simulation community is to agree on and build a framework for object-oriented simulations.

1 INTRODUCTION

Object-orientation is the current buzz word in the DoD simulation community. Virtually all new simulations built today, or planned in the near future, use object technology. The real question facing the simulation community is whether object-based simulations will display beneficial characteristics when compared to simulations built using a procedural approach. This paper examines the reasons why the DoD simulation community has embraced the object paradigm, and then presents issues and requirements that must be met in order to successfully implement object-oriented methods into military simulations.

2 WHAT IS OBJECT-ORIENTED?

One problem with a term such as object-oriented programming is that it means different things to different people. Outside the computer domain, an "object" is something perceptible with one of the five senses. Inside the computer domain, object-oriented can mean an architectural design or a set of computer language constructs. As an architecture, object-oriented is a design composed of objects that each system or subsystem manipulates. An object-oriented architecture then leads to the concept of decomposing a system into separate discernible objects. In computer language constructs, programmers decompose objects into classes of data structures. Developing object-oriented programs involves encapsulating abstract data types into, for lack of a better word, base objects. Encapsulating data into base objects almost certainly includes information hiding so that abstract data types can only be acted on (used or changed) in explicit ways. To achieve a reasonable level of data hiding, the procedures for acting on the abstract data types must be bundled with the data.

To build an object-oriented system from base objects, programmers must be able to build components (other objects) that can use a number of base objects yet maintain the properties possessed by the original objects. This idea is called inheritance. Two types of inheritance are multiple and polymorphic. Multiple inheritance allows an object to inherit properties from two different objects that, in turn, inherited their properties from a common ancestor. Polymorphic inheritance is the ability to enhance or modify the functional interaction of a higher level object, using the original abstract data types in the base objects as a foundation.

Smalltalk, LISP, and Prologue were the first computer languages with object characteristics. Although these languages have been around many years, few large systems have been built using them. Their unavailability in a uniform implementation across multiple platforms limited their use. Smalltalk, LISP, and Prologue, also suffered from a lack of readable code and explicit data type checking which made code written by one developer difficult to correct or understand by another developer. Although highly abstract, these languages were capable of expressing complex concepts with a few lines of code (which led to the problem stated above) and easy to reuse. ADA was the first computer language in which the DoD attempted to incorporate object-oriented features into a language that would see widespread use. While ADA has full data encapsulation and data hiding, it lacks the polymorphic and multiple inheritance characteristics necessary for object-oriented design. ADAs type data checking and readability does, however, allow reuse of its packages between simulations.

One of the most difficult issues in building objects in an object-oriented paradigm is selecting base objects on which everything is built; that is defining the taxonomy. The classical approach to programming, which has led to most of the useable programs today, uses top down design. Yet object technology, because it builds up from base objects to more complex objects, requires a bottom up design approach. We have learned, however, from our experience in procedural languages, that bottom up designs often lead to unworkable code. Designing an object-oriented simulation, therefore, implies carefully combining key features of the top down and bottom up design approaches.

Top down and bottom up design concepts are only partially useful to object-oriented programming. An effective object-oriented programming design philosophy must first design the objects with which the system interacts and not design the system to interact with objects. A word of caution however, as this means that the total system function is deferred until the last stages of object building.

3 DOD SIMULATION REQUIREMENTS

In the past, each service constructed and used unique simulations to analyze their own weapon systems. Projects would often build a complete warfare level simulation to analyze their own requirements and system performance. Military down-sizing has significantly reduced the number of simulation analysts available to each service, making inter-service simulation efforts imperative.

At the moment, program offices are expected to "contribute" a simulation of their weapon system to an approved warfare model for analysis. Building large-scale warfare models is becoming a joint service project with the United States Army contributing the ground war, the United States Air Force contributing the air war, and the United States Navy contributing the naval war. In addition, the constant change of tactics and weapon systems demands a flexible, easily changed simulation system. The reality of these new and challenging requirements has led the DoD to embrace the object-oriented paradigm.

The fundamental requirements to which future simulation objects must adhere to meet DoD needs include:

- Verified, Validated and Accredited (VV&A)
- Reusable in the simulation
- Useable across simulations

Currently, many DoD simulations have undergone verification (i.e., does it work as it was designed), few have undergone validation (i.e., does it work like the real "system") and almost none have undergone accreditation. The demand for object reuse makes accreditation much more important. Objects accredited by the responsible service and program office will eliminate political debates regarding the validity of a systems representation when compared to another system.

Encapsulating code in ADA (packages) or objects written in C++ or MODSIM III code is a first step towards meeting VV&A requirements. The real advantage of full object-oriented languages such as C++ and MODSIM III, over ADA for instance, is their inheritance mechanisms. These mechanisms allow a programmer to modify or enhance a system's properties without altering the original system properties, and at the same time, provide a mechanism for tracing changes in the code.

The DoD also requires that a simulation operate at a number of levels of fidelity (albeit with the commiserate cost in computational time and program size). This means a single simulation fills constructive analysis, virtual analysis, and live exercise support roles.

4 ISSUES FACING THE DOD SIMULATION COMMUNITY

Creating computer simulations that interact and interplay with each other presents a formidable challenge that only the object paradigm can solve. Again, the real issue before the simulation community is selecting the fundamental data abstract on which "base" objects act. Each level of object interaction between simulations is different but interrelated. The levels of interaction are:

- Objects used (as is) in another simulation
- Objects used by other objects
- Objects fully inherited and extended by others objects

A fundamental difference exists between used objects and inherited objects. Inherited objects, where methods can be overridden and new methods incorporated, require a computer language implementation match between objects. With careful planning the implementation match can be limited to the data constructs and method invocation, but LISP and C++ objects for example, must be hand-crafted to work well together.

Using a collection of objects from various simulations "as is" seems, at first glance, to be a more tractable solution. A number of object reuse standards such as the Object Management Group's Common Object Request Broker Architecture (CORBA), the Open Software Foundation's Distributed Computing Environment (DCE), Microsoft's Object Linking and Embedding (OLE) and Apple's OpenDoc are in use or under development. The fundamental concept for creating reusable objects is developing a common framework that will be used for calling base objects. This framework requires both a generic and robust interface to other objects and the underlying environment. Developing this framework does not, however, address object inheritance or the time domain of objects; the latter a fundamental simulation precept.

The fundamental obstacle to successfully applying object technology to DoD requirements is the lack of a defined underlying environment. This environment is composed of the following critical factors:

- coordinate systems
- motion
- energy propagation
- physical object interactions
- event timing

Different simulations seldom use the same coordinate system. Even simple flat Cartesian systems have two fundamental orientations, right handed and left handed. When "spherical" coordinate systems are considered, an almost infinite variety of coordinate systems are used.

The motion of simulation objects is also very complex and may need to account for gravitational changes with altitude or more conventional atmospheric conditions. Satellites orbiting earth and trucks moving over a road require completely different mathematical approaches in terms of computational precision and computational time.

Radar, infrared (IR), acoustic, and optical emitters and sensors function as energy traveling through a medium. A description of the medium adequate for radar modeling is inadequate for IR modeling, which is much more sensitive to its environment. Modeling acoustics requires an order of magnitude more detail to achieve results comparable to radar models.

Physical objects such as missiles, guns, and mines attempt to kill other physical objects such as ships, planes, and tanks. The level of interaction can vary from simple probability of kill models to detailed vulnerability models. Often, only parts of physical objects are destroyed, such as when high speed anti-radiation missile (HARM) destroys the search radar of a surface to air missile (SAM) system but leaves the rest of the site intact.

Simulations can proceed in time steps, events steps, or both. Continuous simulations run on digital computers usually use very small time steps to avoid memory and disk storage limitations. Event step simulations are more widely used and two event step paradigms are in extensive use today: event scheduling and process scheduling. A simple event step simulation requires a complex state matrix approach to ensure events are properly scheduled. Process scheduling simulations more closely imitate "real" systems by allowing processes to begin and end independently. Multi-threading the process event scheduler allows the user to more easily simulate the physical processes of a system.

Another important factor, not listed with the preceding items, concerns object fidelity. Building different levels of fidelity into an object will require level-of-fidelity standards that allow some level of interaction between objects operating at different fidelities. For example, a high fidelity radar model requires objects to fly continuous paths with continuous derivatives for changes in path direction. This is incompatible with the low fidelity point-to-point movement of objects with an average speed on each path segment.

5 OBJECT REQUIREMENTS FOR DOD SIMULATIONS

A list of requirements for objects used in military simulations would ensure that simulation objects constructed by different services and organizations remain compatible. **I propose a set of general requirements that could be used to construct compatible simulations**.

1. Physical objects composed of one or more base objects should not directly interact with other physical objects. When complex objects interact, their actions should filter up and down through their respective inheritance trees. In other words, complex objects should only interact through their inherited objects. A "simple" concept of an object's coordinate position should be directly available but only through inheritance or brokered objects.
2. The base objects should only interface with their base data structures. For example, a motion object interacts with the base coordinate system yet is able to conduct motion and report coordinates to other objects, regardless of the simulation's coordinate system. When an object, a ship for example, is moved to a simulation which operates on a different coordinate system, only the coordinate transform needs to be added to the ship's motion object so that it may use and move any of the objects in the current simulation. Similarly, radar propagation model interfaces will have to be defined to interact with the different mediums.
3. The ability to inherit from and modify an accredited object should be limited to a single inheritance level. If the accreditation of objects extends beyond simple systems to full physical platforms such as ships, objects should be used "as is" without the ability to invoke polymorphic behavior. The depth of polymorphic inheritance should also be limited to no more than three levels to maintain clarity of understanding.
4. Multiple inheritance should be limited to inheriting objects of different functions. Multiple inheritance should be delayed as long as possible in the polymorphic inheritance tree to eliminate the

accumulation of useless code in future inheritances. Simulation objects often inherit graphical interactions much too early, requiring almost all objects to be graphic capable.

5. After defining a base set of objects, the object-oriented design should be hierarchical in nature. Base objects represent the flat "playing field" that interacts with the fundamental abstract data types. A hierarchical design builds systems that are easily configured to meet changing requirements.

Over time, we will find that most problem classes will gravitate towards certain underlying abstract data structures. For example, combat-level simulations may have a tendency to use flat earth coordinates as their coordinate system and multi-regional conflict models may find spherical coordinate systems more "economical". Yet the objects of one simulation class could easily participate in the other simulation class.

Finally, we must also consider amount of data that passes between user objects and the base object as user objects set up their interaction with the environment. Only small data requests should be required to retrieve data from the environment and interact with it. Using a standard interface to every simulation carries both computation and code size penalties. I believe these penalties can be more than compensated for by optimizing the base objects interaction with their environment.

6 WHERE ARE WE NOW?

A number of simulations already exist, or are under construction, which use object technologies. One of the largest models used and worked on today is the Multi-warfare Assessment & Research System (MARS) at the Naval Surface Weapon Center (NSWC) in both Dahlgren (VA) and Panama City (FL). Written in MODSIM III, MARS models naval warfare including air, surface, subsurface, and littoral warfare. MARS has different levels of object fidelity and can easily participate in Distributed Information System (DIS) exercises and perform detailed constructive analysis.

Another Navy warfare model, the Naval System Simulation (NSS), version 1.0, uses object taxonomy concepts from MARS and model concepts from the Composite Warfare Model (CWM). While many other models used today, such as the Concurrent Theater Level Simulation (CTLS), are object-oriented, they do not possess the full object technology needed to meet future military simulation requirements for the next generation of models.

Unfortunately, no model to date fully embraces the new simulation concepts under the object paradigm.

7 SUMMARY

Joint warfare simulations in both virtual and real exercises require rapid definition of their simulation objects. The Joint Simulation System (JSIMS), the Warfighting Simulation (WARSIM), and the National Air and Space Model (NASM) have immediate requirements that can only be successful met through the employment of object technology. Joint service development efforts such as JAST place similar demands on constructive simulations. Current discussions about integrating THUNDER (the Air Force's campaign model), CTLS, and NSS into the next generation warfare model, called the Advanced Regional Exploratory System (ARES), will drive the approach to a new generation of constructive models for Cost and Operational Effectiveness Analysis (COEA) efforts.

Much like the commercial computer software community that has found success in using "standards based" systems, the military simulation community will only succeed by working together to build a coherent object-oriented approach for DoD simulations.

REFERENCES

Cox, B. J. 1944. *Object-Oriented Programming, An Evolutionary Approach.* New York: Addison-Wesley.

Meyer, B. 1988. *Object-oriented Software Construction..* London: Prentice-Hall International Ltd.

Shumate, K. 1984. *Understanding ADA..* New York: Harper & Row.

AUTHOR BIOGRAPHY

RONALD D. PAINTER is the Director of Simulation for the Projects Division of CACI Products Company where he serves as program manager for the THUNDER and MARS models. As Engineering Director at United Technologies Advanced System Division, he developed the SimMaster object-oriented simulation framework which has been used to successfully build nearly a dozen new simulations. Dr. Painter served as the chief Operations Analyst for the Advanced Medium Range Air-to-Air Missile (AMRAAM) JSPO and was chief analyst in the United States Navy's Anti-ship Missile Defense (ASMD) Program. He has been writing and using discrete-event simulations throughout his career.