

Adapting UML for an Object Oriented Systems Engineering Method (OOSEM)

Howard Lykins
Software Productivity Consortium
2214 Rock Hill Road
Herndon, Virginia 20170-4227
lykins@software.org

Sanford Friedenthal
Lockheed Martin Management & Data Systems
3201 Jermantown Road
Fairfax, Virginia 22030
sanford.friedenthal@lmco.com

Abraham Meilich, Ph.D., C.C.P.
Lockheed Martin Mission Systems
700 N. Frederick Ave
Gaithersburg, MD 20879
abraham.w.meilich@lmco.com

Abstract. Object-oriented (OO) techniques have been used predominately by software engineers. Why would systems engineers want to add OO techniques to their existing practices? How should such techniques be adapted to be useful to systems engineers? This paper will show how an Object-Oriented Systems Engineering Method (OOSEM) addresses these questions and how the use of the Unified Modeling Language (UML) at the systems engineering (SE) level can facilitate capture of system requirements and design information and ease communication between systems and software engineers. The techniques overviewed in this paper have been applied to information systems and have allowed engineers to model system needs, requirements, architectural design, and their allocation to hardware, software, databases, and manual procedures. The authors believe that OOSEM is applicable to engineering of other types of systems as well.

BACKGROUND

Motivation for OOSEM. OO techniques have been applied successfully by software engineers to software development for years. However, systems engineering, for the most part, continues to apply classical structured analysis techniques to define the system-level requirements and design artifacts. The use of different modeling techniques can result in incompatibilities and communication problems between systems and software engineering. OOSEM is motivated in part by the need to mitigate these incompatibilities.

This is not the only motivation for OOSEM. As evidenced by several recent books and articles, systems engineers are increasingly interested in the use of OO modeling to more effectively capture, analyze, and communicate system requirements and design information. However, it is the opinion of the authors that software-level OO techniques will require

adaptation if they are to fully benefit systems engineers.

Related work. OOSEM has evolved from several initiatives at the Software Productivity Consortium (the Consortium) and Lockheed Martin Corporation. The Consortium developed the Integrated Systems and Software Engineering Process (ISSEP) (Rose et. al 1996; Rose and Scott 1997; Rose 1997), which provides a process framework for integration of systems and software engineering methods. Lockheed Martin Management & Data Systems (M&DS) used ISSEP as the process framework for the initial development of OOSEM (Friedenthal 1998). Key elements of OOSEM were successfully applied to a development increment for the army global command and control system (Meilich and Rickels 1999).

Using OOSEM and other OO methods as inputs, the Consortium is developing an integrated systems and software engineering method. This method, tentatively named Object-Oriented Approach to Software-Intensive Systems (OOASIS), will provide UML-based modeling at the systems and software engineering level.

Integration of non-OO model-based SE and OO software engineering is discussed by Ali (1998).

The SE community also has indicated interest in OO techniques independent of integration with software-level OO. A recent book by Oliver, Kelliher, and Keegan (1997) uses an OO modeling technique to describe both the SE process and the results of the process. An approach for deriving an OO system design from functionally-oriented customer requirements (Hopkins and Rhoads 1998) was presented at the 1998 Symposium of the International Council on Systems Engineering (INCOSE). The need for animating object oriented models (Dockerill 1999) was discussed at the 1999 Symposium, along with some cautionary notes about using OO models for SE (Cocks 1999).

Schoening (1996) discusses use of object-based¹ software to simulate complex systems.

OOSEM APPROACH

The OOSEM approach is characterized as follows:

- It implements a well-defined SE process that encompasses the activities from needs analysis through verification based on the ISSEP process model.
- It adapts UML Version 1.3 (Object Management Group 1999) to capture system requirements and design information. UML was selected because of the rigor of its syntax and semantics and because it has become the de facto industry standard for OO software development.
- It maintains well-defined relationships between its models and components to ensure requirements traceability and system design integrity.
- It addresses the total system, including allocation of requirements to hardware, software, databases (in data-intensive projects), and manual procedures.

It should be emphasized that OOSEM does not advocate replacing specialty engineering models with UML. Rather, it encourages use of OO models to capture system and component behavioral, performance, and physical characteristics that provide the basis for integrating other specialty engineering models. Other traditional techniques and models are required to fully capture the system requirements and design information, such as performance timelines and physical layouts.

UML CONSTRUCTS USED BY OOSEM

This section summarizes the UML constructs most commonly used by OOSEM. Some constructs are used without significant change; others must be adapted for use in the systems context.

Classes, attributes, and operations. A UML class is a template from which one or more objects is derived. OOSEM uses objects to represent input/output (I/O) entities and abstract physical components that can:

- Store items such as data, energy, mass, or parts
- Perform sets of operations on inputs or stored entities in response to a stimulus.

The class from which an object is derived specifies a set of attributes and operations associated with the object. OOSEM uses UML operations to capture functional behavior associated with the system, its components, or external systems. Attributes are used to model system

state, transient and persistent data, and performance characteristics, such as response time, size, weight, and cost.

The automated teller machine (ATM) example in Figure 1 illustrates the OOSEM adaptation of UML classes, attributes, and operations. Stereotypes are used to categorize these artifacts by what they represent. For example, a class with the stereotype <<System>> represents the system under development, and <<IO>> designates an input/output entity that captures interface information (described below). OOSEM uses attributes of the <<System>> class to represent system state variables, transient and persistent data stores, and system performance parameters.

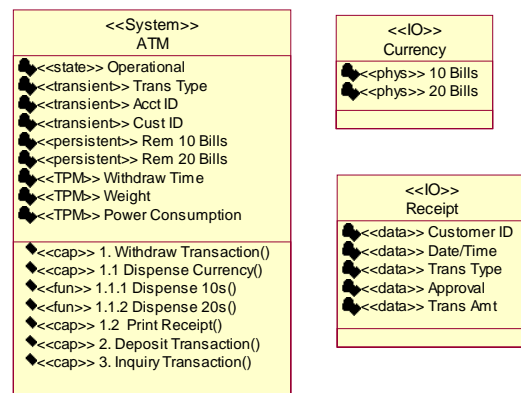


Figure 1. Classes and I/O Entities

Various component types and subtypes, such as hardware, software, and data components, also can be represented as class stereotypes.

Numbering indicates the hierarchy of system capabilities (<<cap>> stereotype) and functions (<<fun>> stereotype). In Figure 1, Dispense Currency and Print Receipt are subcapabilities of Withdraw Transaction. Dispense 10s and Dispense 20s are functions of Dispense Currency. The distinction between capabilities and functions is explained below.

Input/output entities, illustrated in Figure 1, are a special type of class that are used to capture inputs and outputs of the system and its components. The I/O entities have no operations but are characterized in terms of attributes. These attributes include data content and their associated data type, frequency, size, and data transfer method (e.g., File Transfer Protocol [FTP] and database transactions). The data content attributes represent input and output parameters to system functions. These attributes can represent inputs and outputs other than data, such as physical inputs for mass, energy, or parts. Interface control documents and interface requirements specifications are generally used to capture this information. Data attributes are identified

¹ Defined to mean “object oriented without inheritance”

by the <<data>> stereotype—physical items such as \$10 and \$20 bills dispensed by the ATM are identified by the stereotype <<phys>>.

OOSEM defines system capabilities as operations that produce or consume I/O entities. System functions operate on individual attributes of I/O entities.

Use case diagrams, illustrated in Figure 2, are used by OOSEM to conceptualize system capabilities for both the as-is system and the to-be system and provide the basis for defining detailed scenarios. They are initially defined at the system black-box level and further elaborated to capture system functionality and interaction between system components.

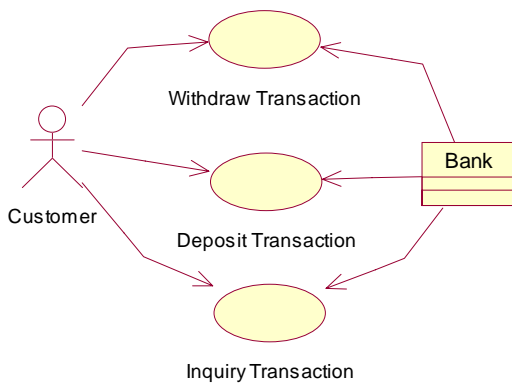


Figure 2. Use Cases

Inheritance is used to capture commonality in OOSEM as in other OO techniques. OOSEM also uses inheritance to capture potential requirements and design variations. For example, in Figure 3, a variant of the ATM system adds a funds transfer capability.

Aggregation is used by OOSEM to decompose the system into components and to decomposed I/O entities into lower-level entities.

Deployment view. OOSEM uses class diagrams or deployment diagrams to represent allocation of software and data components to hardware components and allocation of manual procedures to people or organizational components.

OOSEM ACTIVITIES AND MODELS

OOSEM includes the following development activities:

- Analyze Needs
- Define System Requirements
- Define Logical Architecture
- Synthesize Candidate Allocated Architectures
- Optimize and Evaluate Alternatives
- Validate and Verify the System

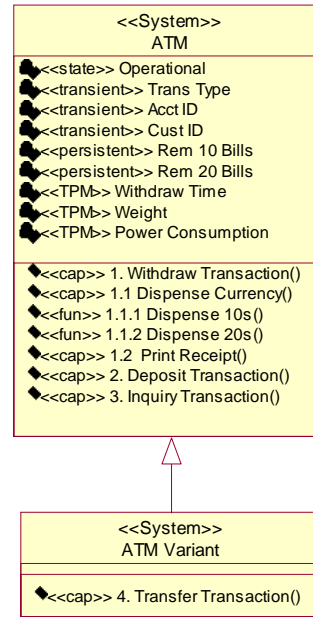


Figure 3. Inheritance

Analyze Needs. This activity characterizes the problem space by defining as-is system operation, current deficiencies, mission/enterprise-level measures of effectiveness, and required capabilities.

This activity applies use cases, scenarios, and other OOSEM models to describe the current system and its processes. Various types of system users are characterized by UML actors and the use of inheritance. The general-purpose actor User, illustrated in Figure 4, captures characteristics common to all users. Specific types of user are modeled as subclasses that add their own unique characteristics. The use cases and scenarios capture the key functionality of the current system.

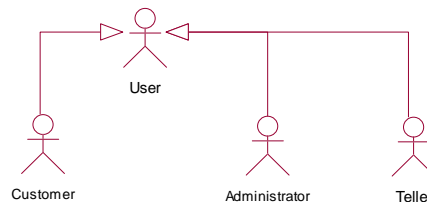


Figure 4. User Classification

Analysis of as-is use cases is used to identify deficiencies of the current enterprise (that the new system will correct) and preliminary measures of effectiveness. A root cause analysis of these deficiencies provides the basis for establishing the needs for the improved system.

An Enterprise Model, illustrated in Figure 5, uses classes and aggregation relationships to describe how an enterprise is constructed from the system to be developed and other, external systems. In the Enterprise Model, the <<System>> class is shown as a component of the <<Enterprise>> class. Additional classes denote elements external to the system with which the system collaborates to accomplish its mission. Operations on the <<Enterprise>> class denote enterprise-level capabilities (e.g., to withdraw money from the bank), which are allocated to both the system (e.g., ATM) and to the external systems (e.g., bank). Operations on the <<System>> class (shown in Figure 1), on the other hand, will describe system capabilities. Detailed elaboration of these capabilities will be captured in the Define System Requirements activity. Performance attributes of the <<Enterprise>> class (e.g., amount of time to make a withdrawal) are effectiveness measures (<<moe>> stereotype), which are allocated to the system (ATM) and external systems (e.g., bank, user).

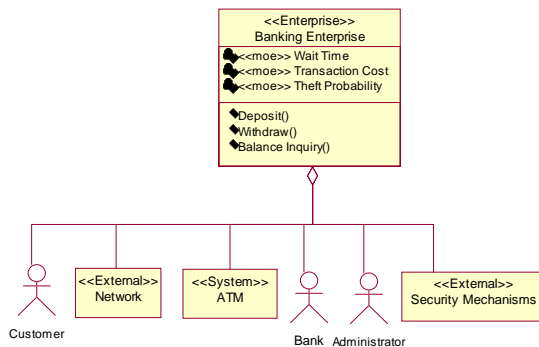


Figure 5. Enterprise Model

Define System Requirements. This activity treats the system as a black box to characterize system stimulus-response behavior and performance requirements. The system, external systems, and users are defined as individual classes as previously described in the enterprise model. The system operational concept defines a set of system-level use cases and scenarios based on the needs analysis. This operational concept is used to derive the system functional, interface, data, and performance requirements. The functional requirements are grouped into system capabilities. System states/modes and related system control information are also defined (e.g., using state machines). The resulting system requirements information is captured in an elaborated system context diagram, as illustrated in Figure 6. In the elaborated context diagram and elsewhere in OOSEM, the <<Actor>> stereotype is used to denote human users. To distinguish humans

from external systems, the <<External>> stereotype is used for the latter.

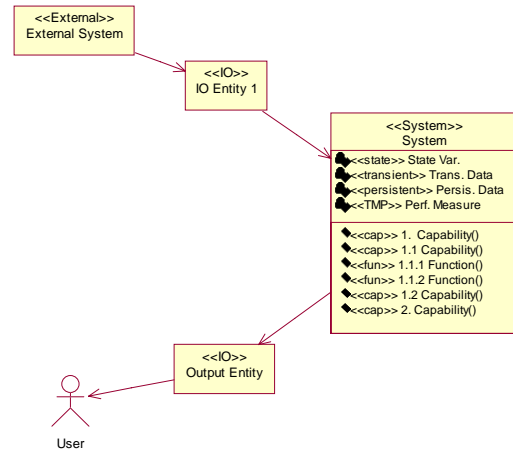


Figure 6. Elaborated System Context Diagram

This activity also identifies system design constraints, which limit the solution space. Examples are pre-identified hardware and software commercial off-the-shelf (COTS) products and legacy databases. The constraints are captured in a constraints list (external to the UML models) and used as an input to the system architecture activities. Requirements variation is also captured and quantified in terms of a probability of change. A typical examples is a system interface that is likely to change because the other side of the interface is still under development. The requirements and verification traceability (RVT) database is initiated during this activity, and traces each system functional, interface, and performance requirement to a system-level use case.

Scenarios are captured using UML sequence, collaboration, or activity diagrams. The purpose of these scenarios is to illustrate how the system will interact with actors and external systems on the context diagram. The scenarios show only the system, actors, and external systems that produce or consume I/O entities and attributes, thus providing the foundation for defining the detailed functional, interface, performance, and storage requirements. Interactions among system components are described in the next two activities.

Define Logical Architecture. This activity decomposes the system into its logical components, defines their interactions, and allocates system requirements to each logical component. OOSEM guidelines for decomposing the <<System>> class into logical components are different from traditional functional

decomposition guidelines, emphasizing, among other concerns, planning for change. The interaction between logical components is captured in data flows (represented using associations and I/O entities) and logical system threads. Partitioning criteria are established and used as a basis for partitioning the logical components to address component cohesiveness, design for change, performance, and other design considerations. System functional, interface, data, and performance requirements are allocated to logical components and captured in the RVT database.

The Define Logical Architecture activity uses aggregation relationships to describe a hierarchical view of the system logical architecture, as illustrated in Figure 7. The system is decomposed into logical components, which may be further decomposed into lower-level logical components. Each logical component is an abstraction of a set of possible system components (e.g., hardware, software, databases, manual procedures), any one of which will meet the requirements allocated to the logical component. An example of a logical component is a user interface. The potential corresponding elements in the allocated architecture (developed in the next activity) may be a Web browser with associated screen representations on a PC client. In the ATM example, a money dispenser may be a logical component that could be implemented in the allocated architecture in many different ways. Sequence, collaboration, and activity and state diagrams are used to describe interactions among logical components.

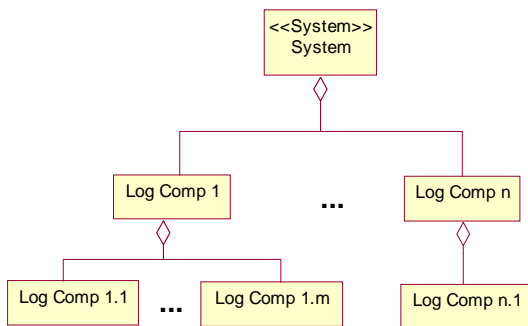


Figure 7. Hierarchical Decomposition Example

The partitioning heuristics in this activity take the following into consideration:

- Design for changes in needs, requirements, and design decisions
- Encapsulation of detailed component behavior and performance characteristics
- Reuse considerations, including use of COTS items
- Timing and other performance considerations

- Reliability and maintainability
- Environmental considerations
- Allocation of development work to subcontractors

Synthesize Candidate Allocated Architecture. This activity maps each logical component to hardware, software, data, and/or manual procedures. The resulting allocated components form the basis for a multilayer system architecture, which typically includes layers for the mission application, a services layer (i.e., functions common to multiple mission applications), and operating system and resources. The allocated components are used to derive the corresponding hardware, software, and data architectures. The software architecture is captured using the full range of UML diagrams and features. The hardware architecture uses UML class diagrams for defining hardware hierarchy; the architecture also uses interconnection and traditional hardware depictions such as physical layouts. The data architecture uses class diagrams to capture data hierarchy and relationships, along with table specifications and other traditional data architecture depictions. Figure 8 illustrates the OOSEM deployment view, using a class diagram to depict the allocation of software and data components to hardware components.

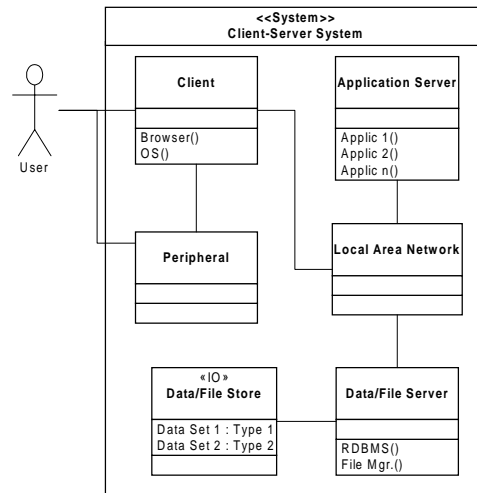


Figure 8. Allocated Architecture

Functional, interface, and performance requirements are allocated to component classes of hardware, software, data, and manual procedures. All system requirements are allocated to system components in this activity. Traceability is maintained between the logical and allocated components and captured in the RVT database.

Process engineers and software, database, and hardware developers use the allocated architecture and associated component requirements as an input to their development activities. Alternatively, components in the

allocated architecture can be procured off the shelf. Continued feedback to the system designers is provided to ensure traceability between the system requirements, system architecture, and component design. Traceability is maintained between the allocated architecture and the component design in the RVT database.

Optimize and Evaluate Alternatives. This activity is invoked throughout the other OOSEM activities to optimize the candidate architectures and conduct trade studies to select the preferred architecture. Parametric models for characteristics such as performance, reliability, availability, and life-cycle cost are used to analyze and optimize each candidate architecture to the level needed to compare the alternatives. The parametric models use architectural components defined in the logical and allocated architectures and their associated performance parameters. Trade studies are used to evaluate each alternative architecture based on defined criteria and weighting, which are traceable to the system requirements and measures of effectiveness.

The parametric models tend to use their own unique syntax and descriptions. OOSEM architectural models provide a common definition of the components, performance and physical attributes, and their interfaces, which is used by the parametric models.

Validate and Verify the System. This activity includes the development of verification plans, procedures, methods (e.g., inspection, demonstration), and analysis of verification data. System-level use cases, scenarios, and associated requirements are key inputs to the development of the test cases and associated procedures. A powerful verification technique is for the systems engineer to include the definition of the test/verification systems that implement the verification methods. As illustrated in Figure 9, the test/verification system can be represented in terms of the OOSEM artifacts, beginning with a high-level context diagram for the test/verification system. The RVT database is further elaborated during this activity to trace the system requirements and design information to the system verification methods, test cases, and results.

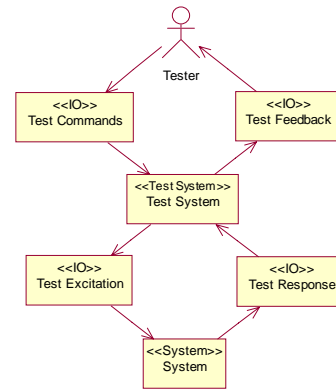


Figure 9. Test System Context Diagram

SOFTWARE OO LIMITATIONS AND OOSEM RESPONSES

This section summarizes some key limitations of software-level OO techniques for use at the system level and discusses how OOSEM addresses each limitation.

OO Limitation. “Flat” (i.e., nonhierarchical) models, such as class, collaboration, and sequence diagrams, inhibit the systems engineer from capturing the decomposition and traceability of a complete system among heterogeneous architectural components.

OOSEM Response: hierarchical decomposition. A key feature of OOSEM is that it decomposes the system into logical components then decomposes the logical components into subcomponents, continuing the decomposition until each logical component can be allocated as a whole to an element of the allocated architecture. This is in contrast to some existing OO techniques that do not take advantage of hierarchical decomposition. The fragment of a logical architecture in Figure 7 is an example of hierarchical decomposition.

OO Limitation. Lack of comprehensive definition of interfaces between the system and its external environment.

OOSEM Response: elaborated context diagram. This is the OO equivalent of the context diagram in classical structured analysis. It replaces the system transformation with a system object characterized in terms of multiple types of attributes and operations that denote system capabilities, functions, and other characteristics. Data flows are replaced by I/O entities, and terminators are replaced by UML actors and <<external>> classes that represent external interfaces. Two lessons learned apply here: First, the context diagram is just as indispensable in OO analysis as in structured analysis. Second, a context diagram can be captured effectively in an OO notation.

OO Limitation. Inability to characterize performance and other non-behavioral characteristics

OOSEM Response: multiple attribute categories. As shown in Figures 1 and 6, OOSEM uses attributes to capture required performance characteristics and requirements for transient data and persistent data. Attributes in software-level OO models tend to be items that will appear directly in the implementation. OOSEM uses attributes in this manner but also uses them to characterize key requirements and implementation characteristics. The effect is to make performance requirements explicit and to facilitate trade studies by specialty engineering.

OO Limitation. Inability to integrate systems and specialty engineering models.

OOSEM Response: application of OO models as a common framework for integrating engineering models. According to accepted SE maturity models such as EIA/IS 731 (EIA 1999), it is the responsibility of SE to integrate engineering and specialty disciplines. OOSEM facilitates this by using OO modeling to capture behavioral, performance and physical characteristics, and interfaces, which are common to other engineering models. OOSEM can be used to communicate key information among many disciplines.

OTHER UNIQUE FEATURES OF OOSEM

System-level use cases and scenarios. UML use cases are used to capture system-level capabilities and to develop scenario-based requirements, which include functional, interface, and performance requirements. System-level use cases are the basis for system scenarios, which show sequences of events and actions performed by the system, external systems, and users. These scenarios are further decomposed and allocated during logical architecture design to illustrate interactions among logical components of the system. These scenarios also provide the basis for the interactions among the physical components in the allocated architecture and are used as a basis for test cases in the validation and verification (V&V) process.

I/O entities. OOSEM explicitly captures inputs and outputs in terms of I/O entities and their attributes. OOSEM models captured in computer-aided systems/software engineering (CASE) tools use the stereotype <<IO>> to distinguish I/O entities from other types of classes. Use of I/O entities is a marked improvement over OO methods that do not make inputs and outputs explicit (or easy to find) in the model.

Disciplined use of classes. OOSEM uses a systematic approach for identifying classes and objects and their relationships. Classes can represent the system, users, external systems, I/O entities, logical components, and

the hardware, software, data, and procedural components in the allocated architecture. Rules for creating classes and objects are well-defined in the method. This explicitness is in marked contrast to guidelines in other methods such as “Look for the nouns in the problem statement to identify your classes.”

Disciplined use of associations. A key feature of OOSEM is that it constrains the use of UML associations (relationships between classes) to a well-defined set. The most commonly used associations are inheritance and aggregation. Certain other associations, such as those on the context diagram connecting I/O entities to the system and external actors, are also allowed. However, the constraints on OOSEM associations are intended to ensure that every association is implementable and traceable.

Allocated architecture. OOSEM uses a combination of OO concepts and multilayer architectures to capture allocation of system requirements and design decisions to hardware, software, databases, and manual procedures.

SUMMARY

OOSEM incorporates OO concepts and modeling approaches using UML with classical, top-down SE. The method is intended to facilitate communication among system, software, and other engineering disciplines. In addition, the method provides an effective means to capture system requirements and design information.

OOSEM encompasses the broad scope of SE activities for defining system requirements and design and allocating the requirements to system components. OOSEM includes activities for needs analysis, requirements analysis, logical and allocated architecture design, alternative evaluation, validation, and verification.

OOSEM adapts UML and, by extension, similar OO modeling notations. The adaptations include defining an elaborated, system-level class with additional attributes representing performance characteristics and defining I/O classes to capture critical interface information. Logical decomposition of the <<System>> class provides the basis for a logical architecture. OOSEM also addresses various component types, including hardware, software, data, and procedural components, which provide the basis for the allocated architecture and allocated requirements.

Future work includes refining and formalizing OOSEM guidelines, refining and improving adaptations of UML, integrating the method with a UML-based method for software engineering, and evaluating tool support for the method.

REFERENCES

- Ali, Michael, "Model-Based Systems Engineering and Object-Oriented Software Engineering: An Integrated Approach," in *Proc. Eighth Int. Symp. INCOSE*, 1998.
- Cocks, Dan, "The Suitability of Using Objects for Modeling at the Systems Level" in *Proc. Ninth Int. Symp. INCOSE*, 1999.
- Dockerill, Kevin, "The Importance of Animation with UML" *Proc. Ninth Int. Symp. INCOSE*, 1999.
- Electronic Industries Alliance (EIA), Systems engineering capability EIA/IS 731, <http://www.geia.org/eoc/G47/731dwnld.htm>, 1999.
- Friedenthal, Sanford, "Object Oriented Systems Engineering," in *Process Integration for 2000 and Beyond: Systems Engineering and Software Symposium*. New Orleans LA: Lockheed Martin Corporation 1998.
- Hopkins, Frank W. and Rhoads, Russel P., "Object Oriented Systems Engineering—An Approach" in *Proc. Eighth Int. Symp. INCOSE*, 1998.
- Meilich, Abe and Rickels, Michael, "An Application of Object Oriented Systems Engineering to an Army Command and Control System: A New Approach to Integration of Systems and Software Requirements and Design," in *Proc. Ninth Int. Symp. INCOSE*, 1999
- Object Management Group, *OMG Unified Modeling Language Specification*. Version 1.3, June 1999. <http://www.rational.com/uml/index.jtmp>
- Oliver, David, Kelliher, Timothy P. and Keegan, James G., Jr., *Engineering Complex Systems with Models and Objects*. McGraw-Hill, New York, 1997.
- Rose, Susan and Scott, Peter, "Integrated Development for Computer-Based Systems" in *Proceedings of the IEEE Technical Committee on Engineering Computer Based Systems*, 1997.
- Rose, Susan, "Engineering Harmony Between Systems and Software" in *Proc. Seventh Int. Symp. INCOSE*, 1997.
- Rose, Susan, Finneran, Lisa, Friedenthal, Sanford, Lykins, Howard and Scott, Peter, *Integrated Systems and Software Engineering Process*. SPC-96001-CMC, version 01.00.04. Herndon, Virginia: Software Productivity Consortium, 1996. (Available to the general public)
- Schoening, William, "Using Object-Based Simulations for Exploring and Testing Complex Systems," in *Proc. Sixth Int. Symp. INCOSE*, 1996.

BIOGRAPHIES

Sanford Friedenthal. Mr. Friedenthal has over 25 years of experience in SE and related engineering areas. His experience includes the full life cycle, from conceptual

design through development and production, on a broad range of systems, including missile systems, electro-optical navigation and targeting systems, and information systems. Most recently, Mr. Friedenthal was a functional manager for SE at Lockheed Martin M&DS, responsible for ensuring that SE processes were implemented on the programs, and enhancing overall SE capability. He has also been involved in developing advanced SE processes and methods, including the Lockheed Martin Systems Engineering Process (LM-SEP) and the Lockheed Martin Integrated Engineering Process (LM-IEP).

Abraham Meilich. Dr. Abe Meilich is employed as a Certified System Architect at Lockheed Martin Mission Systems. He has a B.S. in Engineering from UCLA, an M.S.M.E. from Stanford, an M.S. in Systems Management from USC, a Ph.D. in Systems Management from Walden University, and a Certified Computing Professional (C.C.P.) Certification. His career spans 30 years in the mechanical, electrical, computer, aerospace, and information SE domains. He is an adjunct professor at the University of Maryland Graduate Program in Technology Management, Systems Acquisition, and Information Systems. He is past Secretary and Newsletter Editor and past Director of the Washington Metropolitan Area Chapter of INCOSE. His current focus is on the application of SE principles and best practices in systems architecture development to the design of command and control systems for the U.S. Army and government information systems.

Howard Lykins. Howard Lykins is a senior member of the technical staff at the Software Productivity Consortium and has over twenty-five years of experience. Since joining the Consortium in 1991, Mr. Lykins has developed or contributed to a number of software engineering and SE processes and methods. His recent accomplishments include adapting the Object Modeling Technique for development of embedded real-time control software and tailoring ISSEP for component-based development. He holds a B.A. from Transylvania University and an M.S. in Computer Science from Washington University in St. Louis. Mr. Lykins is Co-Chair of the INCOSE Model-Driven System Design Working Group.

ACKNOWLEDGMENT

Rational Software provided a gratis copy of Rational Rose 2000 Enterprise Edition to the Consortium for research and development use. This tool was used to generate all figures in this paper except for Figure 8. The authors and the Consortium gratefully acknowledge their support.