
**Information technology — Object
Management Group Unified Modeling
Language (OMG UML) —**

**Part 1:
Infrastructure**

*Technologies de l'information — Langage de modélisation unifié OMG
(OMG UML) —*

Partie 1: Infrastructure



COPYRIGHT PROTECTED DOCUMENT

© ISO/IEC 2012

All rights reserved. Unless otherwise specified, no part of this publication may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm, without permission in writing from either ISO at the address below or ISO's member body in the country of the requester.

ISO copyright office
Case postale 56 • CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 749 09 47
E-mail copyright@iso.org
Web www.iso.org

Published in Switzerland

Table of Contents

1. Scope	1
2. Conformance	1
2.1 General	1
2.2 Language Units	2
2.3 Compliance Levels	2
2.4 Meaning and Types of Compliance	3
2.5 Compliance Level Contents	5
3. Normative References	5
4. Terms and Definitions	6
5. Notational Conventions	6
6. Additional Information	6
6.1 Architectural Alignment and MDA Support	6
6.2 How to Proceed	6
6.2.1 Diagram format	7
7. Language Architecture	13
7.1 General	13
7.2 Design Principles	13
7.3 Infrastructure Architecture	13
7.4 Core	14
7.5 Profiles	16
7.6 Architectural Alignment between UML and MOF	16
7.7 Superstructure Architecture	17
7.8 Reusing Infrastructure	18
7.9 The Kernel Package	18
7.10 Metamodel Layering	18
7.11 The Four-layer Metamodel Hierarchy	19

7.12	Metamodeling	19
7.13	An Example of the Four-level Metamodel Hierarchy	20
8.	Language Formalism	23
8.1	General	23
8.2	Levels of Formalism	23
8.3	Package Specification Structure	24
8.3.1	Class Descriptions	24
8.3.2	Diagrams	24
8.3.3	Instance Model	24
8.4	Class Specification Structure	24
8.4.1	Description	25
8.4.2	Attributes	25
8.4.3	Associations	25
8.4.4	Constraints	25
8.4.5	Additional Operations (optional)	25
8.4.6	Semantics	25
8.4.7	Semantic Variation Points (optional)	25
8.4.8	Notation	26
8.4.9	Presentation Options (optional)	26
8.4.10	Style Guidelines (optional)	26
8.4.11	Examples (optional)	26
8.4.12	Rationale (optional)	26
8.4.13	Changes from UML 1.4	26
8.5	Use of a Constraint Language	26
8.6	Use of Natural Language	27
8.7	Conventions and Typography	27
9.	Core::Abstractions	31
9.1	BehavioralFeatures Package	33
9.1.1	BehavioralFeature	33
9.2	Parameter	34
9.3	Changeabilities Package	35
9.3.1	StructuralFeature (as specialized)	36
9.4	Classifiers Package	36
9.4.1	Classifier	37
9.4.2	Feature	38
9.5	Comments Package	39
9.5.1	Comment	39
9.5.2	Element	40

9.6	Constraints Package	41
9.6.1	Constraint	42
9.6.2	Namespace (as specialized)	45
9.7	Elements Package	46
9.7.1	Element	46
9.8	Expressions Package	47
9.8.1	Expression	47
9.8.2	OpaqueExpression	48
9.8.3	ValueSpecification	49
9.9	Generalizations Package	51
9.9.1	Classifier (as specialized)	52
9.9.2	Generalization	53
9.10	Instances Package	54
9.10.1	InstanceSpecification	55
9.10.2	InstanceValue	58
9.10.3	Slot	59
9.11	Literals Package	60
9.11.1	LiteralBoolean	60
9.11.2	LiteralInteger	61
9.11.3	LiteralNull	62
9.11.4	LiteralReal	63
9.11.5	LiteralSpecification	64
9.11.6	LiteralString	64
9.11.7	LiteralUnlimitedNatural	65
9.12	Multiplicities Package	66
9.12.1	MultiplicityElement	67
9.13	MultiplicityExpressions Package	70
9.13.1	MultiplicityElement (specialized)	71
9.14	Namespaces Package	73
9.14.1	NamedElement	73
9.14.2	Namespace	75
9.15	Ownerships Package	76
9.15.1	Element (as specialized)	77
9.16	Redefinitions Package	78
9.16.1	RedefinableElement	79
9.17	Relationships Package	81
9.17.1	DirectedRelationship	81
9.17.2	Relationship	82
9.18	StructuralFeatures Package	83

9.18.1 StructuralFeature	83
9.19 Super Package	84
9.19.1 Classifier (as specialized)	85
9.20 TypedElements Package	87
9.20.1 Type	88
9.20.2 TypedElement	89
9.21 Visibilities Package	89
9.21.1 NamedElement (as specialized)	90
9.21.2 VisibilityKind	91
10. Core::Basic	93
10.1 General	93
10.2 Types Diagram	94
10.2.1 Comment	94
10.2.2 Element	95
10.2.3 NamedElement	95
10.2.4 Type	96
10.2.5 TypedElement	96
10.3 Classes Diagram	97
10.3.1 Class	97
10.3.2 MultiplicityElement	98
10.3.3 Operation	99
10.3.4 Parameter	99
10.3.5 Property	100
10.4 DataTypes Diagram	101
10.4.1 DataType	101
10.4.2 Enumeration	102
10.4.3 EnumerationLiteral	102
10.4.4 PrimitiveType	103
10.5 Packages Diagram	103
10.5.1 Package	103
10.5.2 Type	104
11. Core::Constructs	105
11.1 General	105
11.2 Root Diagram	106
11.2.1 Comment	107
11.2.2 DirectedRelationship	108
11.2.3 Element	108
11.2.4 Relationship	109
11.3 Expressions Diagram	110

11.3.1 Expression	110
11.3.2 OpaqueExpression	111
11.3.3 ValueSpecification	111
11.4 Classes Diagram	112
11.4.1 Association	113
11.4.2 Class	120
11.4.3 Classifier	123
11.4.4 Operation	126
11.4.5 Property	126
11.5 Classifiers Diagram	131
11.5.1 Classifier	132
11.5.2 Feature	133
11.5.3 MultiplicityElement	134
11.5.4 RedefinableElement	134
11.5.5 StructuralFeature	135
11.5.6 Type	136
11.5.7 TypedElement	137
11.6 Constraints Diagram	137
11.6.1 Constraint	138
11.6.2 Namespace	139
11.7 DataTypes Diagram	139
11.7.1 DataType	140
11.7.2 Enumeration	141
11.7.3 EnumerationLiteral	143
11.7.4 Operation	144
11.7.5 PrimitiveType	144
11.7.6 Property	145
11.8 Namespaces Diagram	146
11.8.1 ElementImport	146
11.8.2 NamedElement	149
11.8.3 Namespace	150
11.8.4 PackageableElement	151
11.8.5 PackageImport	152
11.9 Operations Diagram	153
11.9.1 BehavioralFeature	154
11.9.2 Operation	156
11.9.3 Parameter	159
11.9.4 ParameterDirectionKind	160
11.10 Packages Diagram	161
11.10.1 Type	161
11.10.2 Package	162
11.10.3 PackageMerge	165
12. Core::Profiles	175

12.1 General	175
12.2 Profiles package	177
12.2.1 Class (from Profiles)	178
12.2.2 Extension (from Profiles)	179
12.2.3 ExtensionEnd (from Profiles)	182
12.2.4 Image (from Profiles)	183
12.2.5 Package (from Profiles)	184
12.2.6 PackageableElement (from Profiles)	186
12.2.7 Profile (from Profiles)	186
12.2.8 ProfileApplication (from Profiles)	193
12.2.9 Stereotype (from Profiles)	194
13. PrimitiveTypes	203
13.1 General	203
13.2 PrimitiveTypes Package	203
13.2.1 Boolean.....	203
13.2.2 Integer.....	204
13.2.3 Real	205
13.2.4 String	206
13.2.5 UnlimitedNatural	207
Subpart III - Annexes	209
Annex A: XMI Serialization and Schema.....	211
Annex B: Support for Model Driven Architecture	213
Annex C: UML XMI Documents	215
INDEX	217

Foreword

ISO (the International Organization for Standardization) is a worldwide federation of national standards bodies (ISO member bodies). The work of preparing International Standards is normally carried out through ISO technical committees. Each member body interested in a subject for which a technical committee has been established has the right to be represented on that committee. International organizations, governmental and non-governmental, in liaison with ISO, also take part in the work. ISO collaborates closely with the International Electrotechnical Commission (IEC) on all matters of electrotechnical standardization.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of technical committees is to prepare International Standards. Draft International Standards adopted by the technical committees are circulated to the member bodies for voting. Publication as an International Standard requires approval by at least 75 % of the member bodies casting a vote.

Attention is drawn to the possibility that some of the elements of this part of ISO/IEC 19505 may be the subject of patent rights. ISO shall not be held responsible for identifying any or all such patent rights.

This International Standard was prepared by Technical Committee ISO/IEC/TC JTC1, Information technology, in collaboration with the Object Management Group (OMG), following the submission and processing as a Publicly Available Specification (PAS) of the OMG Unified Modeling Language (UML) specification.

This International Standard is related to:

- ITU-T Recommendations X.901-904 | ISO/IEC 10746, the Reference Model of Open Distributed Processing (RM-ODP).

This International Standard consists of the following parts, under the general title *Information technology - Open distributed processing - UML specification*:

- Part 1: Infrastructure
- Part 2: Superstructure

Apart from this Foreword, the text of this International Standard is identical with that for the OMG specification for UML, v2.4.1, Part 1.

Introduction

The rapid growth of distributed processing has led to a need for a coordinating framework for this standardization and ITU-T Recommendations X.901-904 | ISO/IEC 10746, the Reference Model of Open Distributed Processing (RM-ODP) provides such a framework. It defines an architecture within which support of distribution, interoperability, and portability can be integrated.

RM-ODP Part 2 (ISO/IEC 10746-2) defines the foundational concepts and modeling framework for describing distributed systems. The scopes and objectives of the RM-ODP Part 2 and the UML, while related, are not the same and, in a number of cases, the RM-ODP Part 2 and the UML specification use the same term for concepts that are related but not identical (e.g., interface). Nevertheless, a specification using the Part 2 modeling concepts can be expressed using UML with appropriate extensions (using stereotypes, tags, and constraints).

RM-ODP Part 3 (ISO/IEC 10746-3) specifies a generic architecture of open distributed systems, expressed using the foundational concepts and framework defined in Part 2. Given the relation between UML as a modeling language and Part 2 of the RM ODP standard, it is easy to show that UML is suitable as a notation for the individual viewpoint specifications defined by the RM-ODP.

The Unified Modeling Language (UML) is a general-purpose modeling language with a semantic specification, a graphical notation, an interchange format, and a repository query interface. It is designed for use in object-oriented software applications, including those based on technologies recommended by the Object Management Group (OMG). As such, it serves a variety of purposes including, but not limited to, the following:

- a means for communicating requirements and design intent,
- a basis for implementation (including automated code generation),
- a reverse engineering and documentation facility.

As an international standard, the various components of UML provide a common foundation for model and metadata interchange:

- between software development tools,
- between software developers, and
- between repositories and other object management facilities.

The existence of such a standard facilitates the communication between standardized UML environments and other environments.

While not limited to this context, the UML standard is closely related to work on the standardization of Open Distributed Processing (ODP).

Information technology - Object Management Group Unified Modeling Language (OMG UML), Infrastructure

1 Scope

This International Standard defines the Unified Modeling Language (UML), revision 2. The objective of UML is to provide system architects, software engineers, and software developers with tools for analysis, design, and implementation of software-based systems as well as for modeling business and similar processes.

The initial versions of UML (UML 1) originated with three leading object-oriented methods (Booch, OMT, and OOSE), and incorporated a number of best practices from modeling language design, object-oriented programming, and architectural description languages. Relative to UML 1, this revision of UML has been enhanced with significantly more precise definitions of its abstract syntax rules and semantics, a more modular language structure, and a greatly improved capability for modeling large-scale systems.

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

- A formal definition of a common MOF-based metamodel that specifies the abstract syntax of the UML. The abstract syntax defines the set of UML modeling concepts, their attributes and their relationships, as well as the rules for combining these concepts to construct partial or complete UML models.
- A detailed explanation of the semantics of each UML modeling concept. The semantics define, in a technology-independent manner, how the UML concepts are to be realized by computers.
- A specification of the human-readable notation elements for representing the individual UML modeling concepts as well as rules for combining them into a variety of different diagram types corresponding to different aspects of modeled systems.
- A detailed definition of ways in which UML tools can be made compliant with this International Standard. This is supported (in a separate specification) with an XML-based specification of corresponding model interchange formats (XMI) that must be realized by compliant tools.

2 Conformance

2.1 General

UML is a language with a very broad scope that covers a large and diverse set of application domains. Not all of its modeling capabilities are necessarily useful in all domains or applications. This suggests that the language should be structured modularly, with the ability to select only those parts of the language that are of direct interest. On the other hand, an excess of this type of flexibility increases the likelihood that two different UML tools will be supporting different subsets of the language, leading to interchange problems between them. Consequently, the definition of compliance for UML requires a balance to be drawn between modularity and ease of interchange.

Experience with previous versions of UML has indicated that the ability to exchange models between tools is of paramount interest to a large community of users. For that reason, this International Standard defines a small number of *compliance levels* thereby increasing the likelihood that two or more compliant tools will support the same or compatible language subsets. However, in recognition of the need for flexibility in learning and using the language, UML also provides the concept of *language units*.

2.2 Language Units

The modeling concepts of UML are grouped into *language units*. A language unit consists of a collection of tightly-coupled modeling concepts that provide users with the power to represent aspects of the system under study according to a particular paradigm or formalism. For example, the State Machines language unit enables modelers to specify discrete event-driven behavior using a variant of the well-known statecharts formalism, while the Activities language unit provides for modeling behavior based on a workflow-like paradigm. From the user's perspective, this partitioning of UML means that they need only be concerned with those parts of the language that they consider necessary for their models. If those needs change over time, further language units can be added to the user's repertoire as required. Hence, a UML user does not have to know the full language to use it effectively.

In addition, most language units are partitioned into multiple increments, each adding more modeling capabilities to the previous ones. This fine-grained decomposition of UML serves to make the language easier to learn and use, but the individual segments within this structure do not represent separate compliance points. The latter strategy would lead to an excess of compliance points and result to the interoperability problems described above. Nevertheless, the groupings provided by language units and their increments do serve to simplify the definition of UML compliance as explained below.

2.3 Compliance Levels

The stratification of language units is used as the foundation for defining compliance in UML. Namely, the set of modeling concepts of UML is partitioned into horizontal layers of increasing capability called *compliance levels*. Compliance levels cut across the various language units, although some language units are only present in the upper levels. As their name suggests, each compliance level is a distinct compliance point.

For ease of model interchange, there are just two compliance levels defined for UML Infrastructure:

- *Level 0 (L0)* - This contains a single language unit that provides for modeling the kinds of class-based structures encountered in most popular object-oriented programming languages. As such, it provides an entry-level modeling capability. More importantly, it represents a low-cost common denominator that can serve as a basis for interoperability between different categories of modeling tools.
- *Metamodel Constructs (LM)* - This adds an extra language unit for more advanced class-based structures used for building metamodels (using CMOF) such as UML itself.

As noted, compliance levels build on supporting compliance levels. The principal mechanism used in this International Standard for achieving this is *package merge* (see Section 11.10.3, "PackageMerge," on page -164). Package merge allows modeling concepts defined at one level to be extended with new features. Most importantly, this is achieved *in the context of the same namespace*, which enables interchange of models at different levels of compliance as described in "Meaning and Types of Compliance."

For this reason, all compliance levels are defined as extensions to a single core "UML" package that defines the common namespace shared by all the compliance levels. Level 0 is defined by the top-level metamodel shown below.

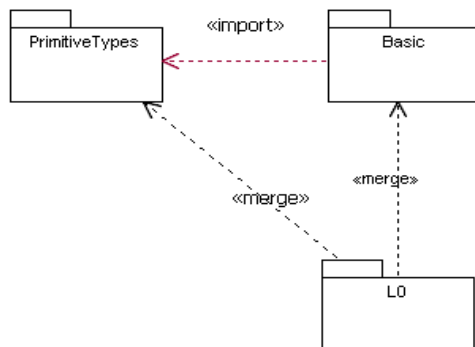


Figure 2.1 - Level 0 package diagram

In this model, "UML" is originally an empty package that simply merges in the contents of the Basic package from the UML Infrastructure. This package, contains elementary concepts such as Class, Package, DataType, Operation, etc.

At the next level (Level LM), the contents of the "UML" package, now including the packages merged into Level 0 and their contents, are extended with the Constructs package.

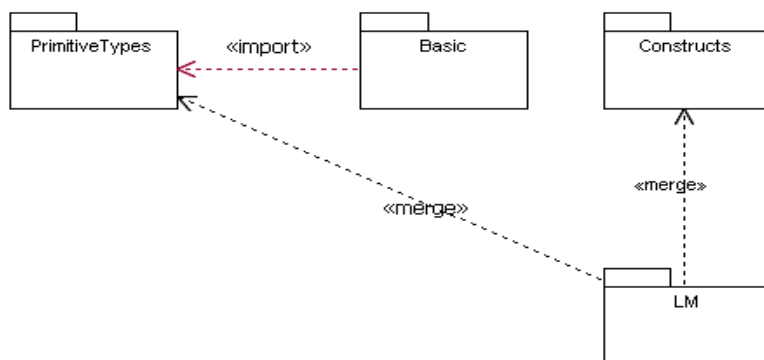


Figure 2.2 - Level M package diagram

Note that LM does not explicitly merge Basic, since the elements in Basic are already incorporated into the corresponding elements in Constructs.

2.4 Meaning and Types of Compliance

Compliance to a given level entails full realization of *all language units* that are defined for that compliance level. This also implies full realization of all language units in all the levels below that level. "Full realization" for a language unit at a given level means supporting the *complete set of modeling concepts* defined for that language unit *at that level*.

Thus, it is not meaningful to claim compliance to, say, Level 2 without also being compliant with the Level 0 and Level 1. A tool that is compliant at a given level must be able to import models from tools that are compliant to lower levels without loss of information.

There are two distinct types of compliance. They are:

- *Abstract syntax compliance.* For a given compliance level, this entails:
 - compliance with the metaclasses, their structural relationships, and any constraints defined as part of the merged UML metamodel for that compliance level, and
 - the ability to output models and to read in models based on the XMI schema corresponding to that compliance level.
- *Concrete syntax compliance.* For a given compliance level, this entails:
 - compliance to the notation defined in the “Notation” sub clauses in this part of ISO/IEC 19505 for those metamodel elements that are defined as part of the merged metamodel for that compliance level and, by implication, the diagram types in which those elements may appear; and optionally
 - the ability to output diagrams and to read in diagrams based on the XMI schema defined by the Diagram Interchange specification for notation at that level. This option requires abstract syntax and concrete syntax compliance.

Concrete syntax compliance does not require compliance to any presentation options that are defined as part of the notation.

Compliance for a given level can be expressed as:

- abstract syntax compliance
- concrete syntax compliance
- abstract syntax with concrete syntax compliance
- abstract syntax with concrete syntax and diagram interchange compliance

Table 2.1 - Example compliance statement

Compliance Summary			
Compliance level	Abstract Syntax	Concrete Syntax	Diagram Interchange Option
LO	YES	YES	NO
LM	NO	YES	NO

In case of tools that generate program code from models or those that are capable of executing models, it is also useful to understand the level of support for the run-time semantics described in the various “Semantics” sub clauses of the specification. However, the presence of numerous variation points in these semantics (and the fact that they are defined informally using natural language), make it impractical to define this as a formal compliance type, since the number of possible combinations is very large.

A similar situation exists with presentation options, since different implementers may make different choices on which ones to support. Finally, it is recognized that some implementers and profile designers may want to support only a subset of features from levels that are above their formal compliance level. (Note, however, that they can only claim compliance to the level that they fully support, even if they implement significant parts of the capabilities of higher levels.) Given this potential variability, it is useful to be able to specify clearly and efficiently, which capabilities are supported by a given implementation. To this end, in addition to a formal statement of compliance, implementers and profile designers may

also provide informal *feature support statements*. These statements identify support for additional features in terms of language units and/or individual metamodel packages, as well as for less precisely defined dimensions such as presentation options and semantic variation points.

An example feature support statement is shown in Table 2.2 for an implementation whose compliance statement is given in Table 2.1. In this case, the implementation adds two new language units from higher levels.

Table 2.2 - Example feature support statement

Feature Support Statement	
Language Unit	Feature
Constructs	An Association A1 specializes another Association A2 if each end of A1 subsets the corresponding end of A2.
Constructs	A redefining property must have the same name as the redefined property.

2.5 Compliance Level Contents

Table 2.3 identifies the packages by individual compliance levels in addition to those that are defined in lower levels (as a rule, Level (N) includes all the packages supported by Level (N-1)). The set of actual modeling features added by each of the packages are described in the appropriate clauses of the related language unit.

Table 2.3 - Metamodel packages added to compliance levels

Level	Metamodel Package Added
L0	Basic
LM	Constructs

3 Normative References

The following normative documents contain provisions which, through reference in this text, constitute provisions of this part of ISO/IEC 19505. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply.

- RFC2119, <http://ietf.org/rfc/rfc2119>, Key words for use in RFCs to Indicate Requirement Levels, S. Bradner, March 1997.
- ISO/IEC 19505-2, Information technology — OMG Unified Modeling Language (OMG UML) Version 2.4.1 — Part 2: Superstructure; pas/2011-08-12
- OMG Specification formal/2011-08-06, UML Superstructure, v2.4.1
- OMG Specification formal/2010-02-01, Object Constraint Language, v2.2
- OMG Specification formal/2011-08-07, Meta Object Facility (MOF) Core, v2.4.1
- OMG Specification formal/2011-08-09, XML Metadata Interchange (XMI), v2.4.1

ISO/IEC 19505-1:2012(E)

- OMG Specification formal/06-04-04 , UML 2.0 Diagram Interchange

Note – UML 2 is based on a different generation of MOF and XMI than that specified in ISO/IEC 19502:2005 Information technology - Meta Object Facility (MOF) and ISO/IEC 19503:2005 Information technology - XML Metadata Interchange (XMI) that are compatible with ISO/IEC 19501 UML version 1.4.1.