

CHAPTER 1

Introduction

This chapter introduces the goals, scope, and formal documentation of the Unified Modeling Language (UML), along with its history and evolution. Upon reading this chapter, you will understand:

- The scope and goals of the UML, and the various issues it attempts to address.
- What documents formally define and establish the UML.
- The historical evolution of the UML.

This chapter introduces some of the key motivations behind the UML. I start by introducing the UML's goals and scope. I then briefly identify the formal documents that define the UML. You should reference these documents for detailed information beyond this book. I then review the history of how the UML came to be.

What Is the Unified Modeling Language?

The Unified Modeling Language

The Unified Modeling Language (Figure 1-1) is a language that unifies the industry's best engineering practices for modeling systems. The UML

- Is a *language*. It is not simply a notation for drawing diagrams, but a complete language for capturing knowledge (semantics) about a subject and expressing knowledge (syntax) regarding the subject for the purpose of communication.
- Applies to *modeling* and systems. Modeling involves a focus on understanding (knowing) a subject (system) and capturing and being able to communicate this knowledge.
- Is the result of *unifying* the information systems and technology industry's best engineering practices (principles, techniques, methods, and tools).
- Is used for specifying, visualizing, constructing, and documenting systems.

- Is used for expressing the artifacts of a system-intensive process.

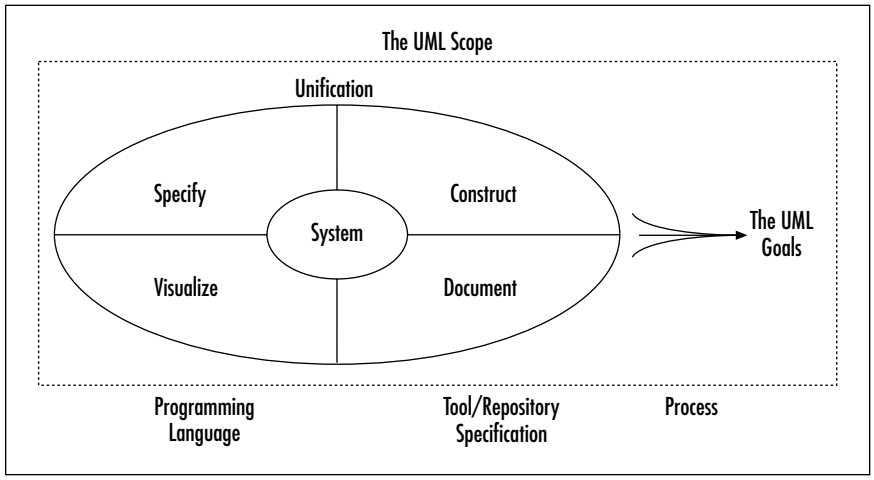


Figure 1-1: The Unified Modeling Language

- Is based on the object-oriented paradigm.
- Is a modeling language for specifying, visualizing, constructing, and documenting the artifacts of a system-intensive process.
- Is an evolutionary general-purpose, broadly applicable, tool-supported, industry-standardized modeling language.
- Applies to a multitude of different types of systems, domains, and methods or processes.
- Enables the capturing, communicating, and leveraging of strategic, tactical, and operational knowledge to facilitate increasing value by increasing quality, reducing costs, and reducing time-to-market while managing risks and being proactive in regard to ever-increasing change and complexity.
- Was originally conceived by Rational Software Corporation and the Three Amigos.
  - Grady Booch's work has involved the use of Ada and the application of data abstraction and information hiding with an emphasis on iterative software development and software architecture. He has been influential in disseminating object orientation. He is the inventor of the Booch method, as expressed in his book *Object-Oriented Analysis and Design*.
  - James Rumbaugh's work has involved various topics concerning research and development within the context of complex algorithms and data structures. He coauthored the Object Modeling Technique (OMT) with Mike Blaha, Bill Premerlani, Fred Eddy, and Bill Lorensen.
  - Ivar Jacobson's work has involved the application of object orientation to the context of business reengineering and the industrialization of software development. He is often known as the father of use cases. He is the inventor of the Object-Oriented Software Engineering (OOSE)

method, as expressed in his book *Object-Oriented Software Engineering: A Use Case Driven Approach*.

- Is supported by the UML Partners Consortium (Rational Software Corporation, Microsoft Corporation, Hewlett-Packard Company, Oracle Corporation, Sterling Software, MCI Systemhouse Corporation, Unisys Corporation, ICON Computing.

Unification

Unification creates a single, consistent system from the most prominent methods (at the modeling language level) within the industry. Unification

- Involves uniting the modeling languages of the following methods:
  - Grady Booch's method: A design- and construction-oriented approach supporting excellent expressiveness for engineering-intensive systems.
  - James Rumbaugh and associates' Object Modeling Technique (OMT) method: An analysis-oriented approach supporting excellent expressiveness for data-intensive systems.
  - Ivar Jacobson's Object-Oriented Software Engineering (OOSE) method: A use-case-oriented approach supporting excellent expressiveness for business engineering and requirements analysis.
- Incorporates other methods and the best engineering practices to capture lessons learned.
- Occurs across types of systems (software and nonsoftware), domains (business versus software), and development phases (requirements, analysis, design, implementation, testing, and deployment).
- Involves modeling language semantics and syntax, not development processes.
- Addresses contemporary and future development issues.

The UML Goals

The goals of the UML are to

- Be a ready-to-use expressive visual modeling language that is simple and extensible.
- Have extensibility and specialization mechanisms for extending, rather than modifying, core concepts.
  - Formalize a core set of concepts that constitute the object-oriented paradigm. Extensions will not require reimplementation of the core object-oriented paradigm (analysis and design) concepts in order to be applicable to a broad set of applications. Instead, practitioners can extend this core set of concepts based on varying interpretations without having to repeatedly redefine the fundamental concepts.
  - Allow adding new concepts and notation beyond the core.
  - Allow variant interpretations of existing concepts when there is no clear consensus.

  - Allow specialization of concepts, notation, and constraints for particular domains.
- Be implementation independent (programming language).
- Be process independent (development).
- Encourage the growth of the object-oriented tools market.
- Support higher-level concepts (collaborations, frameworks, patterns, and components).
- Address recurring architectural complexity problems (physical distribution and distributed systems, concurrency and concurrent systems, replication, security, load balancing, and fault tolerance) using component technology, visual programming, patterns, and frameworks.
- Be scalable.
- Be widely applicable (general purpose and powerful) and usable (simple, widely accepted, evolutionary).
- Integrate the best engineering practices.

The UML Scope

The scope of the UML encompasses

- Fusing the concepts of Booch, OMT, and OOSE.
- Focusing on a standard modeling language (common semantics and then common notation) and not a standard process.
- Incorporating the object-oriented community's consensus on core modeling concepts.
- Providing sufficient notation to address contemporary modeling issues directly and economically.
- Providing sufficient semantics to address the following issues directly and economically:
  - Contemporary modeling issues
  - Future modeling issues (specifically related to component technology, distributed computing, frameworks, and executability)
  - Model interchange among tools
  - Repository interfaces (to modeling artifacts) among tools
- Providing extensibility mechanisms to address the following issues directly and economically:
  - Current deviations within the object-oriented community
  - Extending the UML for individual projects
  - Future modeling approaches (on top of the UML)

What the UML Is Not

The UML is not

- A visual programming language, but a visual modeling language.
  - A programming language communicates an implementation or solution.
  - A modeling language communicates a model (or conceptualization or specification).
- A tool or repository specification, but a modeling language specification.
  - A tool or repository specification specifies a tool's or repository's interface, storage, run-time behavior, and so forth.
  - A modeling language specification specifies modeling elements, notation, and usage guidelines.
- A process, but enables processes.
  - A process provides guidance regarding the order of activities, specification of artifacts to be developed, direction of individual developer and team tasks (or activities), and monitoring and measuring criteria of project artifacts and activities.
  - Processes are organization, culture, and domain specific and dependent.
  - The UML enables and promotes (but does not require nor mandate) a use-case-driven, architecture-centric, iterative, and incremental process that is object oriented and component based.

What Constitutes the Unified Modeling Language?

The UML Definition

The UML is defined by a set of documents (Figure 1-2) from Rational Software Corporation and the Object Management Group. The UML definition

- Is expressed by the following documents (the UML-defining artifacts):
  - UML Semantics
  - UML Notation Guide
  - UML Extension for the Objectory Process for Software Engineering
  - UML Extension for Business Modeling
- Uses the Object Constraint Language (OCL) defined by the Object Constraint Language specification document.
- May be obtained from Rational Software Corporation or the Object Management Group via the World Wide Web (see the references section)

The UML Semantics Document

The UML semantics document constitutes the single, common, definitive, formal, comprehensive, and precise specification of the UML (called the *inside view*). The UML semantics document

- Is primarily used by tool vendors; however, it may be used by practitioners since it is the definitive definition of the UML.

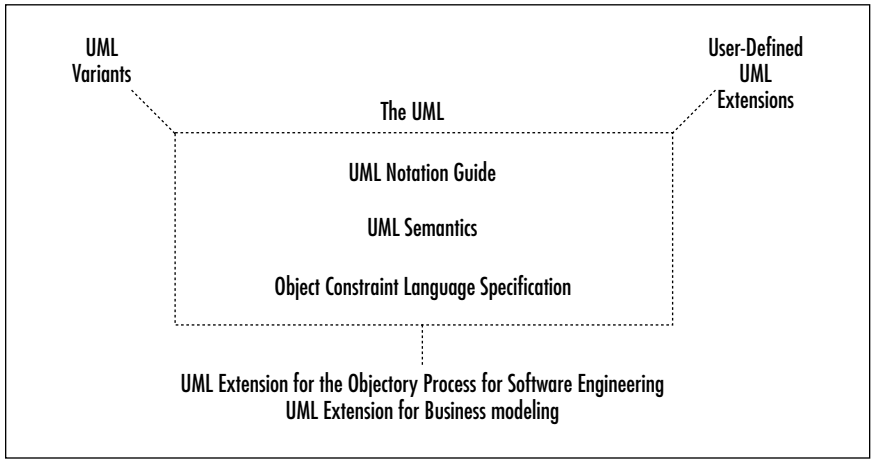


Figure 1-2: The UML Definition

- Specifies the UML's layered architecture, organization (using packages), and defined modeling elements.
- Includes the following inside-view information for a given UML package:
  - Overview: General introduction
  - Abstract syntax: Concept definitions, relationships, and constraints expressed using the UML's graphical notation and English prose
  - Well-formedness rules: Rules and constraints expressed using English prose and the OCL
  - Semantics: Meanings expressed using English prose
  - Standard elements: Applicable standard extension mechanisms
  - Notes: Other commentary
- Also contains appendices on the standard elements and the UML glossary.
- Enables the development of complex systems, consistent usage of the UML, and tool interchange

The UML Notation Guide

The UML Notation Guide constitutes the notational or visual representation of the UML and provides examples (called the *outside view*). The UML Notation Guide

- Is primarily used by practitioners applying the UML
- Specifies the UML's diagrams and their modeling elements
- Includes the following outside-view information for each modeling element:
  - Semantics: Summarizes the UML semantics.
  - Notation (concrete syntax): Explains the notational representation (forward mapping to notation).

  - Presentation options: Describe options in presenting modeling information.
  - Style guidelines: Suggest stylistic markers and options.
  - Examples: Provide notation samples.
  - Mapping: Explains the mapping of the notation to the semantics document (reverse mapping from notation).
- Enables the use of the UML.

The UML Extension Documents

The UML extension documents provide user-defined extensions (using extension mechanisms). These documents

- Extend the UML to the objectory process for software engineering.
- Extend the UML to business modeling.

A *UML extension* is a set of extensions (stereotypes, tagged values, and constraints) that extend, customize, or tailor the UML for a specific domain or process.

A *UML variant* is a semantically well-defined language (expressed as a metamodel) based on (and built on top of) the UML metamodel. It specializes the UML but does not change the UML terms or redefine their meanings.

The Object Constraint Language

The Object Constraint Language (OCL) is a formal, easily readable and writable, nonprogramming, implementation-independent language. The OCL

- Is used for expressing typed, pure, side-effect-free (precise and unambiguous) constraints and expressions.
- Is used for attaching constraints and other expressions to the UML models and modeling elements (or objects).
- Is used to specify the well-formedness rules of the UML within the UML semantics document.
- Is formally specified in the Object Constraint Language specification document, which provides the following information: Connection with the UML semantics, basic values and types, objects and properties, collection operations, predefined OCL types, and the grammar for the OCL.

The Evolution of the Unified Modeling Language

Fragmentation

The historical period between the mid-1970s and the mid-1990s (Figure 1-3) was characterized by fragmentation.

- Software proved to have strategic, tactical, and operational value for organizations.

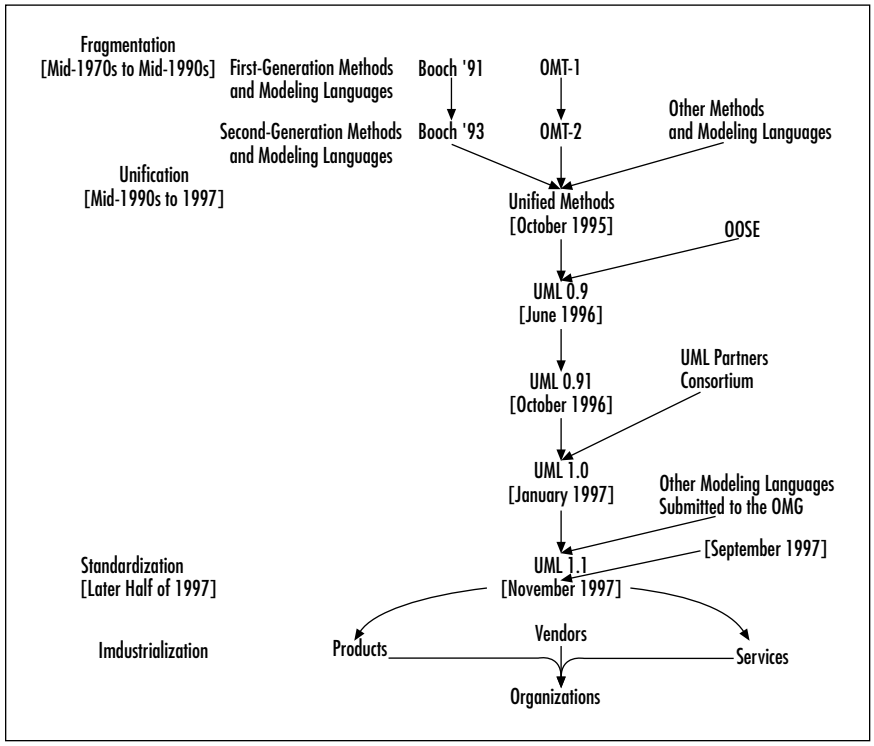


Figure 1-3: The Evolution of the Unified Modeling Language

- Techniques for controlling and automating the production of software were sought in order to increase value by increasing quality, reducing costs, and reducing time-to-market while managing risks and being proactive concerning ever-increasing change and complexity.
- Various techniques and methods emerged to capture, communicate, and leverage technical and business artifacts and knowledge; within the software world, the modern reuse movement was born.
- Object-oriented modeling languages began to appear between the mid-1970s and the late 1980s.
- The number of modeling languages and methods increased from less than 10 to more than 50 between 1989 and 1994. These are known as *first-generation methods*.
- "Method wars" were perpetuated because practitioners had trouble finding satisfaction in using any one modeling language. There was no clearly leading modeling language, but many similar modeling languages with minor differences in overall expressive power. They all shared a set of commonly accepted concepts, but expressed them differently.

- Lack of agreement discouraged new users (due to the cost of use) and vendors (due to the cost of support) from entering the object-oriented tools and methodology market.
- Users longed for a broadly supported general-purpose modeling language.
- New iterations of first-generation methods began to appear that incorporated each other's techniques during the mid-1990s. These are known as *second-generation methods*.
- The following prominent methods emerged: Booch '93 (from Booch '91), OMT-2 (from OMT-1), and OOSE.

Unification

The historical period between the mid-1990s and 1997 (Figure 1-3) was characterized by unification.

- Jim Rumbaugh joined Grady Booch at Rational Software Corporation to unify their methods in October 1994.
- Grady Booch and Jim Rumbaugh released version 0.8 of the Unified Method (as the UML was then called) in October 1995.
- Ivar Jacobson and his Objectory Company joined Rational Software Corporation to merge his method in the fall of 1995. The three methodologists became known as the Three Amigos.
- The basis of the Three Amigos' unification effort concerning the UML included the following:
  - Since their independent methods were evolving toward one another, they could continue the evolution by unifying their semantics and notation.
  - By unifying their semantics and notation, they could bring stability to the object-oriented marketplace and promote the use of object orientation via a mature modeling language.
  - By collaborating, they could improve their earlier methods and build on lessons learned while addressing problems that their earlier methods insufficiently addressed.
- The goals of the Three Amigos' unification effort concerning the UML were, among others, to
  - Enable the modeling of systems, not just software systems, using the object-oriented paradigm and concepts.
  - Couple conceptual as well as executable artifacts.
  - Address scalability issues involving complex mission-critical systems.
  - Be usable by both humans and machines.
- The unification approach used by the Three Amigos involved trade-offs regarding the following issues:

  - Binding the problem. What is the scope of the language? Should it address requirements gathering? (Yes.) Should it address visual programming? (No.)
  - Balancing expressiveness and simplicity. Expressiveness involves a balance between minimalism and overengineering. Simplification involves removing existing elements that did not work in practice, adding more effective elements from other methods, and inventing new elements only when existing elements are not available. If the language is too simple, the number of problems it may address becomes limited. If the language is too complex, its usability becomes hindered.
  - Being sensitive to the installed base of methods. Introducing too much new notation will only cause unnecessary confusion. Not introducing (and unifying) notation will miss the opportunity to engage a broader set of users.
- Identify and incorporate underlying fundamental semantic concepts, agreeing on the importance and consequences of these concepts, building a metamodel to precisely describe these concepts, building a notation connected to the abstract syntax of these concepts, and organizing these concepts across abstraction layers, levels of complexity, and domains.
- The Three Amigos released version 0.9 of the UML in June 1996, and version 0.91 in October 1996. They invited, received, and incorporated feedback from the general community during 1996.
- Organizations began to see the strategic strength of the UML during 1996.
- The Object Management Group's Object Analysis and Design Task Force issued a Request for Proposal (RFP) regarding establishing standards for tools that support object-oriented analysis and design; it was aimed at defining semantics and a metamodel standard for object-oriented technology.
- The OMG's request provided the catalyst for organizations to join forces and jointly respond.
- Rational Software Corporation formed the UML Partners Consortium.
- The UML Partners collaborated and released a well-defined, expressive, powerful, and generally applicable version 1.0 of the UML that was submitted to the OMG (as an initial RFP response) in January 1997. Other organizations also submitted separate RFP responses to the OMG.
- The result achieved by the Three Amigos' unification effort included the following:
  - It ends many of the (inconsequential) differences between modeling languages of previous methods.
  - It unifies many perspectives among different kinds of systems (business versus software), development phases (requirements, analysis, design, implementation, testing, and deployment), and internal concepts.
  - It introduces new concepts, including extensibility mechanisms (stereotypes, tagged values, and constraints), threads and processes, distribution

and concurrency, patterns and collaborations, activity diagrams for business modeling, refinement for levels of abstraction, interfaces and components, and a constraint language.

– It unifies the Booch, OMT, OOSE, and other object-oriented methods (at the modeling language level) with object-oriented and computer science practices from the object-oriented field and community of ideas. By removing unnecessary differences in notation and terminology that obscure underlying similarities, the UML is an evolutionary modeling language that is more expressive yet cleaner, more unified, and more holistically integrated than its predecessors.

Standardization

The historical period of standardization (Figure 1-3) began in the latter half of 1997 and had the following highlights:

- Others who submitted responses to the OMG joined the UML Partners to contribute their ideas and produce version 1.1 of the UML during 1997.
- The UML Partners collaborated and released a more clear and updated version 1.1 of the UML that was submitted to the OMG (as a revised RFP response) for consideration and adoption as a standard in September 1997.
- The following constitute the differences (and changes) between UML version 1.0 and UML version 1.1:
  - Increased formalism in defining the UML and eradicating ambiguities.
  - Improved structural packaging.
  - Increased unification of collaboration and interaction semantics and of relationship semantics.
  - Increased simplification of the class, type, and interface semantics model.
  - Incorporation of extensions (model management semantics, including models and subsystems and use case semantics).
  - Improved mapping of notation to semantics.

Industrialization

The historical period of industrialization (Figure 1-3) started during standardization and flourished once the OMG adopted the UML. Some of its highlights and characteristics are as follows:

- The UML was adopted by the OMG on November 17, 1997.
- The UML definition is readily available and evolving.
- Industry organizations and vendors embraced the UML.
- Products and services for the UML emerged and will continue to do so.
- Products (tools, books, etc.) will incorporate the usage of the UML.

- Services (training, mentoring, consulting, etc.) will promote and support the use of the UML.

The UML's success will be measured by its appropriate use on successful projects. The UML does not guarantee project success, but does enable practitioners to focus on delivering business value using a consistent, standardized, and tool-supported modeling language.