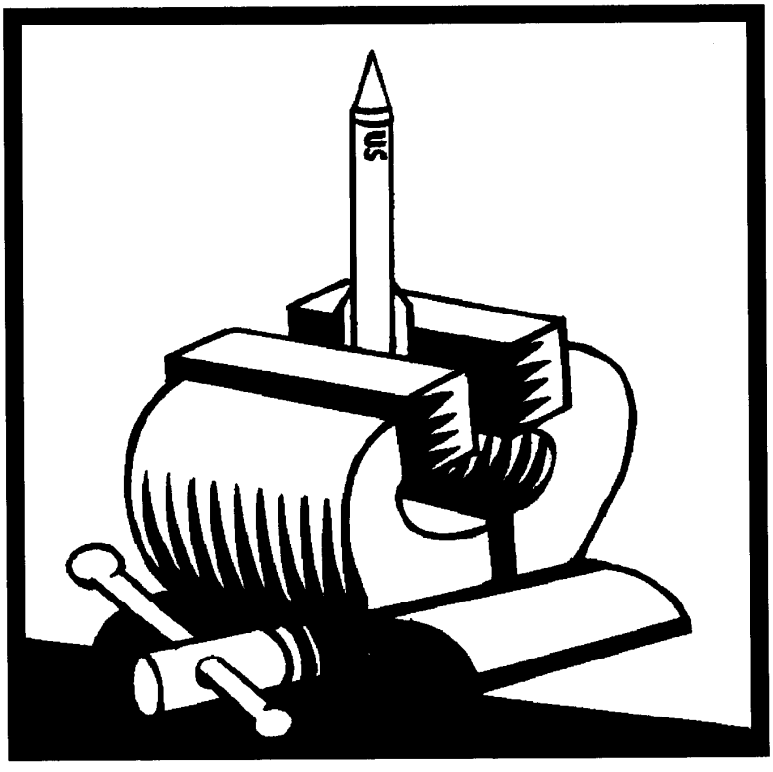
USING THE SHLAER-MELLOR OBJECT-ORIENTED ANALYSIS METHOD

The Shlaer-Mellor method is good for applications with well-defined requirements, such as databases. It supports bottom-up development and produces better abstract objects than most methods.

MOHAMED E. FAYAD
LOUIS J. HAWN,
MARK A. ROBERTS
JERRY R. KLATT
McDonnell Douglas Corp.



As part of the guidance-software group at McDonnell Douglas's Missile Systems Division, we examined the feasibility of adapting object-oriented analysis to engineer the requirements of a mission-planning system.

Before starting this project, we evaluated several object-oriented requirements-analysis methods and selected the one developed by Sally Shlaer and Stephen Mellor.[1,2] The Shlaer-Mellor Object-Oriented Analysis Method provides a structured means of identifying objects within a system by analyzing abstract data types. The analyst uses these objects as a basis for building three formal models: information, state, and process.

Our group successfully adapted the Shlaer-Mellor method and used it to engineer the requirements for the Mission Generation System, which is described in the box on p. 50. The MGS development team consisted of six software engineers and a domain expert with five years software-development experience. All team members had a minimum of three years experience in Ada software development and two years experience in object-oriented methods.

In adapting the Shlaer-Mellor method, we learned that it consistently produced better abstract objects than other object-oriented methods (in which data objects and operations are packaged according to

0740-7459/93/0300/0043/$03.00 © IEEE

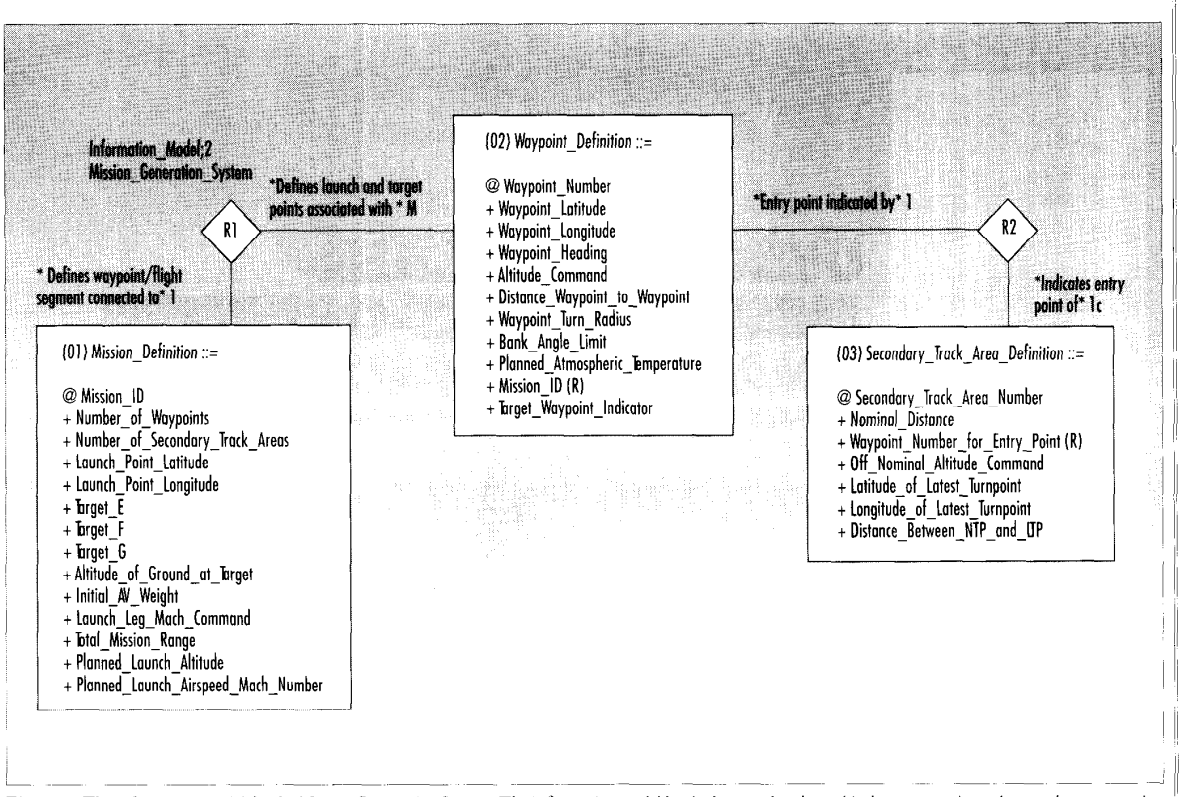


Figure 1. *The information model for the Mission Generation System. The information model has both textual and graphical representations. A textual representation consists of a set of textual definitions (semantics) for each object, attribute, and relationship describing the basis of abstraction. The graphical representation provides a global view of the entire MGS.*

the method's rules), and that it is best applied to information systems or to reengineering situations, in which data objects are already identified.

WHY AN OBJECT-ORIENTED APPROACH?

We decided to use an object-oriented approach for two reasons.

First, we had successfully used Ed Colbert's approach to object-oriented software development[3] to develop a hardware-in-the-loop simulation. This type of simulation provides a real-time environment for simulating a hardware system, which in this case was a missile-guidance computer in flight.[4]

Second, we found a number of shortcomings in structured analysis and design approaches. The translation from requirements analysis to design is ill-defined and difficult. Mapping and tracing between development phases are usually done manually using walkthroughs and inspections. Finally, for our application we needed support for Ada packaging and class definitions, which is not provided.

APPLICATION

The first step in the Shlaer-Mellor method is to define the information model, which consists of data objects, attributes, and relationships derived from the real-world problem. It took us about two months to develop the information model. Once we had completed it, we used state models to formalize the life cycles of objects and relationships according to the operating policies, laws, and rules of the problem domain. Each state within a life cycle is used to generate a process model. The process models represent the final step in the analysis.

Creating the information model. The information model addresses the static aspects of all the objects and is the method's cornerstone. Figure 1 shows the information model for the MGS. The model describes the relevant objects in mission planning (the application domain) but ignores the temporal dimension. It uses a notation similar to that of an entity-relationship model.

The information model can be produced both as textual and graphical representations. A textual representation consists of a set of textual definitions (semantics) for each object, attribute, and relationship describing the basis of abstraction. The graphical representation provides a global view of the entire MGS, as Figure 1 shows.

To create the information model, you identify objects, determine the objects' attributes, and construct the model itself from the attributes and the relationships between objects. The Shlaer-Mellor method provides various tests to help you determine what objects exist, as well as defining object interrelationships.

Objects. An object is the abstraction of a set of real-world things. An object has instances, each of which represents one real-world thing. All instances must have the same characteristics and are subject to and must conform to the same rules.[1,2]

In the Shlaer-Mellor method, most objects fall into five categories, which are used only to assist in object definition

(after an object is defined, they are meaningless). (The authors give no basis or rationale for these classifications):

♦ *Tangibles.* Tangible objects are abstractions of something in the physical world. An example is a piece of equipment.

♦ *Roles.* Role objects represent the purpose or assignment of a person, piece of equipment, or organization. An example is a supervisor.

♦ *Incidents.* Incident objects represent an occurrence or event, something which happens at a specific time, such as an election.

♦ *Interactions.* Interaction objects generally have a transaction or contract quality, and are related to two or more other objects in the model. An example is a user request or a contract.

♦ *Specifications.* Specification objects represent rules, standards, or quality criteria. An example is a mission definition. All objects of MGS are specification objects, as Table 1 shows. (The table also gives examples to illustrate other categories.)

The Shlaer-Mellor method provides a set of refinement criteria to help you identify the right objects and reject the wrong ones. Each object must meet all these criteria, which form the basis for the object-test matrix in Table 2:

♦ *Uniformity test.* Each instance must have the same set of characteristics and be subject to the same rules.

♦ *More-than-a-name test.* An object must have characteristics other than its name that you can use to describe it. These characteristics are attributes. A person (object) has a name, Social Security number, and an address (attributes), for example.

♦ *Or test.* The inclusion criteria in the object description must not use the word "or" in a significant way.

♦ *More-than-a-list test.* The inclusion criteria in the object description must not be merely a list of instances.

Attributes. An attribute is a single characteristic possessed by all the entities that were abstracted as an object. It is essentially data about the object, which the system must store and retrieve. Attributes can be normalized by applying the following criteria, which form the basis for the object-attribute test matrix in Table 3:

♦ *One value per attribute.* An instance can have only one value for each attribute.

♦ *No internal structure.* An attribute must have no internal structure. For example, the attribute "age" for the object

TABLE 1
MATRIX OF OBJECT TYPES

Object	Tangibles	Roles	Incidents	Interactions	Specifications
MGS objects					
Mission definition					X
Waypoint definition					X
Secondary track area					X
Illustrative objects					
Mission summary*		X			
User request*				X	

**Mission summary is a graphical representation of a selected segment of a mission presented to the user in a form selected by that user. User request is an action by the user in which some mission-planning tool option is selected to complete some desired task.*

TABLE 2
MATRIX OF OBJECT TESTS

Object	Uniformity test	More-than-a-name test	Or test	More-than-a-list test
MGS objects				
Mission definition	Pass	Pass	Pass	Pass
Waypoint definition	Pass	Pass	Pass	Pass
Secondary track area	Pass	Pass	Pass	Pass
Illustrative objects				
Mission summary	Pass	Pass	*	Pass
User request	†	‡	Pass	‡

Objects are defined in Table 1.
**Although mission summary does not use "Or" prominently, a summary can appear as a hardcopy or on a screen, which may warrant two objects rather than one.*
†An attribute of user request, Action to Be Completed, may not be uniform and hence may not conform to the rules.
‡User request may be considered a list of user options.

TABLE 3
MATRIX OF OBJECT-ATTRIBUTE TESTS

Object	One value per attribute	No internal structure	Attribute characteristic of entire object	Represent characteristic of instance named
MGS objects				
Mission definition	Pass	Pass	Pass	Pass
Waypoint definition	Pass	Pass	Pass	Pass
Secondary track area	Pass	Pass	Pass	Pass
Illustrative objects				
Mission summary	Pass	Maybe*	Pass	Pass
User request	Fail†	Maybe‡	Fail§	Fail§

Objects are defined in Table 1.
**An attribute of mission summary, Mission Range Selected, may represent an internal structure.*
†An attribute of user request, Action to be Completed, can have more than one value.
‡Action to be Completed may represent an internal structure.
§Option Requested, an attribute of user request, and Action to be Completed are functionally independent.

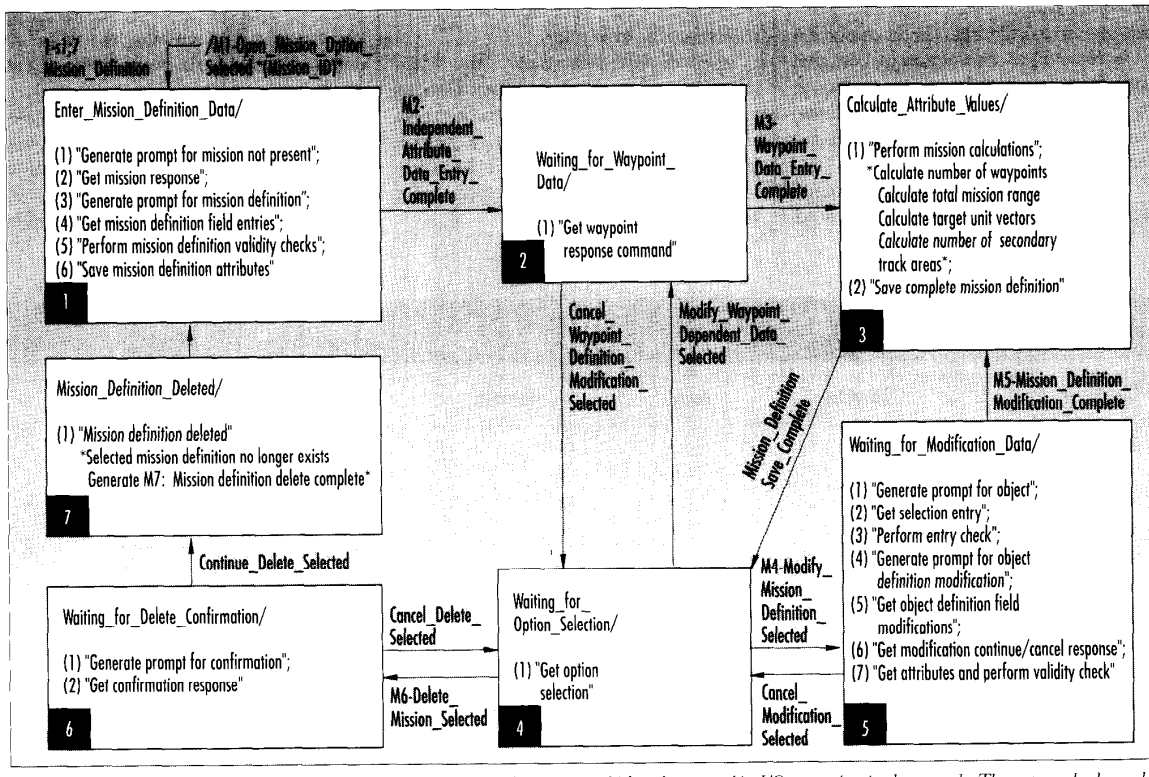


Figure 2. *The life cycle of the mission object shown as a state-transition diagram, in which each state and its I/O are maintained separately. The rectangular boxes show different states. Arrows indicate transitions during an event.*

"person" would pass the test because it is only one thing. "Address" would not because it is composed of street, city, state, and zip code.

♦ *Attribute characteristic of entire object.* When an object's identifier has two or more attributes, every attribute that is not part of the identifier must represent a characteristic of the entire object, not just part of it.

♦ *Represent characteristic of instance named.* Each attribute that is not part of an identifier must represent only a characteristic of the instance named by the identifier.

Relationships. A relationship is the abstract set of associations that systematically hold among objects. Relationships show how each object is related to the other objects and how it forms a part of the whole system. Relationships are represented by lines drawn between objects. Each relationship is described by two verb phrases. For example, relationship R1 in Figure 1 defines the waypoint-light segment connected to a mission and the launch and target points associated with each mission. This relationship is a one-to-many relationship. Relationship R2 in the figure is a one-to-one relationship between the

waypoint and secondary-track-area definitions. The MGS information model does not contain a many-to-many relationship.

Creating state models. State models formalize object life cycles and relationships by constructing a life-cycle diagram for each nontrivial object in the information model. (We defined an object as nontrivial on the basis of our team's domain knowledge.) Each state in the life cycle represents a condition of the object during which a defined set of rules, laws, and policies apply.

Each state in the life cycle accomplishes at least one action. An action can be made up of any number of smaller action processes which, as a whole, complete the action for that state. An object does not change state until the action of that state is completed.

An event is an occurrence that tells when an object is moving or needs to move from one life-cycle state to another. An event is the object's actual transition.

There are two types of state models. State-transition diagrams show each object's dynamic behavior; an object-communication diagram represents object interactions graphically.

Figure 2 shows a state-transition diagram, in which each state and its I/O — in the form of communicating events — are maintained separately. The Shlaer-Mellor uses E.F. Moore's state-transition diagrams, in which actions are performed when the object enters the state.[5]

Figure 3 shows an object-communication diagram. Each object is shown as a rectangle; connecting arcs are labeled with communicating events.

Process models. The process model depicts the action processes associated with each state within a state model. It is essentially a dataflow diagram, similar to that in structured analysis, that describes the actions associated with transitions. Figure 4 is a process model of the MGS's mission object state "enter mission definition data." Each circle in the diagram represents an action process of a given state. The dataflow diagrams contain processes (like Get_Mission_Response), data stores, dataflows (like Mission_ID, which creates a new mission), and a number of sources and sinks.

CASE tool support. We used Cadre's Teamwork, a computer-aided software-engineering tool, on Digital Equipment Corp.'s VAX workstation 2000s running DEC Windows to directly support MGS requirements engineering and analysis. Teamwork consists of tools that support the development life cycle in a completely integrated environment and provides facilities for model configuration management, document generation, and report generation.

We chose Teamwork over other CASE tools, such as Mark V Systems ObjectMaker, primarily because it provides a great deal of automated consistency checking between diagrams through a single data dictionary. It also directly supports Shlaer-Mellor graphics. It provides entity-relationship, control-flow and dataflow editors, which you can use to represent the information model, state-transition diagrams and

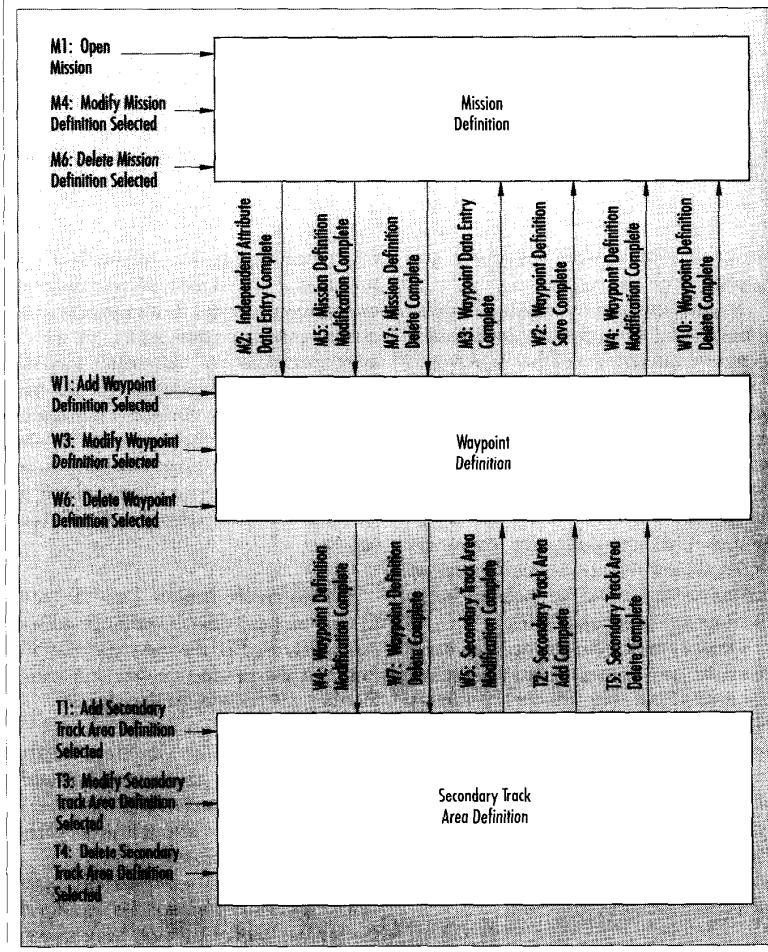


Figure 3. *Object-communication diagram for the Mission Generation System. Each object is shown as a rectangle; connecting arcs are labeled with communicating events.*

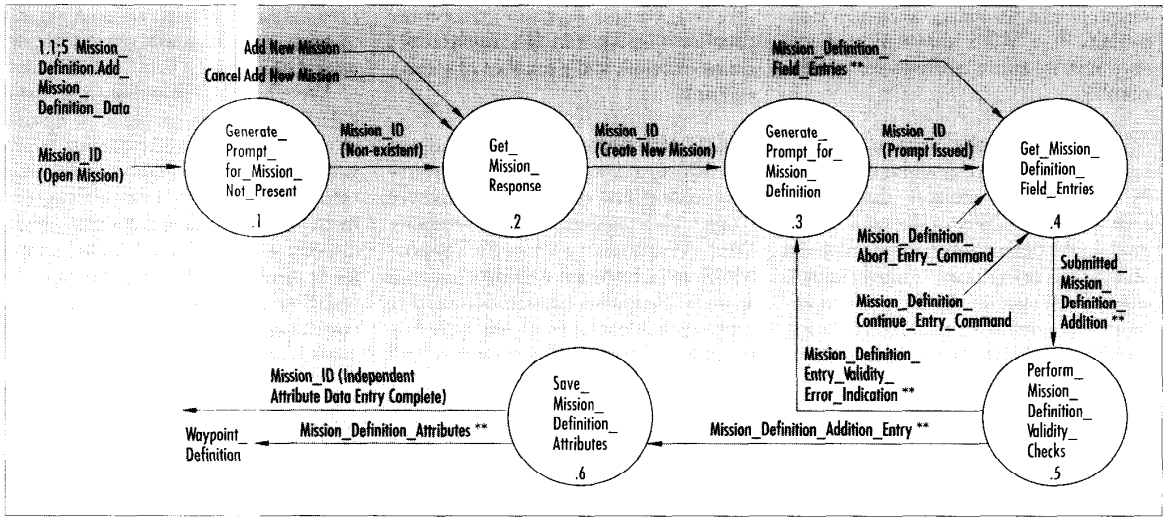


Figure 4. Dataflow diagram for the the mission object state "Enter Mission Definition Data State." Each circle in the diagram represents an action process of a given state. The dataflow diagrams contain processes (like Get_Mission_Response), data stores, dataflows (like Mission_ID, which creates a new mission), and a number of sources and sinks.

object-communication diagrams, and dataflow diagrams, respectively.

By coupling Teamwork's dataflow diagrams to its data dictionary, we were able to examine an element of the diagram and then expand that element to its dictionary parts. If something illegal is tried or an aspect of the system is missing, the user is warned. The elements can themselves are easy to examine, and you can quickly zoom from one level to another with the mouse. The data dictionary editor includes several very convenient functions such as circularity and redundancy checks and import/export capabilities.

Other advantages include very clean dataflow and state-transition diagrams and ease of learning, although it takes a while to grasp the interrelationships of menu hierarchies.

On the down side, Teamwork document generation requires painstaking organization and preparation of notes, and control and process specifications. It must also be supported by a publishing package such as Interleaf Inc.'s Interleaf.

LESSONS LEARNED

The Shlaer-Mellor method differs from functional-decomposition approaches, in which the analysis consists of examining processes. It also differs from event-response approaches, in which analysis consists of examining external events. Before you decide to use this method, you should consider a number of issues, including those specific to the method, those related to changing to an object-oriented culture, those related to training, and those related to CASE tool support.

Method-specific issues. The Shlaer-Mellor method offers significant advantages if it is used for the right type of application. It also produces better abstraction than any methods we have evaluated.

Object selection. The Schlaer-Mellor method provides a step-by-step approach to identifying objects. We observed a number of things about the way the method handles objects:

♦ *Object-selection techniques are sound.* Once the information model is developed, the Schlaer-Mellor method has concrete rules for grouping data and forming objects based on relational database theory. These rules eliminate many of the difficulties we have experienced with methods that begin with the definition of abstract objects. In these cases, we found it difficult to get every requirements analyst to understand the abstract objects in the same way. In the Schlaer-Mellor method, every analyst agreed on the objects, since the approach was based on concrete rules and not abstract thinking.

♦ *Object constraints and tests are rigorous.* The Schlaer-Mellor method imposes rigorous constraints on defining what constitutes an object. This forces you to do an in-depth analysis, which in turn aids in limiting the scope of the proposed system. It also helps you better define system boundaries. It provides a good starting point for further analysis by helping identify the objects about which the system must store data. It also offers a set of simple tests or criteria for normalizing the attributes of each object. The tests help ensure that each instance of an object will have one and only one value for each attribute identified, leading to a minimum of data redundancy and ambiguity.

♦ *The information model must be well developed.* A difficult but manageable problem with the Schlaer-Mellor method is the need to identify all or most data objects in

the first phase. Other object oriented methods let you create data objects as you recursively progress through requirements analysis. With the Schlaer-Mellor method, you must start with a well-developed information model, which is not always possible, especially in high-technology projects, which generally have poorly understood requirements and data objects.

♦ *Attribute description is insufficient.* The attributes used to define objects are descriptive names (like Launch_Point_Latitude). There are a number of drawbacks to this type of description. First, there is no way to express attribute values in actual instances of objects, which might be supplied by default, might be dependent on other attribute values, or might have to be set explicitly. Second, there is no way to formalize the conditions that some or all object instances may require. Finally, many existing information models emphasize the significance of an attribute through its features.

♦ *Support for low-level objects is limited.* The information model provides little support of unstructured objects. These objects or data types are primitives not constructed by aggregating lower level types like strings, integers, and reals. Aggregation provides a means for specifying the attributes of a new object type. Semantic data modeling permits the aggregation of entity types to form higher order entities (objects).

Application. The Shlaer-Mellor method includes a well-defined and detailed approach to object-oriented analysis that emphasizes up-front system development. We found that, in the long run, the extra effort during requirements analysis pays off in less effort during system design and implementation. Unfortunately, the flip side is that analysts can easily become bogged down in system analysis and lose sight of the ultimate goal.

The Schlaer-Mellor method is best applied to information systems or to reengineering situations, in which data objects are already identified.

Another important application issue is that the method supports bottom-up development. Some other methods, like the Colbert method, support top-down development. A thorough understanding of the individual data items at the start of the requirements process may yield a design that is easier to package, especially if the system is data intensive. Of course, in some applications, you cannot know detailed data items this early. In our hardware-in-the-loop simulation, for example, we did not know most of the low-level data items, and requirements were not complete. In applications like this, the Shlaer-Mellor method would not be appropriate.

We observed several application-related issues:

♦ *Abstracts are well packaged.* The Schlaer-Mellor method consistently produced better abstract objects than the other object-oriented methods we have evaluated and used. Like other methods, this method packages data objects and operations according to its rules. However, unlike other methods, as analysis progressed, we found no good reason to improve the objects by repackaging the operations and data objects differently. With other methods, we often discovered better packaging alternatives after further analysis. In many of these cases, the analysis had progressed so far that we did not feel it was cost-effective to go back and repackage the abstract objects, so these difficulties were never fixed. The abstract objects produced by the Schlaer-Mellor method are clearly superior to most other object-oriented requirements-analysis methods.

THE METHOD WORKS BEST WHEN YOU KNOW THE DATA OBJECTS.

♦ *Both single and multiple inheritance are supported.* Inheritance is a technique for sharing information among objects linked in the object hierarchy. Multiple inheritance lets objects inherit properties from multiple higher level objects in a specialized hierarchy. The Shlaer-Mellor information model supports single and multiple inheritance, and the method supplies an inheritance diagram for depicting inheritance relationships.[2]

♦ *Examples given are stereotypical.* As with other object-oriented methods we have used, we found our application to be much more difficult than the examples presented by the authors. This method is a rigorous approach to building an information model, yet the guidelines Shlaer and Mellor provided for building state and process models did not seem well defined.[1] The examples presented for the state and process models were fairly simple and limited. In performing our analysis, we encountered situations not addressed in the book, and had to use our own interpretations.

♦ *There is a tendency toward the wicked problem.* The Shlaer-Mellor method does not prevent a solution-oriented model within the requirements (problem-oriented) model. The object state-transition diagrams show details that are actually more preliminary design than requirements. This may cause the specification to be intertwined with the design, the wicked problem described by C. Ramamoorthy[6]

♦ *Ways of specifying requirements are limited.* The Schlaer-Mellor method is geared toward specifying program objects and system objects (like a computer system) or system specifications.[7] This view ignores problem specification, which, in our view, is a serious omission. We believe that requirements analysis must deal with the issues of the problem to be solved, not with issues of how a system should be designed to solve that problem. The Schlaer-Mellor method also lacks the specification of nonfunctional requirements — system performance, reliability, and cost constraints[8,9] — and environmental requirements — those describing the environment or domain in which a system is to be specified.

Modeling and notation issues. Notations are simple, straightforward, and easily understood. Because the analyst is forced to define system objects in detail up front, the requirements-analysis products should be simple and well-organized and have a minimum of ambiguity and redundancy.

MISSION GENERATION SYSTEM

The Mission Generation System creates or modifies a mission file for a self-guided cruise missile. The mission file contains the preplanned flight path and specific in-route instructions for the missile. Before the missile is flown, a launch computer downloads the mission file into the guidance unit of the missile. The missile autopilot processes the file and directs the missile to the intended target.

The group responsible for flight software uses the system to generate missions for algorithm development and for the software's acceptance testing. Many missions need to be generated to thoroughly test all the required software capabilities.

The MGS operates in several modes. It supports mission data building, modification, and viewing operations. The system accepts mission data from several sources to modify previously developed missions. It provides several output formats that are compatible with existing guidance simulations. It displays individual mission parameters in engineering units, letting the operator add to or modify the data in a screen-edit mode. The user interface has pull-down menus for access to data-modification, -entry, and -viewing screens.

When the MGS is in the mission-data-building mode, the user can create a mission from scratch or use all or part of an existing mission in the database. The user provides certain parameters, such as waypoint locations and altitude profiles, which the MGS uses to generate the rest of the data, such as flight segment lengths and turn control parameters.

When the MGS is in either the mission-data-viewing or -modification mode, the user can examine and modify individual mission-data parameters on the screen. In the modification mode, the MGS recomputes all mission data according to changes that the user has input. The system displays a ground-track plot of the selected mission on a latitude/longitude grid, which the user can send to a printer.

On the down side, some of the guidelines are idealistic, and it is difficult to express semantics.

The graphics are easy to understand, being taken from the familiar dataflow, entity-relationship, and state-transition diagrams. Other approaches, like the Colbert method, provide new, unfamiliar graphical notations like object-interaction, object-hierarchy, and object-class diagrams.[3,4,9]

We observed a number of things about the method's modeling approach and its notation:

♦ *Data and process analysis are directly linked.* Because a process model in the form of a dataflow diagram is created for each state in the state model — which is in turn created from the information model — there is a reasonably direct link between data analysis and process analysis. The method also coordinates object life cycles. We discovered, for example, many cases in which an object at a particular life-cycle state would require a response from another object. The object-communication diagram attempts to depict these interactions.

♦ *Constraints can be specified.* To ensure the integrity of the information model, you must be able to specify insertion, deletion, and modification constraints. If objects are connected through relationships, any of these constraints on an object will affect the status of objects connected to it. The Shlaer-Mellor information model lets you specify these relationship semantics.

♦ *Guidelines are idealistic.* Shlaer and Mellor's guidelines for creating state and process models were nearly impossible to adhere to. They recommend you describe each process directly on the model (dataflow diagram) without providing any process descriptions, since that purpose was fulfilled by describing the actions. When we tried to do this, the dataflow diagram quickly became cluttered and unreadable. Rather than break down the offending processes further, we simply used Teamwork's process-specification feature, which let us declare the child of each process to be a process specification. In that way, we directly accessed the process description from the dataflow diagram and kept our process models readable.

Another guideline was equally difficult to follow: Aside from some systemwide data items, only attributes from the information model can appear on the dataflow. While the dataflow created on our process models, with a few exceptions, did consist of only attributes from the information model, some dataflow included enough pieces of data that labeling each one in this manner soon became impractical. Using the data dictionary feature on Teamwork, we abstracted the dataflows that proved cumbersome and marked the abstracted dataflows with a **. We could then easily access the contents of these flows from their respective process models.

♦ *State models have redundant processes.* We encountered some cases in which a particular state in one state model would involve the same set of action processes as another state in the same or a different model. Consequently, some process models were very similar, and the redundancy left us feeling as if our analysis was in some way incomplete even though we had adhered to the method as closely as possible. The Shlaer-Mellor method did not address this issue.

♦ *Representation views are limited.* One of the main reasons to use an object-oriented model of user requirements is to understand them better. Thus, you would like to read and manipulate specifications in a variety of representations. The same real-world entity can be described in many ways, from a high-level, black-box template to a low-level, detailed one. In the Shlaer-Mellor method, requirements specifications represent only a single view of the world.

♦ *Expressing semantics is difficult.* Although the information model lets you select an expressive description of relationships, there is a danger of choosing words poorly and making it harder to understand the model. It is a significant challenge to select just the right words to resolve vary-

ing points of view and to avoid ambiguity. The method offers no guidance for selecting relationship semantics so as to avoid overloading words.

♦ *Interaction description is insufficient.* Although the state model describes each object's dynamic behavior separately through a state-transition diagram, it does not develop complete descriptions of system-object interactions. In addition, the model's external states correspond one to one with externally recognizable object states, which can cause an infinite number of iterations in the model if the requirements are not complete.

♦ *The process model relies on dataflow diagrams.* The process model's reliance on dataflow diagrams is a mixed blessing. Object interaction is described indirectly via the occurrence of dataflow. Processes that result from decomposing actions into dataflow are usually highly cohesive. However, dataflow diagrams themselves have some drawbacks. They are not very helpful for systems or parts of systems that primarily update and retrieve data. They make it difficult to identify transformations on data and partitioning a transition into I/O branches can be artificial.[8] Finally, the transition from analysis to design can be challenging.

Cultural-change issues. As with any new method, there are apt to be changes in fundamental ways of thinking about the problem and how to transfer that new way of thinking to the customer.

To deliver a quality object-oriented product, project members must think and design in terms of objects. This means the focus is on the object as the unit of design. The history of our team was to localize on functionality — the system is partitioned along functions — not on objects — all the information pertaining to an object, or class of objects, is grouped in one part of the system. In Figure 1, for example, all the information required by the mission-definition object would be localized in one part of the system. This is quite a change of thinking, a change in the way of doing things, and a change in conceptual learning.

Customers must also fully understand the object-oriented method and the tools used to develop and represent the system. They must be fully aware of the impact of object-oriented analysis and design on the entire life cycle. Compared to structured analysis and design, a typical object-oriented analysis and design method produces software documents that describe requirements in terms of graphical representations rather than text and may require unique tailoring of standards. Customers must understand the underlying concepts to adequately review these documents and support software reviews.[8,9]

Training issues. The training and education of the project team play a major role in improving object-oriented development. However, the curriculum should not focus on object-oriented analysis and design alone. The team needs to see how these concepts affect subsequent development stages as well.

A good way to do this is to emphasize the need to make requirements traceable through the development life cycle. You can use a requirements and design case study of a system similar in size and complexity to the target system. The study should show you problems that might arise during requirements analysis and design. You should also make this training available to customers if they will be evaluating the requirements specification and software design documents and participating in various reviews.[8,9]

IT TOOK ABOUT TWO MONTHS TO BECOME COMFORTABLE WITH THE METHOD.

The learning curve is also a factor. We did not have as much of a problem with this because we all had experience in applying an object-oriented method. However, we were hampered by a general lack of familiarity with the Schlaer-Mellor method and the MGS domain. It took us about two months to become comfortable with the method. If your team has no experience with object-oriented methods, you can expect a learning curve of between two and four months.

Finally, for a large system, you may need to add a team member, a chief methodologist, to ensure that the methods and tools are rigorously followed and to provide follow-up training to the development team when needed.

CASE tool support. Making sure that you have good CASE tools should be a prerequisite to any object-oriented development effort. An effective CASE tool is important throughout the development life cycle:[8,9]

♦ System engineers can quickly develop requirements descriptions and easily maintain them.

♦ Designers can express their designs in a format compatible with the chosen method.

♦ The tool provides traceability from requirements through design and code.

♦ In addition to satisfying contractual requirements, the requirements and design documents can serve as working papers during development.

Bear in mind that you are not likely to find the perfect tool. Object-oriented analysis and design methods are changing rapidly, and CASE tool vendors are hard pressed to keep pace. They seem always to be one or two releases behind. However, the following selection guidelines might be helpful:

♦ Choose a CASE tool that supports your selected method and that runs on the required platform in an organized manner. We recommend as a first step to select an object-oriented analysis and development method and then find a CASE tool that supports it and runs on your equipment. If you choose a tool before the method, you'll have less flexibility to tailor the method to meet your needs. It is important select the method and tool hand in hand. If you don't, the method you choose may lack adequate CASE tool support, or the tools you select to support the method may not meet other project requirements. If the latter situation arises, you should continue the evaluation until you find an optimal method-CASE tool combination.[8,9]

Overall, our use of the Schlaer-Mellor method was successful. We felt especially comfortable with the objects defined using object-oriented analysis because object selection was governed by stringent rules. Although the Ada design for the MGS has not been completed, we feel that the objects defined when we built the information model are well established and will not change.

Our success was largely due to our previous experience with object-oriented techniques, a CASE tool that directly supported our method and provided consistency checking, and, most important, team members who recognized the benefits of new techniques and were willing to struggle through the learning process. ♦

ACKNOWLEDGMENTS

We thank the guidance software group at McDonnell Douglas's Missile Systems Division for their support of this work. We also thank Thomas Hill and Constance Palmer of McDonnell Douglas Corp., Wei-Tek Tsai of the University of Minnesota and Dennis de Champeaux of Hewlett-Packard laboratories for their valuable comments and suggestions on a draft of this article. We also acknowledge the valuable suggestions of anonymous *IEEE Software* referees.

REFERENCES

1. S. Shlaer and S. Mellor, *Object-Oriented System Analysis: Modeling the World in Data*, Yourdon Press Computing Series, Prentice-Hall, Englewood Cliffs, N.J., 1988.
2. S. Shlaer and S.J. Mellor, *Object Lifecycles: Modeling the World in State*, Yourdon Press Computing Series, Prentice-Hall, Englewood Cliffs, N.J., 1992.
3. E. Colbert, "The Object-Oriented Software Development Method: A Practical Approach to Object-Oriented Development," *Proc. TRI-Ada*, ACM Press, New York, 1989, pp. 400-415.
4. M. Fayad et al., "Hardware-In-the-Loop (HIL) Simulation: An Application of Colbert's Object-Oriented Software Development Method," *Proc. TRI-Ada*, ACM Press, New York, 1992, pp. 176-188.
5. E. Moore, "Gedanken Experiments on Sequential Machines," in *Automata Studies*, Princeton University Press, Princeton, N.J., 1956, pp. 129-153.
6. C. Ramamoorthy et al., "Software Engineering: Problems and Perspectives," *Computer*, Oct. 1984, pp. 191-209.
7. I. Zaulkernan and W. Tsai, "Are Knowledge Representations the Answer to Requirement Analysis?" *Proc. IEEE Conf. Computer Languages*, IEEE Press, New York, 1988, pp. 437-443.
8. M. Fayad and D. de Champeaux, "Object-Oriented Experiences," *Proc. TRI-Ada: Volume 1*, ACM Press, New York, 1992, pp. 380-496.
9. M. Fayad and L. Hawn, "Object-Oriented Software Engineering: Techniques and Tools for a Better Software Development," *Proc. Digital Avionics Systems Conf.*, IEEE Press, New York, 1992, p. vii.

Mohamed E. Fayad is the founder and chief executive officer of Object Technologies, Inc., an object-oriented training, research and development, and publishing firm. His technical interests include object-oriented software engineering, methods and techniques, and languages; real-time systems development, verification and validation; software testing; data modeling; and software-engineering processes. He did the work described in this article when he was a principal specialist in engineering at McDonnell Douglas Corp. He has written numerous technical papers and tutorials on object-oriented software engineering, experiences, and techniques.

Fayad received a BS in science from Cairo University and an MS in computer science from the University of Minnesota at Minneapolis, where he is pursuing a PhD in computer science. He is a member of the IEEE, IEEE Computer Society, and ACM and is a senior member of the American Institute of Aeronautics and Astronautics.

Louis J. Hawn is a senior software-engineering manager at McDonnell Douglas Corp., where he has pioneered the company's use of object-oriented development methods and is responsible for training in object-oriented software development. His technical interests are object-oriented software engineering real-time software and software management

Hawn received a BS in computer science from Michigan State University. He is a member of the IEEE, IEEE Computer Society, and ACM.

Mark A. Roberts is a software group manager at McDonnell Douglas Corp. His technical interests are real-time software, Ada, and object-oriented techniques.

Roberts received a BS in electrical engineering from Christian Brothers University in Memphis and an MS in engineering management from Washington University at Saint Louis. He is a member of the IEEE, IEEE Computer Society, and ACM.

Jerry R. Klatt when this article was written, he was a programmer/analyst at Computerized Business Systems. He is a senior specialist in engineering at McDonnell Douglas Corp. His technical interests are object-oriented methods, CASE tools, database/development, Ada and C++.

Klatt received a BS in electrical engineering from the University of Missouri at Columbia, a graduate certificate in information management from Washington University at St. Louis and is currently toward a masters in that field. He is a member of the IEEE, IEEE Computer Society, and ACM.

Address questions about this article to Fayad at Object Technologies, Inc., PO Box 410857, St. Louis, MO 63141; Internet 75000.1021@compuserve.com.