Object-Oriented Development Using The Shlaer-Mellor Method

Rodney C. Montrose
Project Technology, Inc.
5800 Campus Circle Dr., Suite 214
Irving, Texas 75063-2740
800 845-1489
Internet: montrose@projtech.com
http://www.projtech.com

ABSTRACT

The Shlaer-Mellor Method for Object-Oriented Analysis (OOA) and Recursive Design (RD) is one of the most popular and widely used methods for Object-Oriented development. More than 2000 projects have used it since Sally Shlaer and Stephen J. Mellor defined the method 10 years ago. This paper will characterize three such projects: an embedded controller, a client-server information system, and a military avionics project. Each project was chosen to illustrate real-world uses of the method and the unique characteristics of the method that the projects exploited.

INTRODUCTION

Objects have been sold as the panacea for all software development ills. There are Object-Oriented (OO) methods, languages, drawing tools, operating systems, editors, and almost anything else you can think of relating to software. Why the excitement and hype? Object-Oriented promises to make software development better and deliver many benefits along the way. The common benefits cited are:

- Faster development through the use of reusable components (objects)
- Ability to handle development of large systems
- More understandable software

What exactly is an object? There are many definitions of an object but it is primarily defined as a unit of code that contains both data and functionality. This is in contrast with a typical code module that has only functionality, or a database table that only contains data. How, then, are objects supposed to provide these benefits? The general idea is to package (encapsulate) the data around functions that can operate on the data. Transferring the object to another system will immediately provide the same functionality as before, eliminating the need to write code. Developers can concentrate on writing the object's functionality (methods) independent of other objects in the system. By working on each object independently and connecting them together (through inheritance and message passing), this will allow faster development of large systems. To understand a software system, you only have to understand the interconnections between the various objects - not the details of each object's code.

The important question is: Do objects and Object-Oriented development provide common benefits? This paper will examine three projects that utilized the Shlaer-Mellor Method to develop systems in an OO fashion. It will determine: 1) if reusable objects were developed, 2) if reusable objects helped the system development, 3) if reusable objects made the system easier to understand, 4) if reusable objects helped structure a system that was easier to maintain. It will also discuss the unique characteristics of the Shlaer-Mellor Method and how it benefited the projects.

OVERVIEW OF THE SHLAER-MELLOR METHOD

The Shlaer-Mellor Method [1,2,3] comprises the following steps, grouped into two separate activities: Object-Oriented Analysis (OOA) and Recursive Design (RD).

I. Object-Oriented Analysis

a. Partition the system into domains.
b. Perform a detailed Object-Oriented Analysis (OOA) on each domain that must be built. OOA consists of developing an Object Information Model (OIM), State Model (SM) and Process Model (PM) (see figure 1).
c. Verify the analysis of each domain.

II. Recursive Design

a. Determine an architecture and specify the rules for translation of the OOA models into implementation.
b. Build the translation components.
c. Translate the OOA models into implementation.

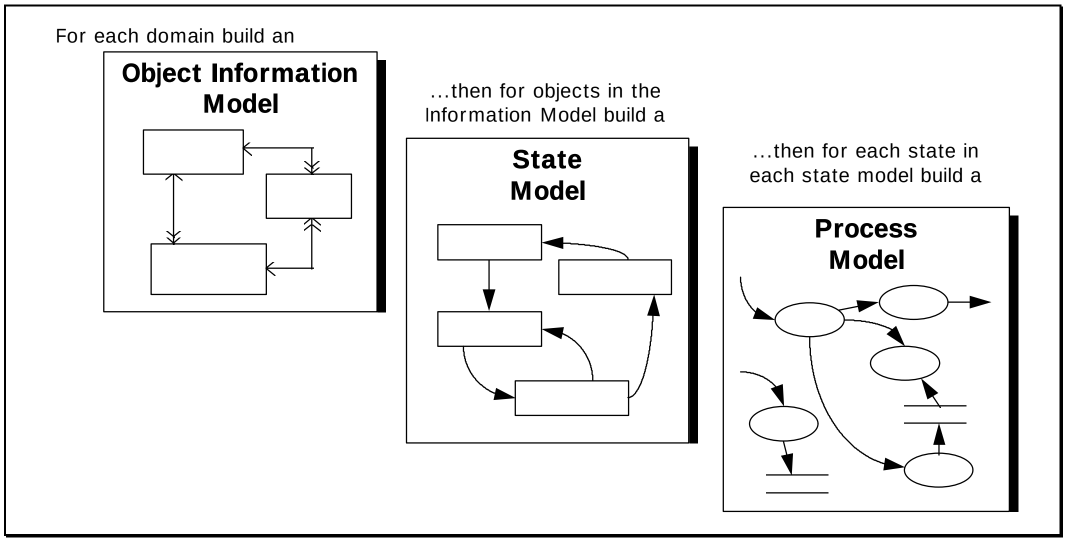
For each domain build an
Object Information Model
...then for objects in the Information Model build a
State Model
...then for each state in each state model build a
Process Model

Figure 1 - Steps in Shlaer-Mellor OOA

There are many different Object-Oriented methods in current use. While a casual glance at the notations may indicate similarities, there are significant differences between methods. The **fundamental** difference between the Shlaer-Mellor Method and others is that Shlaer and Mellor view analysis and design as two separate subject matters (domains) while the others view analysis and design as a difference in the level of detail described. The differences in notations, ways of identifying objects (roles or use cases), and the amount of "OO-ness" (encapsulation, inheritance) are all a result of this fundamental difference. This approach leads to three main characteristics of the method.

1) Domains: A key component in the method is early separation of the problem into various subject matters or domains. A domain is defined as: <u>A separate real, hypothetical, or abstract world inhabited by a distinct set of objects that behave according to rules and policies characteristic of the domain.</u> What does this mean? A defined world (i.e., the abstract world of User Interface) has a set of objects (Icon, Window, Dialog Box) that exhibit behavior expected of objects in that domain. For instance, icons can overlap other icons and dialog boxes cannot be moved outside a window. Almost every system is a mixture of different subject matters. This mixture might include (for example) the Application, User Interface, Peripheral

Interfacing, Operating System, Computer Language, and Reporting (see Figure 2). Typically, these are all combined together, making it necessary for an individual working with the project team to be an expert in each domain. This is called *vertical partitioning* [4]. Treating these as separate domains allows individual specialization in each problem area, often in parallel. This is called *horizontal partitioning.*

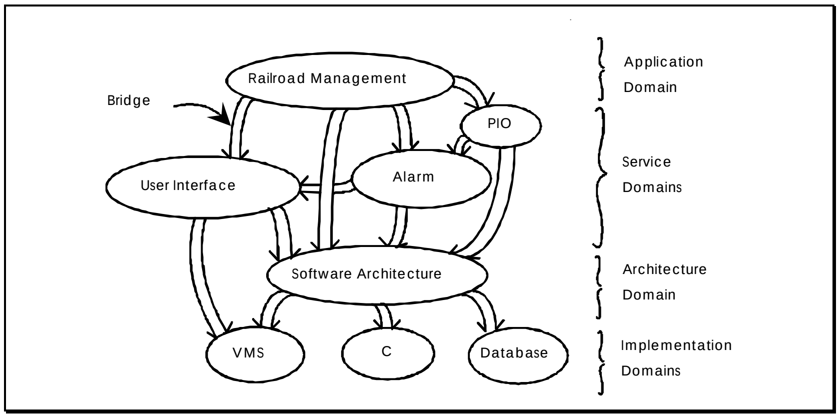

Figure 2 - Domains exist for different subject matters in a system

Domains exist as separate, reusable frameworks available to project teams that have similar requirements. Through reuse of domains, users of Shlaer-Mellor often achieve 50 to 60 percent reusability of the OOA objects, as compared to typically 5 to 15 percent reusability of code classes. Domains achieve their independence through the use of bridges which connect the requirements in the client domains to the functionality provided by the service domains (see Figure 3). In the example shown, a new application domain, requiring the display of icons, can reuse the objects (in the User Interface domain) that provided this capability. In many companies that develop families of products, only the application domain changes between projects.

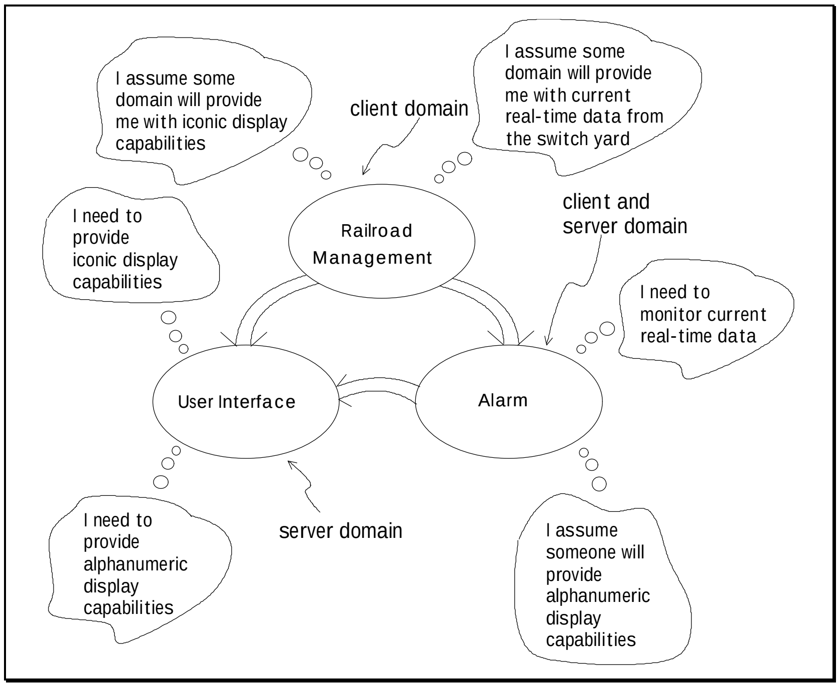
I assume some domain will provide me with iconic display capabilities
client domain
I assume some domain will provide me with current real-time data from the switch yard
I need to provide iconic display capabilities
Railroad Management
client and server domain
I need to monitor current real-time data
User Interface
Alarm
I need to provide alphanumeric display capabilities
server domain
I assume someone will provide alphanumeric display capabilities

Figure 3 - Client/Server Relationships between domains

2) Purpose of Analysis: In many methods (i.e., Object Modeling Technique/OMT or Rumbaugh method [5]), analysis is viewed in less detail than shown in the design phase. Shlaer and Mellor define analysis as a complete, unambiguous definition of a problem. The analysis is complete when a detailed description of the problem is captured in the OOA models. There is a well defined formalism that instructs analysts how to model in each domain. This amount of rigor is needed to provide a clear definition of the problem and to allow it to be translated into implementation (code). This is similar to a compiler requiring the source file to be in the exact syntax of the language. Informal models cannot be translated. This structure also allows a complete static check and dynamic simulation of the problem.

3) Implementation through Translation: This difference allows Shlaer-Mellor analysis models to be translated (compiled) directly to code. With other methods the developers evolve the objects from sketchy analysis drawings to final code by adding detail at the various steps in the development process.

The result is that the Shlaer-Mellor analysis models contain far more detail than what people normally expect in analysis. The OOA models contain a level of detail that people often associate with detailed design or coding. This is because "level of detail" is not used to separate analysis from design or coding. Considerable time will be spent in the analysis phase of a project, but, this time will be made up in the coding phase. In the Shlaer-Mellor courses, Project Technology instructors teach users to expect that approximately 60 percent of the project's time will be used to develop the OOA models.

USE IN THE REAL WORLD

The three project teams included here were chosen as examples of the method as used in different industry applications and will illustrate use of various facets of the method. The project descriptions list the challenges that projects encountered using the method so that readers get an unbiased report on the use of the method. Few real-world development projects are picture-book perfect; every group has its own difficulties it must overcome.

Each project received training and consulting services from Project Technology Inc., Sally Shlaer's and Stephen J. Mellor's company that promotes and supports the method. The projects are identified as "Project A, B or C" because the real names do not matter as much as the project team experience. Each project's report follows a common format to allow comparisons. The results obtained can be compared with the projects described in Michael Lee's paper "Object-Oriented Analysis in the Real World" [6], in which he examined five separate projects: the process each used to implement OOA in the systems, and the challenges each faced.

PROJECT A

Type of Project	Information Technology, Client-Server
Language, OS	C++, Sun Solaris Unix, Motif Windows builder
Number of Analysts	10
Number of Architects	3
Project Length (planned/actual)	1 year / 14 months
Number of Domains	3
Number of Objects	120 in Application domain, 30 in two Service domains
Number of State Models	About 1/3 of the objects have State Models
Time in Analysis	9 months
Time to Build Architecture and Translation	6 months
Time in Coding	1.5 months (Automated code generation)

Project Overview: This project is an on-line information system. There is a main Unix server with client workstations connected to it. The server provides information from either its main database or from a network connection to a mainframe computer. The client workstation is responsible for presenting the data and handling any data entry or changes.

Project Goals:

1. The project was the first in a series of information providers. The objective was to decrease the development time for additional projects. Reusability of large groups of objects (domains) in other business projects was very important.

2. The project team wanted support for various hardware platforms and future changes in hardware and operating systems.

3. The team wanted fast maintenance Lifecycles. The information and system operation changed often and the team wanted to drastically reduce the time needed to implement these changes.

Unique characteristics of this project: The project team consisted of a group of people with excellent familiarity with the problem domain but little experience in C++ or developing client-server applications. The method allowed the application domain developers to focus on the problem (application and service domains) without getting involved with language or operating system issues (software architecture domain).

The team took advantage of Recursive Design to automate much of the code production process. They developed an architecture specifically for their client-server system. Object access across the network was transparent to the developers; the developers could access any object's data, regardless of which machine it resided upon. The architecture generated C++ code for the Unix clients and database server.

The user interface applied the Motif X-Windows system and a GUI builder to generate the interface code. This meant that the User Interface domain was well understood and no analysis was needed on this domain. The project team produced a list of User Interface domain requirements from other client domains and used this list to detail how the GUI builder was to assemble the interface.

Benefits using Shlaer-Mellor Method: The structuring of the project by use of domains was the largest benefit. It allowed the team to develop reusable domains of large numbers of objects. The service domains are already being reused in other projects, drastically reducing their development time.

By isolating the code and operating system dependencies in the RD translator, there is the capacity to develop a family of translators for various hardware platforms. This also allows experimentation with various client and server partitioning of the code without affecting the source (the OOA models). The automation of the RD phase allows maintenance activities to be performed on the analysis models and then translated to produce the code quickly.

Team members also found that the OOA models made it easier to communicate with the various developers and with other groups and users of the system.

As other groups began to use the method, team members found that a formally defined OOA method by-passed ambiguities in the analysis models and made the transfer of information easier.

Challenges using Shlaer-Mellor Method: The largest obstacle was the lack of management trust in the idea of code generation through translation. This was overcome the first time that management translated models. The translation concept is one that must be successfully used before it is trusted and accepted by most developers. This problem is common with projects using Recursive Design.

The developers also encountered some problems in communication with other groups that expected to see textual requirements instead of OOA models. An educational process was needed to allow these groups a chance to review the models and understand how they fit into the software development process.

PROJECT B

Type of Project	Embedded Controller / Programmer
Language, OS	C++, pSOS (Controller), MS-Windows (Programmer), Case-W GUI builder
Number of Analysts	4 on Controller, 2 on Programmer
Number of Architects	2: one for Embedded, one for MS-Windows
Project Length (planned/actual)	1 year / 2 years
Number of Domains	7
Number of Objects	250 objects in all domains
Number of State Models	150 State Models
Time in Analysis	1 1/3 years in analysis
Time to Build Architecture and Translation	6 months
Time in Coding	1/2 year (4 phases of code deliverables)

Project Overview: This project team had to develop two applications: (a) the Controller product and (b) a Programmer for the Controller. A customer would purchase one programmer to customize many different Controllers. The Controller had an Intel microprocessor embedded within it, while the programmer was designed to run on any Windows 3.0 PC.

Project Goals:

1. The project team wanted to produce a product that met current and future requirements for a world-wide market. In the past, products developed using structured techniques were difficult to modify as requirements changed, and team members hoped that an OO approach would allow faster implementation of new requirements.

2. The team wanted to use a defined process that would meet ISO-9000 and SEI level 3 requirements. The company has a reputation for the best in quality and developers were actively working to improve it.

Unique characteristics of this project: The project was planned for one year but took two years to complete. There were a variety of reasons for this, the foremost being that the project was a lot larger than expected. The one year expectation was determined before requirements were resolved (like most real projects!). Once the group began analysis of the requirements, the scope of the project was quickly understood. Four months into the project, the group faced a major schedule slippage. Team members viewed the method as beneficial since it forced them to look at and analyze requirements that would have been delayed. This "early warning" allowed project management to make a business decision: Should the schedule be extended or should some of the requirements be removed? The decision was made to push the schedule out a year and to schedule some of the international product requirements for a later release. The team members were adamant that the Shlaer-Mellor Method saved the project. A person was hired whose only responsibility was to collect and analyze metrics of the process. Data was gathered on the length of time needed to model an object, review a model, and implement corrections. Data was also collected on the number of corrections put into each model and other detailed measurements. This data was later useful to team members for accurate prediction of project time and staffing requirements. It allowed team members to measure the improvement in performing OOA during the life of the project.

The project consisted of two pieces: the Controller and the Programmer. Even though the two applications were implemented on different architectures and had different purposes (application domains), they shared several service domains. The project's domain chart had two application domains (Controller application and Programmer application) and two Software Architecture domains (one for pSOS and one for Windows).

Benefits of using the Shlaer-Mellor Method: As stated before, the ability to quickly and accurately assess the size of the project and make a rational decision on how to proceed was a benefit of the method. Once the process metrics were collected, the ability to accurately predict time and staffing was a further benefit of this approach.

The project is completed and maintenance updates to the product are being released. These updates primarily incorporate requirements that were deleted from the first version as well as newer requirements. Team members are finding it very easy to incorporate the changes into their OOA models and to translate them into the implementation. Even though the translation process was done manually, the project team felt that the final code was more stable with less ripple effect from changes than maintenance of the source C++ code. The ability to replace a single domain with a new, different domain for introducing new groups of features to their products is very useful.

Another benefit (because the Software Architecture is the keystone to the performance of their system) is that relatively simple changes to the Software Architecture can lead to drastic improvement in the system performance.

The project received ISO-9000 certification and is currently conducting an SEI audit which is expected to receive a level 3 or 4 rating for this project.

Challenges using Shlaer-Mellor Method: The developers experienced the majority of their challenges while developing the Programmer. The Programmer was a Microsoft Windows-based application that allowed each customer to graphically program the Embedded Controller for unique requirements and functionality. Shlaer-Mellor OOA treats objects as concurrently executing entities communicating asynchronously between each other. Microsoft Windows thinks of operations as synchronous communications. While RD supports the translation of OOA objects into a synchronous architecture, the analysts had difficulty thinking about the problem using asynchronous objects.

Developers found that Project Technology courses and the Shlaer-Mellor books provided many examples of the modeling ideas to be used and referenced in their OOA models. Unfortunately, because the RD book was not publication-ready, there were fewer reference examples while constructing their Software Architecture.

PROJECT C

Type of Project	Military Avionics
Language, OS	Ada, Proprietary Ada Kernel
Number of Analysts	70
Number of Architects	3 architects
Project Length (planned/actual)	5 years / 3 years to date
Number of Domains	16
Number of Objects	10-50 per Domain (About 500 Objects)
Number of State Models	About 300 State Models
Time in Analysis	2 years
Time to Build Architecture and Translation	1 year
Time in Coding	1/2 year

Project Overview: This project team was responsible for rewriting the avionics software for an Air Force fighter. The purpose was to bring the software from a combination of different languages into a single language, Ada, to allow the incorporation of future features into the software, and to introduce improvements in the hardware.

Project Goals:

1. The project team wanted to develop reusable software components to be used in this and future avionics systems.

2. The team wanted to produce a system allowing rapid changes of functionality, ease of maintenance, and portability to different hardware and languages - primarily Ada 9X.

3. The team also wanted to produce a highly reliable, fault tolerant system.

Unique characteristics of this project: The project team had spent one year using a classic Ada design method. The challenge the team encountered was that the Software Architecture (operating environment, hardware) was constantly changing and team members could not proceed with designing the system. Any work done was reworked to accommodate any changes. The team had a proprietary CPU that was modified to improve system efficiency. The Shlaer-Mellor Method allowed team members to isolate the changing hardware/OS from the development of the application and to proceed with developing the primary application's OOA models. The method facilitated this process through the separation of subject matter in the problem into domains.

There was a large number of analysts on the project and they needed a rigorous process to ensure that the team worked together and that objects developed by one analyst could be used by another. The Shlaer-Mellor Method was the only one analysts found to meet this criterion.

Benefits using Shlaer-Mellor Method: The primary benefit cited was the rigor of the analysis formalism. This resulted in consistent models **-** a feature analysts found lacking in other methods. The formalism allowed the analysts to concentrate on building correct models of the problem without having to interpret and refine the method.

The separation of the applications and software architecture domains provided two benefits. First, the analysts could focus on the application problem without having to worry about the implementation problem. Similarly, the designer could focus on implementation issues (operating system, hardware, communication) without getting overloaded by the application. Even though analysts were performing translation by hand, the opinion was that the code produced was cleaner and easier to understand than if 70 individuals had each coded his/her own way. Team members have been able to try several optimizations of the hardware processor without worrying about the affect on the analysis models.

The project has generated several newer r projects to develop specialized enhancements for the main system. These groups are able to leverage the previously built domains and considerably shorten their development time. The project succeeded in developing reusable software components for use in avionics systems.

Challenges using Shlaer-Mellor Method: The principal challenge was in ensuring that the Shlaer-Mellor Method and 2167A (a DOD documentation/process standard) existed in harmony. 2167A is based around a functional decomposition, waterfall lifecycle and requires work products (deliverables) produced in an order different from the manner in which they naturally occur in the Shlaer-Mellor Method. For example, it requires architectural decisions to be made during the initial requirements analysis, while the method allows this to be postponed until very late in the process. Project Technology Inc. publishes a reference guide on how to tailor 2167A for use with this method [7], and NCCOSC has an OO Design Process document that incorporates the Shlaer-Mellor Method and 2167A [8]. Unfortunately, the tailoring was finalized before the method had been selected and the choice of documentation products to be produced was final.

SUMMARY

This paper examined several projects to determine whether: 1) reusable objects were developed 2) objects helped with development of the systems, 3) objects made the systems easier to understand, 4) objects helped make the systems easier to maintain.

1) Each project team cited the development of large groups of reusable objects (domains) as one of the main benefits of this method. Project B was able to leverage these domains of objects in their original project development by sharing domains between the Controller and the Programmer.

2) Each project team was successful in developing its product. For Projects A and C, the formalism of the method greatly helped in the development of the system. Even though Project B did not meet its

original schedule, there was no problem meeting the new schedule after using the method to help determine the true size of the project. Project A, which had used another method unsuccessfully for a year with little results, found that the Shlaer-Mellor Method allowed them to develop their system during hardware requirement changes.

3) Each project team found that the method facilitated system understanding and enhanced communication between the developers and other groups. An issue raised by Projects A and C was that of integration into existing documentation frameworks. The Shlaer-Mellor Method introduces changes in the development process and all groups that interact with this process must be aware of the format of the deliverables produced by it.

4) Each project team found that using the method made system maintenance easier and each was able to respond to changing requirements quicker, with fewer side effects. Separating the implementation into the software architecture domain is one activity that gained the maximum benefit because characteristics of the Software Architecture (hardware, operating system, performance) were changeable without affecting the analysis models.

Did the use of objects, solely, give the developers the benefits they observed? Concepts such as separation of subject matters into domains, a formal OOA method, and Recursive Design provide these benefits. The Shlaer-Mellor Method uses objects because of the benefit of grouping data and functionality into a common entity, but it is not expected that objects, **alone**, will greatly benefit the development process. Objects are like building blocks: unless there is a defined and verified plan to assemble the blocks into a steady structure, they may end up being assembled into a structure that will soon collapse. For this reason, projects that are based solely on implementing objects in C++ or Smalltalk, without a formal method, have found little benefit in adapting the object-oriented paradigm [9].

In summary, the primary benefits that users experience using the Shlaer-Mellor Method are :

- Separation of different subject matters into domains: this allows large scale reuse of analysis objects between projects, and the reuse of software architectures and designs. It helps focus the analysts to the problem.

- Ability to build application-independent designs allowing translation of the OOA models into implementation: this was illustrated with both automated and hand translation, which give similar benefits, but differed in the amount of time used. Automation allows coded objects to be produced in minutes instead of hours.

- Communication between groups of developers occurs in an unambiguous, well defined language (OOA).

These three projects from different industries also illustrate the universal applicability of this method in software development.

REFERENCES

[1] Sally Shlaer, Stephen J. Mellor, Object-Oriented Systems Analysis: Modeling the World in Data, Prentice Hall, Englewood Cliffs, NJ, 1988.

[2] Sally Shlaer, Stephen J. Mellor, Object Lifecycles: Modeling the World in States, Prentice Hall, Englewood Cliffs, NJ, 1991.

[3] Sally Shlaer, Stephen J. Mellor, "The Shlaer-Mellor Method", paper available from Project Technology Inc., 1993.

[4] Sally Shlaer, Stephen J. Mellor, "A Deeper Look...at Partitioning the Works on a Project" column, Journal of Object-Oriented Programming (JOOP), September 1993.

[5] James Rumbaugh, Michael Blaha, et. al., Object-Oriented Modeling and Design, Prentice Hall, Englewood Cliffs, NJ, 1991.

[6] Michael M. Lee, "Object-Oriented Analysis in the Real World", paper available from Project Technology Inc., 1992.

[7] Fredrick N. Hill, Stephen J. Mellor, Klancy de Nevers, Sally Shlaer, "Documentation Guidelines for Ada Object-Oriented Development Under 2167A", booklet available from Project Technology Inc., 1989.

[8] Commanding Officer, NCCOSC, "Draft OBJECT-ORIENTED DESIGN PROCESS", document SEPO-SPE-OOD-PR-03/31/93-1.0, 1993.

[9] "Object Orientation Under Fire", Software Management News, May-June 1993, Vol. 11, No 3, Page 6.