

Reference Manual
Version 0.8

Unified Method

Grady Booch
James Rumbaugh

RATIONAL
SOFTWARE CORPORATION

Copyright ©1995 by Rational Software Corporation.

[This document is incomplete and still under development.]

1. Introduction

1.1 Background

The Unified Method is a third generation method for specifying, visualizing, and documenting the artifacts of an object-oriented system under development. The Unified Method represents the unification of the Booch and OMT methods, and additionally incorporates ideas from a number of other methodologists, most notably Jacobson, Wirfs-Brock, Ward, Cunningham, Rubin, Harel, Gamma, Vlissides, Helm, Johnson, Meyer, Odell, Embley, Coleman, Coad, Yourdon, Shlaer, and Mellor. The Unified Method is the direct and upward-compatible successor to both Booch and OMT. By unifying these two leading object-oriented methods, the Unified Method provides the basis for a *de facto* standard in the domain of object-oriented analysis and design.

Identifiable object-oriented methods first appeared in the late 1980's. In the years following - characteristic of every emerging discipline - there was an explosion of object-oriented methods as various methodologists experimented with different approaches to object-oriented analysis and design. Experience with a number of these methods grew, accompanied by a growing maturation of the field as a whole as more and more projects applied these ideas to the development of production-quality mission-critical systems. By the mid 1990's a few second generation methods began to appear, most notably Booch '94, the continued evolution of OMT, and Fusion.

Given that the Booch and OMT methods were already independently growing together and were collectively recognized as the dominant methods world-wide, Booch and Rumbaugh joined forces in October 1994 to forge a complete unification of their work. This document and its companion *User Guide* represent the culmination of that effort.

1.2 Goals and Guiding Principles

Devising a notation for use in object-oriented analysis and design is not unlike designing a programming language. First, the author must bound the problem: should the notation encompass requirements specification? Should the notation extend to the level of a visual programming language? Second, the author must strike a balance between expressiveness and simplicity: too simple a notation will limit the breadth of problems that can be solved; too complex a notation will overwhelm the mortal developer. In the case of unifying existing methods, the author must also be sensitive to the installed base: make too many changes, and you will confuse existing users; resist advancing the notation, and you will miss the opportunity of engaging a much broader set of users.

As the primary authors of the Booch and OMT methods, we were motivated to forge a unified method for three simple reasons. First, the Booch and OMT methods were already evolving toward each other independently, and it made sense to continue that evolution together rather than apart, thus eliminating the potential for any unnecessary and gratuitous differences that would further confuse users. Second, by unifying these methods now, we could bring some stability to the object-oriented marketplace by ending the method wars, thus allowing projects to settle on one mature method and letting tool builders focus on delivering more useful features. Third, we expected that our collaboration would yield improvements in both earlier methods, helping us to

capture lessons learned (such as the application of use cases) and to address problems that neither method currently handled well (such as the modeling of distributed systems).

As we began our unification, we established four goals to help bound our efforts. Specifically, the goals of the unification effort were

- To model systems (and not just software) using object-oriented concepts.
- To establish an explicit coupling to conceptual as well as executable artifacts. Thus, end users, domain experts, and analysts should be able to use this method, as well as developers. Furthermore, the models generated under this method should be sufficient for a skilled craftsman or a tool to produce an executable artifact, as well as to reverse-engineer executable artifacts back into conceptual models.
- To address the issues of scale inherent in complex, mission-critical systems.
- To be usable by both humans and machines.

We began with the creation of a metamodel (described in detail in Appendix A), so that we would first have a common and unambiguous statement of the method's semantic model, important to both ourselves as well as to tool builders. Once this metamodel was in place, we could then decide upon the cosmetic graphical syntax for the visual elements of this semantic model. To that end, we established a number of principles to guide our efforts:

- **Simplicity** The method should require only a few concepts and symbols.
- **Expressiveness** The method should be applicable to a wide spectrum of production-quality systems.
- **Usefulness** Not all things that are possible are useful; the method should focus only upon those elements that are meaningful to practical system and software engineering.
- **Prioritized** Common problems should be simple to model; rare or fringe problems should still be expressible, but may require some complexity, commensurate with their frequency of use.
- **Self-consistency** The same concept and symbol should be applied in the same fashion throughout the method.
- **Orthogonality** Independent concepts should be modeled independently.
- **Layered** Advanced concepts should be treated as additions to the method for more basic concepts.
- **Stable** Adopt concepts and symbols that are already commonly understood and used.
- **Printable** The method should be amenable to both hand-drawn

sketches on napkins and whiteboards as well as black and white printed images. This principle does not prevent tools from exploiting color, animation, and other more advanced approaches to visualization but requires a base format that can be extended.

- Extensible

Users and tool builders should have some degrees of freedom to extend and adapt the method.

Devising any real method requires a balance among each of these principles.

1.3 Organization

This manual provides a complete reference to the Unified Method for the expert user and tool builder. It is not meant to serve as a means of teaching the method to developers, although its companion *User Guide* provides a number of such practical lessons. Rather, this *Reference Manual* serves as a definitive statement of the Unified Method semantic model and notation. The *Reference Manual* and companion *User Guide* do not address the issue of process. We anticipate a number of other papers and books to appear that will provide vehicles for teaching and using Unified Method for the development of systems in a variety of problem domains.

Unified Method's semantic model and notation are themselves layered, and so this manual describes Unified Method according to these layers. Section 2, *Basic Concepts*, presents Unified Method's lexical elements, including the semantics of names, uninterpreted elements, and other primitives. This section also addresses a few fundamental concepts, such as the semantics of views versus models, scope and visibility, and the semantics of properties.

Section 3, *Core Modeling Concepts*, presents the essential elements necessary for modeling the logical architecture of a system, encompassing class diagrams, state machine diagrams, object instance diagrams, and scenario diagrams (encompassing both object message diagrams and message trace diagrams). Section 4, *Advanced Modeling Concepts*, builds upon the previous section, and provides the syntax and semantics of the layered elements that address issues of scale, use cases (which extend the notation to requirements capture) and detailed design features (which help bolt the artifacts of Unified Method to executable programming languages).

The remaining three sections address the other nearly-independent views of an object-oriented system under development. Section 5, *Physical Modeling*, presents the syntax and semantics of those elements necessary for the development view of a system, encompassing modules and subsystems. Section 6, *Process Modeling*, presents those elements essential for modeling concurrent systems, including threads, processes, communication, synchronization primitives, and timing considerations. Section 7, *System Modeling*, presents the syntax and semantics of those elements essential for modeling distributed systems, including processors, devices, connections, and distribution units.

This manual concludes with three appendices. Appendix A, *Unified Method Metamodel*, provides a concise and complete description of Unified Method's semantic model, illustrated both in text as well as in the Unified Method notation itself. Appendix B, *Unified Method Notation Summary* provides a summary of Unified Method's graphical syntax. Appendix C, *Mapping from Booch and OMT to Unified Method*, illustrates how Unified Method has evolved from both Booch and OMT as well as how elements from Booch and OMT map to Unified Method and vice versa.

This document concludes with extensive index to aid in navigating among the various elements of Unified Method.

1.4 Conventions

The explanation of each element in Unified Method follows a simple pattern. First, we define the concept itself in text. Second, we present its semantics, followed by its graphical syntax where applicable. Third, we provide some commentary, in which we explain the rationale and/or relevance of that element, a simple example of its use, an explanation of any recommended idioms, and finally hints to users and tool builders. To distinguish these various parts, we use the following typographic conventions:

Each part of these explanations are introduced by headings in italicized Avant Garde:

- *Definition, Semantics, Commentary*

Fundamental Unified Method elements are named in bold face Helvetica:

- **Class, Association, Subsystem**

The semantics of these elements are expressed in plain text (as in this sentence). *Recommended idioms and hints to tool builders are expressed in italicized text (as in this sentence).*

2. Basic Concepts

2.1 Views and Models

2.2 Lexical Elements

2.3 Scope and Visibility

2.4 Properties

3. Core Modeling Concepts

3.1 Class Modeling

3.2 State Machine Modeling

3.3 Object Instance Modeling

3.4 Object Message Modeling

3.5 Message Trace Modeling

4. Advanced Modeling Concepts

4.1 Categories and Composites

4.2 Use Cases and Scenarios

4.3 Class and Object Adornments

4.4 Non-Class Modeling

5. Physical Modeling

5.1 Modules

5.2 Subsystems

6. Process Modeling

6.1 Threads and Processes

6.2 Communication, Synchronization, and Timing

7. System Modeling

7.1 Devices, Processors, and Connections

7.2 Distribution Units

.

A. Unified Method Metamodel

[the final metamodel guide will be placed here]

B. Unified Method Notation Summary

[the final notation summary (for the Unified Method only) will be placed here]

C. Mapping from Booch and OMT to Unified Method

[the final notation summary (for the mapping of Booch and OMT to the Unified Method only) will be placed here]

D. Index