A Comparison of the Booch Method and Shlaer-Mellor OOA/RD

Stephen J. Mellor

Project Technology, Inc.
7400 N. Oracle Rd., Suite 365
Tucson Arizona 85704
520 544-2881
http://www.projtech.com

2 May 1993

The purpose of this paper is to compare Shlaer and Mellor's OOA/RD with the Booch Method, as published in *Object-Oriented Design with Applications*, Benjamin Cummings Publishing Company, 1991, and extensions to the notation published in two installments of *Computer Language* Magazine in 1992.

It is assumed that the reader has a broad familiarity with both methods.

1. Comparison Summary

There are three fundamental differences between the Booch and Shlaer-Mellor methods: the underlying software development processes, levels of abstraction, and approaches to design.

Process: The two methods represent fundamentally different processes. Booch defines four steps, but none of these are connected to a particular set of models. Instead, the models are built incrementally and iteratively as the developers go through the steps. Completeness criteria are not defined for the models, nor is there a way to know how much iteration there will be. Each iteration may cover large parts of the lifecycle, and require reconstruction of large portions of the model.

Booch describes the process as "round-trip iterative gestalt design." The intent is to iterate through the several layers of abstraction until the entire system is understood. At each layer, several models are built. There are no rules on the order of creation of the models, nor are there rules on how much information should appear on a given model. There are recommendations for the order of the work (as a way of getting started), but if problems lead the developer to want to look at a different aspect of the system, then the developer is encouraged to go ahead and do so. There are no completeness criteria for the models stated in the method. Information is collected in each model sufficient to drive the next step. The models are built as an organic whole, and the models are not complete until they are all complete.

This lack of a process with well-defined entry and exit criteria for each step means that on the SEI maturity model scale, Booch can support only a level 1 to level 2 process.

Shlaer-Mellor is, by contrast, a much more strict method, and we use models in a more formal manner. The models form a single integrated unit with strict rules about how to put these (predefined) elements together. Iteration is limited, by design of the method: the method limits iteration to a single domain at a time. Rules for completion are defined for each model. Models are tightly connected and must be created in a specific order. The integration rules tend to focus attention on a single model at once, though other models may be begun if the information is available. For each model, there are rules that define when the model is complete. The purpose of each model is to capture all the information explicitly, since that information will be *translated* into the code.

The Shlaer-Mellor Method can support an SEI maturity level of 3 to 4 because of the emphasis on process and on entry and exit criteria for the models.

The net effect is that Booch is a much looser method, and Shlaer-Mellor has much more formality and guidelines about how to build the models.

Levels of Abstraction/Domains: Booch discusses building the design models at several "levels of abstraction," where objects in one level are used to construct objects at other levels. As the designer's understanding of a problem increases, this leads to discovery of deeper layers of abstraction which are then incorporated into the model. In contrast, Shlaer-Mellor begins by identifying the domains (distinctly different subject matters) in the problem, and then building a separate analysis for each domain.

This difference has strong implications for reuse. Complete subject matters can be reused in their entirety. But if a "level of abstraction" mixes several subject matters, then reuse can be achieved only at the component level (a class or a set of classes).

In Booch, a system can be broken down into parts and each part can be worked on separately. This can be viewed as a vertical partitioning of the system in which each vertical partition incorporates several layers of abstraction. In Shlaer-Mellor, however, the system is first partitioned horizontally (into domains), then each domain can be partitioned vertically (into subsystems). This two-way partitioning strongly supports concurrent development.

Design Approaches: The two methods take very different approaches to design. Booch describes the place of design as follows: "Design can start as soon as we have some (possibly incomplete) formal or informal models of the problem to be solved." However, nowhere in the book does he build a separate model of the problem as an input to the design process, nor does he demonstrate any method to get from such a model to a design. He lists several structured analysis approaches and object-oriented approaches to analysis, including Shlaer-Mellor OOA, as potential front-ends to object-oriented design.

Shlaer-Mellor, on the other hand, views design as a *translation* process. Shlaer-Mellor builds an implementation-free formal model of the application, and an application-free formal model of the system architecture. Both models can be formally verified, and the processes of building these models have formal entry and exit criteria. The application model is then systematically translated according to the rules of the architecture. This translation process can be automated.

One effect of these differences is that the cost of building a design in Booch (and in other object-oriented and structured methods) is proportional to the size of the analysis. In Shlaer-Mellor, there is a fixed cost associated with defining the system architecture and the translation rules, regardless of the size of the system. While both methods work for both large and small systems, the Shlaer-Mellor design approach scales up easily to large systems, and to sets of closely-related systems with significant degrees of reuse.

There is a defined approach for moving from analysis to design in Shlaer-Mellor.

2. Philosophy and Goals

Significant philosophical differences between the two methods make a point-by-point direct comparison of the notations a rather fruitless exercise.

Booch uses models to help visualize, and reason about, important aspects of a system in an *ad hoc manner.* The models are generally heavy with "adornments;" the developer is encouraged to annotate the models, and to add further adornments as desired. Booch's philosophy does not require completeness, nor absolute consistency, but instead uses the models as a basis for adding detail until the system is

completed. There is one set of models for the system as a whole, but relatively few rules about how to integrate the several views into a consistent whole. Booch finds a rich notation the most compact way of representing the rich set of concepts in the object model.

In contrast, the Shlaer-Mellor approach to design relies on translation of the models, so completeness and consistency are paramount. The Shlaer-Mellor Method prefers simple models, with minimum orthogonal constructs. While it is certainly possible to define a notation that has specific notations for certain common elements of object-oriented designs, a separate notation for every concept over time will lead to a very complex notation with many components.

The implications of these philosophies, as well as the summary concepts from the previous section can be summarized in the table below.

Goals of the Methods

	Shlaer-Mellor Method	**Booch Method**
Readability	Sparse notation	Notation heavy with adornments
Consistency of models	Many rules given by method	Very few rules given by method
Capture of information	Capture all information explicitly	Capture enough information to drive the next step
Completeness of models	Defined by method	Not defined
Analysis to design transition	Analysis models are mathematically translated into design.	Elements of the object model are incorporated into the logical models. The physical model is a repackaging of the logical model.

In summary, Booch uses notation in the sense of an artist's rendition, whereas Shlaer-Mellor uses notation in the sense of a blueprint.

3. Detailed Comparison

3.1. System Partitioning Notations

The following table compares the notations of Shlaer-Mellor and Booch for system partitioning.

Shlaer-Mellor Concept	**Booch Concept**
Domain	Not modeled explicitly (cf. levels of abstraction)
Subsystem	Class Category & Subsystem
Project Matrix	No concept

Shlaer-Mellor distinguishes between subject matters, and then analyzes each one separately. The different domains are linked by building mappings between them. Booch also distinguishes between levels of abstraction, but he does not keep them separate.

Both Booch and Shlaer-Mellor have the concept of dividing the system into manageable units. The class category is a group of classes that can be understood a as unit (cf. subsystem in OOA). The subsystem in Booch is a physical packaging of classes.

Booch has no notation for the management structure that corresponds to the Project Matrix.

3.2. Comparison: Analysis Notations

The following table compares Shlaer-Mellor and Booch's Logical Models for (loosely) analysis.

Shlaer-Mellor Concept	**Booch Concept**
Object Information Models	Class and Object Diagrams
Object and Attribute Descriptions	Class Templates
Relationship Descriptions	No concept
Object Communication Model	Object Diagrams
Object Access Model	Object Diagrams
State Process Table	Operations on a Class Diagram
State Model	State Model
Action Data Flow Diagram	Operations on a Class Diagram
Process Descriptions	Operation Templates
Thread of Control	Timing Diagram

Object Information Models vs. Class and Object Diagrams: The intent of the Class Diagram is to declare the abstractions in the problem, just like the Object Information Model. The relationships on the Class Diagram come from a predefined set (Part-Of, Is-A) that are strongly related to implementation. The Object Information Model, on the other hand, defines relationships from the domain at hand: Train RUNS ON Tracks; Train HAS Doors, etc. The Object Diagram shows visibility and messages between objects. In Shlaer-Mellor OOA, communications between objects are distinct from relationships between objects, and are represented on a different Object Communication Model. The Shlaer-Mellor Object Information Model declares conceptual entities in terms of typical, unspecified instances, so there is only one model, instead of one for classes and one for objects.

Object and Attribute Descriptions vs. Class Templates: The intent of both models is to define the meaning of the class. Booch does not define the meaning of the data of the class; the Shlaer-Mellor model defines both objects and attributes. The class template also defines some implementation characteristics of the class, such as visibility, concurrency and persistence.

Relationship Descriptions: In Booch, relationships are defined from a set of design relationships such as Part-Of, Is-A and the like. Booch defines the meaning of these relationship (Part-Of etc.) in the Concepts section of the book. Shlaer-Mellor abstracts relationships from the application, and does not predefine the relationships. For example Dog Owner OWNS Dogs, Cat IS INFESTED WITH Fleas.

Object Communication Model vs. Object Diagrams: Booch's object diagram shows communication between object instances as messages. Message may be simple, synchronous, balking, timeout asynchronous. The Shlaer-Mellor OCM shows communication of events between state models. Events are presumed to be asynchronous. Note that at implementation time, the events may be implemented in any manner desired by the designer.

Object Access Model vs. Object Diagrams: Booch's object diagram shows communication between object instances as messages. Message may be simple, synchronous, balking, timeout asynchronous. The Shlaer-Mellor OAM shows access from one object to another. Access is presumed to be

synchronous. Note that at implementation time, the accesses may be implemented in any manner desired by the designer.

State Process Table vs. Operations on a Class Diagram: The State Process Table of Shlaer-Mellor lists all the operations (processes) in the system, including both where the operation is used and the class in which the operation will be defined at implementation time. Booch's class diagram defines the operations of a class.

State Models: Booch's book does not fully define the state model, and only describes it briefly. There is no discussion of communication of events between state models. The Shlaer-Mellor Method relies heavily on state models to model the dyanmic behavior of each object instance. State machines communicate with each other by sending events back and forth to synchronize behavior.

Action Data Flow Diagram vs. Operations on a Class Diagram: Booch's class diagram defines the operations of a class. Less detail is shown than on the Action Data Flow Diagram (ADFD). The ADFD of the Shlaer-Mellor Method shows the functions required in each state in detail, including sequence, conditionality and iteration. There is sufficient information of the ADFD to move directly into code.

Process Descriptions vs. Operation Templates: The operation template is used to describe operations of classes, including the meaning of the operation, its formal parameters, exceptions and concurrency. The process descriptions of Shlaer-Mellor describe only the meaning of the process, its inputs and outputs.

Thread of Control vs. Timing Diagram: Booch's timing diagram shows the transmission of events between object instances. Shlaer-Mellor's thread of control diagram does the same except that it also shows actions within object instances, as well as contention for resources.

3.3. Comparison: Design Notations

It is particularly difficult to provide a detailed comparison in this area because the goals of the methods are so different.

In Booch, the physical models show the allocation of the logical elements of the models to their physical locations: Processors, Tasks, Packages etc. In addition, the logical models show the results of a variety of implementation decisions: What is the inheritance hierarchy? Who calls whom? What are the interfaces to the operations of classes?

Shlaer-Mellor takes a unique perspective on the depiction of the elements of software that make up a system. The software architecture, in Shlaer-Mellor, is seen as a completely separate subject-matter that can be understood independently of any application. The concepts of lists, tasking paradigms, classes and inheritance, are simply concepts that need to be understood. This means that we could use OOA to model the software architecture, and define exactly (using OOA) what we mean by each one of these "design" concepts. This is the same as using OOA to extract and define the concepts defined in Booch's Object Model.

Alternatively, it is possible to define a notation specifically for object-oriented designs. This has been done with the OODLE notation. OODLE is not a fundamental part of Shlaer-Mellor Recursive Design, but OODLE has notational elements for some important parts of the Object Model. OODLE is deliberately restricted. It covers classes, their internal design, inheritance, and dependencies between classes. It does not cover tasking models: As Booch himself observes "Concurrency is orthogonal to object-oriented design." This approach was taken to the definition of OODLE to avoid building a notation for each and every concept of object-oriented architectures.

We have chosen to build a table that shows the correspondence between OODLE and Booch's notation. Booch also has notations for visibility that appear on the Object Diagram, and for tasking, packaging, and generics on the Module Diagram. There are no corresponding notations for these concepts in OODLE.

Shlaer-Mellor OODLE Concept	**Booch Concept**
Class Diagram	A class appears on a class diagram
Operations for a class	Operations of a class appear on a class diagram
Statically bound operation	No notation
Dynamically bound operation	No notation
Deferring operation	No notation
Deferred operation	No notation
Parameters of an operation	Operation template for a class.
Internal structure of a class	Not shown
Inheritance Structure	Class diagram relationships
Dependency Structure	Object Diagram messages
class/module	Class/class utility (a packaging of modules)
client-server/friend	No notation

Note that this comparison does not include the many notations and adornments provided by Booch for specific elements of object-oriented designs. For a complete list, see Booch's book and subsequent articles on the notation in *Computer Language* Magazine.

3.4. The Process

Booch Process: Booch describes his method as "round-trip iterative gestalt design." He then defines four steps to the method:

- Identify the classes and objects at a given level of abstraction
- Identify the semantics of the objects and classes
- Identify the relationships among these classes and objects
- Implement these classes and objects

These four steps are carried out iteratively, and the order of the four steps is not absolute. Each of the four steps is defined in terms of activities and work products, as follows.

Booch Process:

Step	Activities	Products
Identify the classes and objects at a given level of abstraction	Discover key abstractions Invent important mechanisms	List of things — OR — Formal specification of classes and objects with templates May include any of the diagrams
Identify the semantics of these classes and objects	Identify the semantics from perspective of interfaces	Refinements of the templates Document the static and dynamic semantics Build object diagrams for new objects Prototype parts of the design
Identify the relationships among these classes and objects	Discover patterns to help reorganize the objects and classes Make visibility decisions	Build in relationships between classes and objects Module diagrams Prototypes
Implement these classes and objects	Decide on representation of the classes and objects Allocate classes and objects to modules and programs to processors	Finish templates

In reading this table, note that the work products vary in the level of completeness and in the level of detail. For example, the work product for "Identify the classes and objects at a given level of abstraction" could be a list of things, or a formal specification of the objects.

Shlaer-Mellor Process: The steps, activities, and products in Shlaer-Mellor are summarized in the following table. In the general case, there may be several domains that require analysis. Each of the domains is analyzed separately, then mappings are built between the domains.

Shlaer-Mellor Process:

Step	Activities	Products
Divide the system into subject matters	Identify the domains in the problem.	Domain Chart
	Identify the client and server relationships between the domains (the bridges).	Mission statements for the domains. Bridge descriptions for the bridges.
OOA for each domain	Identify the objects, attributes and relationships between objects in the domain.	Object Information Model with object, attribute and relationship descriptions
	Define the dynamic behavior for each object and relationship in the problem .	State models for each object and relationship. Object Communication Model
	Define the functions for each object, and their callers.	Action Data Flow Diagram per state of the state model. State Process Table for the whole model. Process Descriptions for each process (if required).
Software Architecture	Identify mechanisms of the architecture. Define archetypes. Define rules for mapping from the application to the architecture.	OODLE diagrams: Class Diagram Class Structure Chart Inheritance Diagram Dependency Diagram Templates

Summary

There are three fundamental differences between the Booch and Shlaer-Mellor methods: the underlying software development processes, levels of abstraction, and approaches to design.

Booch uses models to help visualize, and reason about, important aspects of a system in an *ad hoc manner*. The models are generally heavy with "adornments;" the developer is encouraged to annotate the models, and to add further adornments as desired. Booch's philosophy does not require completeness, nor absolute consistency, but instead uses the models as a basis for adding detail until the system is completed. There is one set of models for the system as a whole, but relatively few rules about how to integrate the several views into a consistent whole. Booch finds a rich notation the most compact way of representing the rich set of concepts in the object model.

In contrast, the Shlaer-Mellor approach to design relies on translation of the models, so completeness and consistency are paramount. The Shlaer-Mellor Method prefers simple models, with minimum orthogonal constructs. While it is certainly possible to define a notation that has specific notations for certain common elements of object-oriented designs, a separate notation for every concept over time will lead to a very complex notation with many components. In summary, Booch uses notation in the sense of an artist's rendition, whereas Shlaer-Mellor uses notation in the sense of a blueprint.