Preface

0.1 About the Unified Modeling Language (UML)

The Unified Modeling Language (UML) provides system architects working on Object Analysis and Design with one consistent language for specifying, visualizing, constructing, and documenting the artifacts of software systems, as well as for business modeling.

This specification represents the state-of-the-art convergence of best practices in the object-technology industry. UML is the proper successor to the object modeling languages of three previously leading object-oriented methods (Booch, OMT, and OOSE). The UML is the union of these modeling languages and more, since it includes additional expressiveness to handle modeling problems that these methods did not fully address.

One of the primary goals of UML is to advance the state of the industry by enabling OO visual modeling tool interoperability. However, in order to enable meaningful exchange of model information between tools, agreement on semantics and notation is required. UML meets the following requirements:

- Formal definition of a common OA&D meta-model to represent the semantics of OA&D models, which include static models, behavioral models, usage models, and architectural models.
- IDL specifications for mechanisms for model interchange between OA&D tools. This document includes a set of IDL interfaces that support dynamic construction and traversal of a user model.
- A human-readable notation for representing OA&D models. This document defines the UML notation, an elegant graphic syntax for consistently expressing the UML's rich semantics. Notation is an essential part of OA&D modeling and the UML.

0.2 About the Object Management Group (OMG)

The Object Management Group, Inc. (OMG) is an international organization supported by over 800 members, including information system vendors, software developers and users. Founded in 1989, the OMG promotes the theory and practice of object-oriented technology in software development. The organization's charter includes the establishment of industry guidelines and object management specifications to provide a common framework for application development. Primary goals are the reusability, portability, and interoperability of object-based software in distributed, heterogeneous environments. Conformance to these specifications will make it possible to develop a heterogeneous applications environment across all major hardware platforms and operating systems.

OMG's objectives are to foster the growth of object technology and influence its direction by establishing the Object Management Architecture (OMA). The OMA provides the conceptual infrastructure upon which all OMG specifications are based.

Contact the Object Management Group, Inc. at:

OMG Headquarters
492 Old Connecticut Path
Framingham, MA 01701
USA
Tel: +1-508-820 4300
Fax: +1-508-820 4303
pubs@omg.org
http://www.omg.org

OMG’s adoption of the UML specification reduces the degree of confusion within the industry surrounding modeling languages. It settles unproductive arguments about method notations and model interchange mechanisms and allows the industry to focus on higher leverage, more productive activities. Additionally, it enables semantic interchange between visual modeling tools.

0.3 About This Document

This document is intended primarily as a precise and self-consistent definition of the UML’s semantics and notation. The primary audience of this document consists of the Object Management Group, standards organizations, book authors, trainers, and tool builders. The authors assume familiarity with object-oriented analysis and design methods. The document is not written as an introductory text on building object models for complex systems, although it could be used in conjunction with other materials or instruction. The document will become more approachable to a broader audience as additional books, training courses, and tools that apply to UML become available.

The Unified Modeling Language specification defines compliance to the UML, covers the architectural alignment with other technologies, and is comprised of the following topics:

UML Summary (Chapter 1) - provides an introduction to the UML, discussing motivation and history.

UML Semantics (Chapter 2) - defines the right semantics of the Unified Modeling Language. The UML is layered architecturally and organized by package. Within each package, the model elements are defined in the following terms:

1. Abstract syntax	UML class diagrams are used to present the UML metamodel, its concepts (metaclasses), relationships, and constraints. Definitions of the concepts are included.
2. Well-formedness rules	The rules and constraints on valid models are defined. The rules are expressed English prose and in a precise Object Constraint Language (OCL). OCL is a specification language that uses simple logic for specifying invariant properties of systems comprising sets and relationships between sets.
3. Semantics	The semantics of model usage are described in English prose.

UML Notation Guide (Chapter 3) - represents the graphic syntax for expressing the semantics described by the UML metamodel. Consequently, the UML Notation Guide's chapters should be read in conjunction with the UML Semantics chapters.

UML Extensions (Chapter 4) - contains the UML Extension for Objectory Process for Software Engineering and UML Extension for Business Modeling.

OA&D CORBAfacility Interface Definition (Chapter 5) - contains the UML-consistent interoperability defined in terms of CORBA IDL.

In addition, you will find an appendix of Standard Elements and an appendix that contains the Object Contraint Language (OCL) syntax, semantics, and grammar. All OCL features are described in terms of concepts from the UML Semantics chapter.

0.3.1 Dependencies Between Sections

UML Semantics (Chapter 2) can stand on its own, relative to the others, with the exception of the *OCL Specification*. The semantics and the OCL are interdependent. The semantics and notation are nearly independent. What this means is that you can certainly specify and understand each one in isolation, but the one affects the other. For example, knowing what kinds of things a developer or modeler finds important to visualize impacts what kind of underlying semantics are needed. For example, modeling patterns is something that in our experience we find to be valuable for systems of scale; this is why the UML metamodel has collaborations as a first-class citizen. If one does not consider what is important to be visualized, you end up with a less rich metamodel.

The *UML Notation Guide* and *OA&D CORBAfacility Interface Definition* both depend on the semantics. We consider it advantageous to separate the UML definition and the facility interface. Having these as separate standards will permit their evolution in the most flexible way, even though they are not completely independent.

The specifications in the *UML Extension* documents depend on both the notation and semantics sections.

0.4 Compliance to the UML

The UML and corresponding facility interface definition are comprehensive. However, these specifications are packaged so that subsets of the UML and facility can be implemented without breaking the integrity of the language. The UML Semantics is packaged as follows:

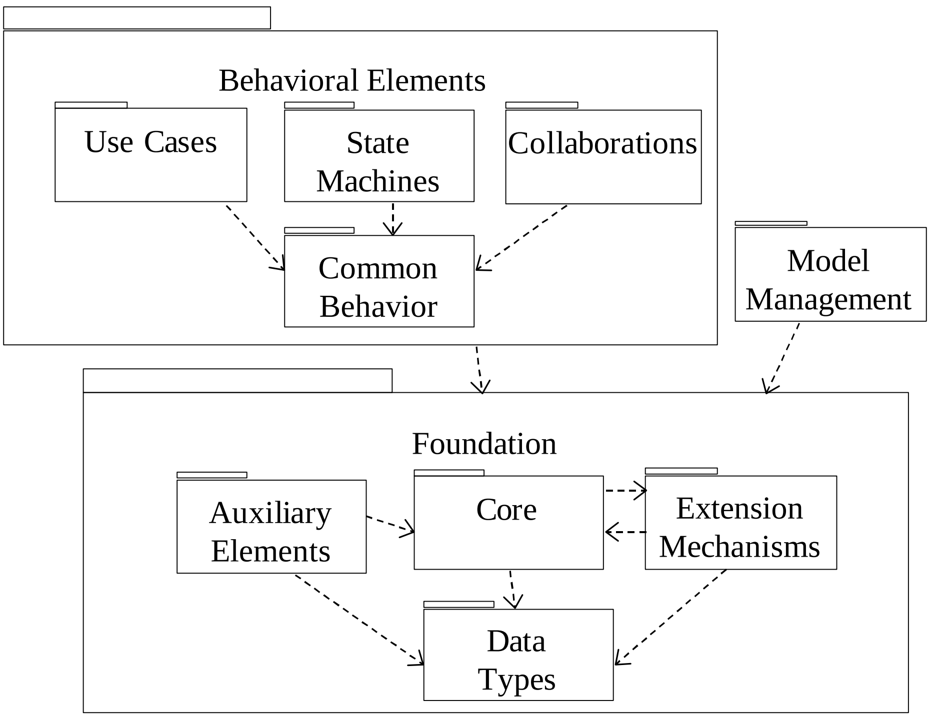

Figure 0-1 UML Class Diagram Showing Package Structure

This packaging shows the semantic dependencies between the UML model elements in the different packages. The IDL in the facility is packaged almost identically. The notation is also “packaged” along the lines of diagram type. Compliance of the UML is thus defined along the lines of semantics, notation, and IDL.

Even if the compliance points are decomposed into more fundamental units, vendors implementing UML may choose not to fully implement this packaging of definitions, while still faithfully implementing some of the UML definitions. However, vendors who want to precisely declare their compliance to UML should refer to the precise language defined herein, and not loosely say they are “UML compliant.”

0.4.1 Compliance to the UML Semantics

The basic units of compliance are the packages defined in the UML metamodel. The full metamodel includes the corresponding semantic rigor defined in the Semantics section.

The class diagram illustrates the package dependencies, which are also summarized in the table below.

Table 0-1 Metamodel Packages

Package	Prerequisite Packages
DataTypes	
Core	DataTypes
Auxiliary Elements	Core, DataTypes
Common Behavior	Core, DataTypes
State Machines	Common Behavior, Core, DataTypes
Collaboration	Common Behavior, Core, DataTypes
Use Cases	Collaboration, Common Behavior, Core, DataTypes
Model Management	Core, DataTypes
Extension Mechanisms	Core, DataTypes

Complying with a package requires complying with the prerequisite package.

The semantics are defined in an implementation language-independent way. An implementation of the semantics, without consistent interface and implementation choices, does not guarantee tool interoperability. See the *OA&D CORBAfacility Interface Definition* (chapter 16).

In addition to the above packages, compliance to a given metamodel package must load or know about the predefined UML standard elements (i.e., values for all predefined stereotypes, tags, and constraints). These are defined throughout the semantics and notation documents and summarized in the UML Standard Elements appendix. The predefined constraint values must be enforced consistent with their definitions. Having tools know about the standard elements is necessary for the full language and to avoid the definition of user-defined elements that conflict with the standard UML elements. Compliance to the UML Extensions is defined separate from the UML Semantics, so not all tools need to know about the UML Extensions a priori.

For any implementation of UML, it is optional that the tool implement the Object Constraint Language. A vendor conforming to OCL support must support the following:

- Validate and store syntactically correct OCL expressions as values for the UML data types BooleanExpression, Expression, ObjectSetExpression, TimeExpression, and ProcedureExpression.
- Be able to perform a full type check on the object constraint expression. This check will test whether all features used in the expression are actually defined in the UML model and used correctly.

All tools conforming to the UML semantics are expected to conform to the following aspects of the semantics:

- its abstract syntax (i.e., the concepts, valid relationships, and constraints expressed in the corresponding class diagrams),
- well-formedness rules, and
- semantics.

However, vendors are expected to apply some discretion on how strictly the well-formedness rules are enforced; tools should be able to report on well-formedness violations, but not necessarily force all models to be well formed. Incomplete models are common during certain phases of the development lifecycle, so they should be permitted. See the *OA&D CORBAfacility Interface Definition* (chapter16) for its treatment of well-formedness exception handling, as an example of a technique to report well-formedness violations.

0.4.2 Compliance to the UML Notation

The UML notation is an essential element of the UML to enable communication between team members. Compliance to the notation is optional, but the semantics are not very meaningful without a consistent way of expressing them.

Notation compliance is defined along the lines of the UML Diagrams types: use case, class, statechart, activity, sequence, collaboration, component, and deployment diagrams.

If the notation is implemented, a tool must enforce the underlying semantics and maintain consistency between diagrams if the diagrams share the same underlying model. By this definition, a simple "drawing tool" cannot be compliant to the UML notation.

There are many optional notation adornments. For example, a richly adorned class icon may include an embedded stereotype icon, a list of properties (tagged values and metamodel attributes), constraint expressions, attributes with visibilities indicated, and operations with full signatures. Complying with class diagram support implies the ability to support all of the associated adornments.

Compliance to the notation in the *UML Extensions* is described separately.

0.4.3 Compliance to the UML Extensions

Vendors should specify whether they support each of the UML Extensions or not. Compliance to an extension means knowledge and enforcement of the semantics and corresponding notation.

0.4.4 Compliance to the OA&D CORBAfacility Interface Definitions

The IDL modules defined in the OA&D CORBAfacility parallel the packages in the semantic metamodel. The exception to this is that DataTypes and Extension mechanisms have been merged in with the core for the facility. Except for this, a CORBAfacility implementing the interface modules have the same compliance point options as described in "Compliance to the UML Notation" listed above.

0.4.5 Summary of Compliance Points

Table 0-2 Summary of Compliance Points

Compliance Point	Valid Options
Core	no/incomplete, complete, complete including IDL
Auxiliary Elements	no/incomplete, complete, complete including IDL
Common Behavior	no/incomplete, complete, complete including IDL
State Machines	no/incomplete, complete, complete including IDL
Collaboration	no/incomplete, complete, complete including IDL
Use Cases	no/incomplete, complete, complete including IDL
Model Management	no/incomplete, complete, complete including IDL
Extension Mechanisms	no/incomplete, complete, complete including IDL
OCL	no/incomplete, complete
Use Case diagram	no/incomplete, complete
Class diagram	no/incomplete, complete
Statechart diagram	no/incomplete, complete
Activity diagram	no/incomplete, complete
Sequence diagram	no/incomplete, complete
Collaboration diagram	no/incomplete, complete
Component diagram	no/incomplete, complete
Deployment diagram	no/incomplete, complete
UML Extension: Business Engineering	no/incomplete, complete
UML Extension: Objectory Process for Software Engineering	no/incomplete, complete

0.5 Acknowledgements

The UML was crafted through the dedicated efforts of individuals and companies who find UML strategic to their future. This section acknowledges the efforts of these individuals who contributed to defining UML.

UML 1.1 Core Team

- *Hewlett-Packard Company*: Martin Griss
- *IBM Corporation*: Steve Cook, Jos Warmer
- *ICON Computing*: Desmond D'Souza
- *I-Logix*: Eran Gery, David Harel
- *MCI Systemhouse Corporation*: Cris Kobryn, Joaquin Miller
- *IntelliCorp and James Martin & Co.*: James Odell
- *ObjecTime Limited*: John Hogg, Bran Selic
- *Oracle Corporation*: Guus Ramackers
- *PLATINUM Technology Inc.*: Dilhar DeSilva
- *Rational Software*: Grady Booch, Ed Eykholt (project lead), Ivar Jacobson, Gunnar Overgaard, Karin Palmkvist, Jim Rumbaugh
- *Taskon A/S*: Trygve Reenskaug
- *Sterling Software*: John Cheesman, Keith Short
- *Unisys Corporation*: Sridhar Iyengar, GK Khalsa

UML 1.1 Semantics Task Force

During the final submission phase, a team was formed to focus on improving the formality of the UML 1.0 semantics, as well as incorporating additional ideas from the partners. Under the leadership of Cris Kobryn, this team was very instrumental in reconciling diverse viewpoints into a consistent set of semantics, as expressed in the revised *UML Semantics*. Other members of this team were Dilhar DeSilva, Martin Griss, Sridhar Iyengar, Eran Gery, Gunnar Overgaard, Karin Palmkvist, Guus Ramackers, Bran Selic, and Jos Warmer. Booch, Jacobson, and Rumbaugh provided their expertise to the team, as well.

Contributors and Supporters

We also acknowledge the contributions, influence, and support of the following individuals.

Jim Amsden, Hernan Astudillo, Colin Atkinson, Dave Bernstein, Philip A. Bernstein, Michael Blaha, Conrad Bock, Mike Bradley, Ray Buhr, Gary Cernosek, James Cerrato, Michael Jesse Chonoles, Magnus Christerson, Dai Clegg, Peter Coad, Derek Coleman, Ward Cunningham, Raj Datta, Mike Devlin, Philippe Desfray, Bruce Douglass, Staffan Ehnebom, Maria Ericsson, Johannes Ernst, Don Firesmith, Martin Fowler, Adam Frankl, Eric Gamma, Dipayan Gangopadhyay, Garth Gullekson, Rick Hargrove, Tim Harrison, Richard Helm, Brian Henderson-Sellers, Michael Hirsch, Bob Hodges, Glenn Hollowell, Yves Holvoet, Jon Hopkins, John Hsia, Ralph Johnson, Anneke Kleppe, Philippe Kruchten, Paul Kyzivat, Martin Lang, Grant Larsen, Reed Letsinger, Mary Loomis, Jeff MacKay, Robert Martin, Terrie McDaniel, Jim McGee, Bertrand Meyer, Mike Meier, Randy Messer, Greg Meyers, Fred Mol, Luis Montero, Paul Moskowitz, Andy Moss, Jan Pachl, Paul Patrick, Woody Pidcock, Bill Premerlani, Jeff Price, Jerri Pries, Terry Quatrani, Mats Rahm, George Reich, Rich Reitman, Rudolf M. Riess, Erick Rivas, Kenny Rubin, Jim Rye, Danny Sabbah, Tom Schultz, Ed Seidewitz, Gregson Siu, Jeff Sutherland, Dan Tasker, Dave Tropeano, Andy Trice, Dan Uhlar, John Vlissides, Larry Wall, Paul Ward, Alan Wills, Rebecca Wirfs-Brock, Bryan Wood, Ed Yourdon, and Steve Zeigler.

0.6 References

[Bock/Odell 94]	C. Bock and J. Odell, "A Foundation For Composition," Journal of Object-oriented Programming, October 1994.
[Booch et al.]	Grady Booch, Jim Rumbaugh, and Ivar Jacobson, Unified Modeling Language User Guide, ISBN: 0-201-57168-4, Addison Wesley, est. publication December 1997. See *www.awl.com/cp/uml/uml.html*.
[Cook 94]	S. Cook and J. Daniels, Designing Object Systems: Object-oriented Modelling with Syntropy, Prentice-Hall Object-Oriented Series, 1994.
[D'Souza 97a]	D. D'Souza and A. Wills, "Input for the OMG Submission," *www.iconcomp.com/catalysis*
[D'Souza 97b]	D. D'Souza and A. Wills, "Catalysis: Component and Framework based development" *www.iconcomp.com/catalysis*
[Fowler 97]	M. Fowler with K. Scott, UML Distilled: Applying the Standard Object Modeling Language, ISBN 0-201-32563-2, Addison-Wesely, 1997. *http://www.awl.com/cp/uml/uml.html*
[Griss 96]	M. Griss, Domain Engineering And Variability In The Reuse-Driven Software Engineering Business. Object Magazine. Dec 1996. (See *www.hpl.hp.com/reuse*)
[Harel 87]	D. Harel, "Statecharts: A Visual Formalism for Complex Systems," Science of Computer Programming 8 (1987), 231-274.
[Harel 96a]	D. Harel and E. Gery, "Executable Object Modeling with Statecharts," Proc. 18th Int. Conf. Soft. Eng., Berlin, IEEE Press, March, 1996, pp. 246-257.
[Harel 96b]	D. Harel and A. Naamad, "The STATEMATE Semantics of Statecharts," ACM Trans. Soft. Eng. Method 5:4 (Oct. 1996).

[Jacobson et al.]	Ivar Jacobson, Grady Booch, and Jim Rumbaugh, The Objectory Software Development Process, ISBN: 0-201-57169-2, Addison Wesley est. publication December 1997. See *www.awl.com/cp/uml/uml.html* and the "Rational Objectory Process" on *www.rational.com.*
[Malan 96]	R. Malan, D. Coleman, R. Letsinger et al, The Next Generation of Fusion, Fusion Newsletter, Oct 1996. (See *www.hpl.hp.com/fusion.*)
[Martin/Odell 95]	J. Martin and J. Odell, Object-oriented Methods, A Foundation, ISBN: 0-13-630856-2, Prentice Hall, 1995
[Ramackers 95]	Ramackers, G. and Clegg, D., "Object Business Modelling, requirements and approach" in Sutherland, J. and Patel, D. (eds.), Proceedings of the OOPSLA95 workshop on Business Object Design and Implementation, Springer Verlag, publication pending.
[Ramackers 96]	Ramackers, G. and Clegg, D., "Extended Use Cases and Business Objects for BPR," ObjectWorld UK '96, London, June 18-21, 1996.
[Rumbaugh et al.]	Jim Rumbaugh, Ivar Jacobson, and Grady Booch, Unified Modeling Language Reference Manual, ISBN: 0-201-30998-X, Addison Wesley, est. publication December 1997. See *www.awl.com/cp/uml/uml.html*.
[UML Web Sites]	*www.rational.com/uml* *uml.systemhouse.mci.com*