BS ISO/IEC 19505-1:2012

BSI Standards Publication

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

Part 1: Infrastructure

bsi.

...making excellence a habit.™

Document Sharing Hub

Full document details: BS ISO/IEC 19505-1:2012

National foreword

This British Standard is the UK implementation of ISO/IEC 19505-1:2012.

The UK participation in its preparation was entrusted to Technical Committee IST/15, Software and systems engineering.

A list of organizations represented on this committee can be obtained on request to its secretary.

This publication does not purport to include all the necessary provisions of a contract. Users are responsible for its correct application.

© The British Standards Institution 2012

Published by BSI Standards Limited 2012

ISBN 978 0 580 65509 8

ICS 35.060

Compliance with a British Standard cannot confer immunity from legal obligations.

This British Standard was published under the authority of the Standards Policy and Strategy Committee on 30 November 2012.

Amendments issued since publication

Amd. No.	Date	Text affected

Documents
Document Sharing Hub

Full document details: BS ISO/IEC 19505-1:2012

INTERNATIONAL STANDARD

ISO/IEC 19505-1

First edition
2012-04-15

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

Reference number
ISO/IEC 19505-1:2012(E)

Documents
Document Sharing Hub

© ISO/IEC 2012

Full document details: BS ISO/IEC 19505-1:2012

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

© ISO/IEC 2012 – All rights reserved

Documents
Document Sharing Hub

Full document details: BS ISO/IEC 19505-1:2012

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

© ISO/IEC 2012 - All rights reserved

AVA Documents
Document Sharing Hub

Full document details: BS ISO/IEC 19505-1:2012

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

© ISO/IEC 2012 – All rights reserved
AVA Documents
Document Sharing Hub
Full document details: BS ISO/IEC 19505-1:2012

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

© ISO/IEC 2012 - All rights reserved

Documents
Document Sharing Hub

Full document details: BS ISO/IEC 19505-1:2012

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

© ISO/IEC 2012 – All rights reserved
AVA Documents
Document Sharing Hub
Full document details: BS ISO/IEC 19505-1:2012

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

© ISO/IEC 2012 - All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 19505-1:2012

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

© ISO/IEC 2012 – All rights reserved
AVA Documents
Document Sharing Hub

Full document details: BS ISO/IEC 19505-1:2012

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.

© ISO/IEC 2012 - All rights reserved

Documents
Document Sharing Hub

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).

© ISO/IEC 2012 – All rights reserved
AVA Documents
Document Sharing Hub
Full document details: BS ISO/IEC 19505-1:2012

INTERNATIONAL STANDARD ISO/IEC 19505-1:2012 (E)

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.

© ISO/IEC 2012 - All rights reserved

Documents
Document Sharing Hub

Full document details: BS ISO/IEC 19505-1:2012

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.

© ISO/IEC 2012 - All rights reserved
AVA Documents
Document Sharing Hub
Full document details: BS ISO/IEC 19505-1:2012

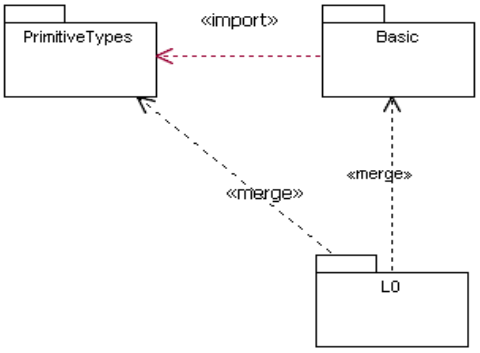

Figure 2.1 - Level 0 package diagram

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.

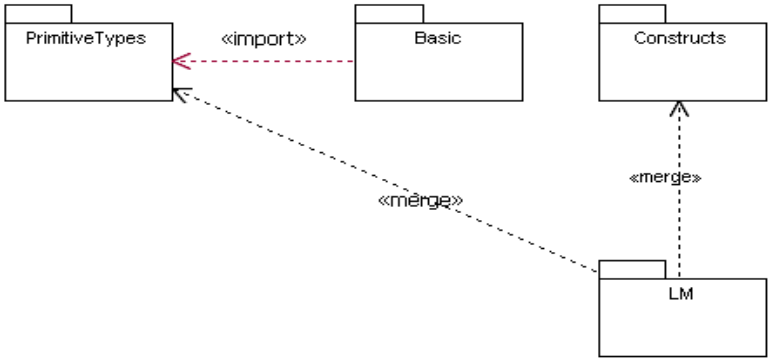

Figure 2.2 - Level M package diagram

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.

© ISO/IEC 2012 - All rights reserved

Documents
Document Sharing Hub

Full document details: BS ISO/IEC 19505-1:2012

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**
L0	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

© ISO/IEC 2012 - All rights reserved
AVA Documents
Document Sharing Hub
Full document details: BS ISO/IEC 19505-1:2012

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 2012 - All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 19505-1:2012

- 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.

4 Terms and Definitions

There are no formal definitions in this part of ISO/IEC 19505 that are taken from other documents.

5 Notational Conventions

The keywords “must,” “must not,” “shall,” “shall not,” “should,” “should not,” and “may” in this part of ISO/IEC 19505 are to be interpreted as described in RFC 2119.

6 Additional Information

6.1 Architectural Alignment and MDA Support

Clause 7, “Language Architecture,” explains how the *UML 2: Infrastructure* is architecturally aligned with the *UML 2: Superstructure* that complements it. It also explains how the InfrastructureLibrary defined in the *UML 2: Infrastructure* can be strictly reused by MOF 2 specifications.

The *MOF 2: Core* Specification is architecturally aligned with this part of ISO/IEC 19505.

The OMG’s Model Driven Architecture (MDA) initiative is an evolving conceptual architecture for a set of industry-wide technology specifications that will support a model-driven approach to software development. Although MDA is not itself a technology specification, it represents an important approach and a plan to achieve a cohesive set of model-driven technology specifications. This International Standard’s support for MDA is discussed in *Annex B: “Support for Model Driven Architecture,” on page 213*.

6.2 How to Proceed

The rest of this document contains the technical content of this part of ISO/IEC 19505. Readers are encouraged to first read “Subpart I - Introduction” to familiarize themselves with the structure of the language and the formal approach used for its specification. Afterwards the reader may choose to either explore the InfrastructureLibrary, described in “Subpart II - Infrastructure Library” or the UML::Classes::Kernel package that reuses it, described in the *UML 2: Superstructure*. The former specifies the flexible metamodel library that is reused by the latter.

Readers who want to explore the user level constructs that are built upon the infrastructural constructs specified here should investigate the specification that complements this, the *UML 2: Superstructure*.

© ISO/IEC 2012 - All rights reserved
AVA Documents
Document Sharing Hub
Full document details: BS ISO/IEC 19505-1:2012

Although the clauses are organized in a logical manner and can be read sequentially, this is a reference specification intended to be read in a non-sequential manner. Consequently, extensive cross-references are provided to facilitate browsing and search.

6.2.1 Diagram format

The following conventions are adopted for all metamodel diagrams throughout this part of ISO/IEC 19505:

- An association with one end marked by a navigability arrow means that:
  - the association is navigable in the direction of that end,
  - the marked association end is owned by the classifier, and
  - the opposite (unmarked) association end is owned by the association.
- An association with neither end marked by navigability arrows means that:
  - the association is navigable in both directions,
  - each association end is owned by the classifier at the opposite end (i.e., neither end is owned by the association).
- Association specialization and redefinition are indicated by appropriate constraints situated in the proximity of the association ends to which they apply. Thus:
  - the constraint {subsets endA} means that the association end to which this constraint is applied is a specialization of association end endA that is part of the association being specialized.
  - a constraint {redefines endA} means that the association end to which this constraint is applied redefines the association end endA that is part of the association being specialized.
- If no multiplicity is shown on an association end, it implies a multiplicity of exactly 1.
- If an association end is unlabeled, the default name for that end is the name of the class to which the end is attached, modified such that the first letter is a lowercase letter. (Note that, by convention, non-navigable association ends are often left unlabeled since, in general, there is no need to refer to them explicitly either in the text or in formal constraints – although they may be needed for other purposes, such as MOF language bindings that use the metamodel.)
- Associations that are not explicitly named, are given names that are constructed according to the following production rule:

  "A_" <association-end-name1> "_" <association-end-name2>

  where *<association-end-name1>* is the name of the first association end and *<association-end-name2>* is the name of the second association end.
- An unlabeled dependency between two packages is interpreted as a package import relationship.

Note that some of these conventions were adopted to contend with practical issues related to the mechanics of producing this International Standard, such as the unavailability of conforming modeling tools at the time the specification itself was being defined. Therefore, they should not necessarily be deemed as recommendations for general use.

© ISO/IEC 2012 - All rights reserved

Documents
Document Sharing Hub

Full document details: BS ISO/IEC 19505-1:2012

6.2.2 Contents of Subparts I and II

6.2.2.1 Contents of Subpart I - Introduction

The Unified Modeling Language is a visual language for specifying, constructing, and documenting the artifacts of systems. It is a general-purpose modeling language that can be used with all major object and component methods, and that can be applied to all application domains (e.g., health, finance, telecom, aerospace) and implementation platforms (e.g., J2EE, .NET).

The OMG adopted the UML 1.1 specification in November 1997. Since then UML Revision Task Forces have produced several minor revisions, the most recent being the UML 1.4 specification, which was adopted in May 2001.

Under the stewardship of the OMG, the UML has emerged as the software industry's dominant modeling language. It has been successfully applied to a wide range of domains, ranging from health and finance to aerospace to e-commerce. As should be expected, its extensive use has raised numerous application and implementation issues by modelers and vendors. As of the time of this writing over 500 formal usage and implementation issues have been submitted to the OMG for consideration.

Although many of the issues have been resolved in minor revisions by Revision Task Forces, other issues require major changes to the language that are outside the scope of an RTF. Consequently, the OMG issued four complementary and architecturally aligned RFPs to define UML: UML Infrastructure, UML Superstructure, UML Object Constraint Language, and UML Diagram Interchange.

This UML specification is organized into two volumes (*UML 2: Infrastructure* and *UML 2: Superstructure*), consistent with the breakdown of modeling language requirements into two RFPs (*UML Infrastructure RFP* and *UML Superstructure RFP*). Since the two volumes cross-reference each other and the specifications are fully integrated, these two volumes could easily be combined into a single volume at a later time.

6.2.2.2 Contents of Subpart II - Infrastructure Library

This subpart includes the following clauses:

9 - Core::Abstractions

10 - Core::Basic

11 - Core::Constructs

12 - Core::Profiles

13 - Primitive Types

This describes the structure and contents of the Infrastructure packages for the UML metamodel and related metamodels, such as the Meta Object Facility (MOF) and the Common Warehouse Metamodel (CWM). The first top level package is *InfrastructureLibrary*, which defines a reusable metalanguage kernel and a metamodel extension mechanism for UML. The metalanguage kernel can be used to specify a variety of metamodels, including UML, MOF, and CWM. In addition, the library defines a profiling extension mechanism that can be used to customize UML for different platforms and domains without supporting a complete metamodeling capability. The nested packages of the *InfrastructureLibrary* are *Core* and *Profile*. The other top level package is *PrimitiveTypes*, which consists of a small number of primitive types that are commonly used for metamodeling. The *PrimitiveTypes* package is imported by nested packages in the *InfrastructureLibrary* and can be imported by other packages, libraries and metamodels that need to define primitive data. The high level architecture of the Infrastructure packages is shown below.

© ISO/IEC 2012 - All rights reserved
AVA Documents
Document Sharing Hub
Full document details: BS ISO/IEC 19505-1:2012

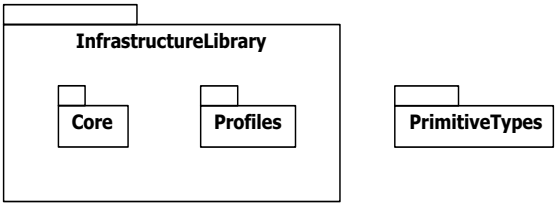

Figure 6.1 - The Metamodel Library package contains the packages Core and Profiles

The *Core* package is the central reusable part of the InfrastructureLibrary, and is further subdivided as shown in the figure below.

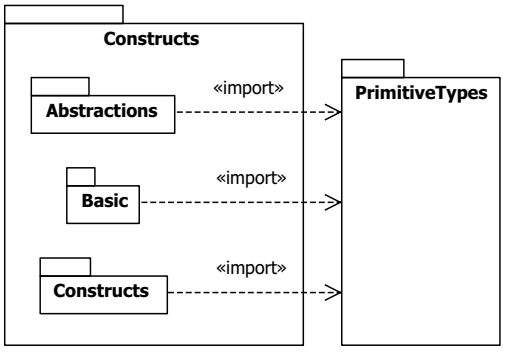

Figure 6.2 - The Core package contains the packages PrimitiveTypes, Abstractions, Basic, and Constructs

The package *PrimitiveTypes* is a simple package that contains a number of predefined types that are commonly used when metamodeling, and as such they are used both in the infrastructure library itself, but also in metamodels like MOF and UML. The package *Abstractions* contains a number of fine-grained packages with only a few metaclasses each, most of which are abstract. The purpose of this package is to provide a highly reusable set of metaclasses to be specialized when defining new metamodels. The package *Constructs* also contains a number of fine-grained packages, and brings together many of the aspects of the *Abstractions*. The metaclasses in *Constructs* tend to be concrete rather than abstract, and are geared towards an object-oriented modeling paradigm. Looking at metamodels such as MOF and UML, they typically import the *Constructs* package since the contents of the other packages of Core are then automatically included. The package *Basic* contains a subset of *Constructs* that is used primarily for XMI purposes.

The *Profiles* package contains the mechanisms used to create profiles of specific metamodels, and in particular of UML. This extension mechanism subsets the capabilities offered by the more general MOF extension mechanism.

The detailed structure and contents of the *PrimitiveTypes*, *Abstractions*, *Basic*, *Constructs*, and *Profiles* packages are further described in subsequent clauses.

© ISO/IEC 2012 - All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 19505-1:2012

© ISO/IEC 2012 – All rights reserved
Document Sharing Hub
Full document details: BS ISO/IEC 19505-1:2012

Subpart I - Introduction

© ISO/IEC 2012 - All rights reserved

Documents
Document Sharing Hub

Full document details: BS ISO/IEC 19505-1:2012

© ISO/IEC 2012 – All rights reserved
Document Sharing Hub
Full document details: BS ISO/IEC 19505-1:2012

7 Language Architecture

7.1 General

The UML specification is defined using a metamodeling approach (i.e., a metamodel is used to specify the model that comprises UML) that adapts formal specification techniques. While this approach lacks some of the rigor of a formal specification method, it offers the advantages of being more intuitive and pragmatic for most implementers and practitioners.[1] This clause explains the architecture of the UML metamodel.

The following sub clauses summarize the design principles followed, and show how they are applied to organize UML's Infrastructure and Superstructure. The last sub clause explains how the UML metamodel conforms to a 4-layer metamodel architectural pattern.

7.2 Design Principles

The UML metamodel has been architected with the following design principles in mind:

- Modularity — This principle of strong cohesion and loose coupling is applied to group constructs into packages and organize features into metaclasses.
- Layering — Layering is applied in two ways to the UML metamodel. First, the package structure is layered to separate the metalanguage core constructs from the higher-level constructs that use them. Second, a 4-layer metamodel architectural pattern is consistently applied to separate concerns (especially regarding instantiation) across layers of abstraction.
- Partitioning — Partitioning is used to organize conceptual areas within the same layer. In the case of the InfrastructureLibrary, fine-grained partitioning is used to provide the flexibility required by current and future metamodeling standards. In the case of the UML metamodel, the partitioning is coarser-grained in order to increase the cohesion within packages and loosening the coupling across packages.
- Extensibility — The UML can be extended in two ways:
  - A new dialect of UML can be defined by using Profiles to customize the language for particular platforms (e.g., J2EE/EJB, .NET/COM+) and domains (e.g., finance, telecommunications, aerospace).
  - A new language related to UML can be specified by reusing part of the InfrastructureLibrary package and augmenting with appropriate metaclasses and metarelationships. The former case defines a new dialect of UML, while the latter case defines a new member of the UML family of languages.
- Reuse — A fine-grained, flexible metamodel library is provided that is reused to define the UML metamodel, as well as other architecturally related metamodels, such as the Meta Object Facility (MOF) and the Common Warehouse Metamodel (CWM).

7.3 Infrastructure Architecture

The Infrastructure of the UML is defined by the InfrastructureLibrary, which satisfies the following design requirements:

---

1. It is important to note that the specification of UML as a metamodel does not preclude it from being specified via a mathematically formal language (e.g., Object-Z or VDM) at a later time.

© ISO/IEC 2012 - All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 19505-1:2012

- Define a metalanguage core that can be reused to define a variety of metamodels, including UML, MOF, and CWM.
- Architecturally align UML, MOF, and XMI so that model interchange is fully supported.
- Allow customization of UML through Profiles and creation of new languages (family of languages) based on the same metalanguage core as UML.

As shown in Figure 7.1, Infrastructure is represented by two packages: *InfrastructureLibrary* and *PrimitiveTypes*. The package *InfrastructureLibrary* consists of the packages *Core* and *Profiles*, where the latter defines the mechanisms that are used to customize metamodels and the former contains core concepts used when metamodeling. The package *PrimitiveTypes* consists of a few predefined primitive types that are commonly used when metamodeling, and is designed specifically with the needs of UML and MOF in mind.

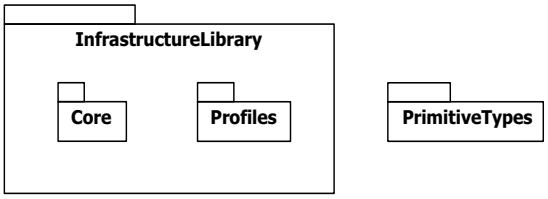

Figure 7.1 - The InfrastructureLibrary packages

7.4 Core

In its first capacity, the *Core* package is a complete metamodel particularly designed for high reusability, where other metamodels at the same metalevel (see 7.7, "Superstructure Architecture," on page 17) either import or specialize its specified metaclasses. This is illustrated in Figure 7.2, where it is shown how UML, CWM, and MOF each depends on a *common* core. Since these metamodels are at the very heart of the Model Driven Architecture (MDA), the common core may also be considered the architectural kernel of MDA. The intent is for UML and other MDA metamodels to reuse all or parts of the *Core* package, which allows other metamodels to benefit from the abstract syntax and semantics that have already been defined.

© ISO/IEC 2012 - All rights reserved
AVA Documents
Document Sharing Hub
Full document details: BS ISO/IEC 19505-1:2012

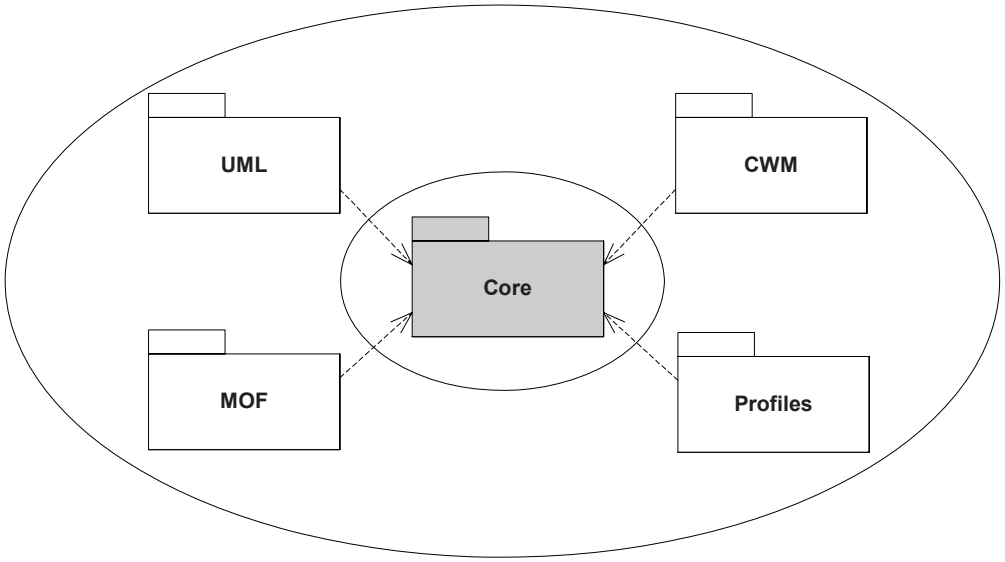

Figure 7.2 - The role of the common Core

In order to facilitate reuse, the *Core* package is subdivided into a number of packages: *Abstractions*, *Basic*, and *Constructs* as shown in Figure 7.3. As we will see in subsequent clauses, some of these are then further divided into even more fine-grained packages to make it possible to pick and choose the relevant parts when defining a new metamodel. Note, however, that choosing a specific package also implies choosing the dependent packages. There are minor differences in the design rationale for the other three packages. The package *Abstractions* mostly contains abstract metaclasses that are intended to be further specialized or that are expected to be commonly reused by many metamodels. Very few assumptions are made about the metamodels that may want to reuse this package; for this reason, the package *Abstractions* is also subdivided into several smaller packages. The package *Constructs*, on the other hand, mostly contains concrete metaclasses that lend themselves primarily to object-oriented modeling; this package in particular is reused by both MOF and UML, and represents a significant part of the work that has gone into aligning the two metamodels. The package *Basic* represents a few constructs that are used as the basis for the produced XMI for UML, MOF, and other metamodels based on the *InfrastructureLibrary.*

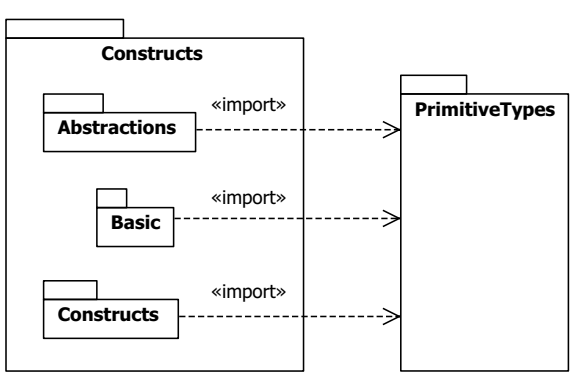

Figure 7.3 - The Core packages

© ISO/IEC 2012 - All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 19505-1:2012

In its second capacity, the *Core* package is used to define the modeling constructs used to create metamodels. This is done through instantiation of metaclasses in the *InfrastructureLibrary* (see 7.10, "Metamodel Layering," on page 18). While instantiation of metaclasses is carried out through MOF, the *InfrastructureLibrary* defines the actual metaclasses that are used to instantiate the elements of UML, MOF, CWM, and indeed the elements of the *InfrastructureLibrary* itself. In this respect, the *InfrastructureLibrary* is said to be self-describing, or *reflective*.

7.5 Profiles

As was depicted in Figure 7.1, the *Profiles* package depends on the *Core* package, and defines the mechanisms used to tailor existing metamodels towards specific platforms, such as C++, CORBA, or EJB; or domains such as real-time, business objects, or software process modeling. The primary target for profiles is UML, but it is possible to use profiles together with any metamodel that is based on (i.e., instantiated from) the common core. A profile must be based on a metamodel such as the UML that it extends, and is not very useful standalone.

Profiles have been aligned with the extension mechanism offered by MOF, but provide a more light-weight approach with restrictions that are enforced to ensure that the implementation and usage of profiles should be straightforward and more easily supported by tool vendors.

7.6 Architectural Alignment between UML and MOF

One of the major goals of the Infrastructure has been to architecturally align UML and MOF. The first approach to accomplish this has been to define the common core, which is realized as the package *Core*, in such a way that the model elements are shared between UML and MOF. The second approach has been to make sure that UML is defined as a model that is based on MOF used as a metamodel, as is illustrated in Figure 7.4. Note that MOF is used as the metamodel for not only UML, but also for other languages such as CWM.

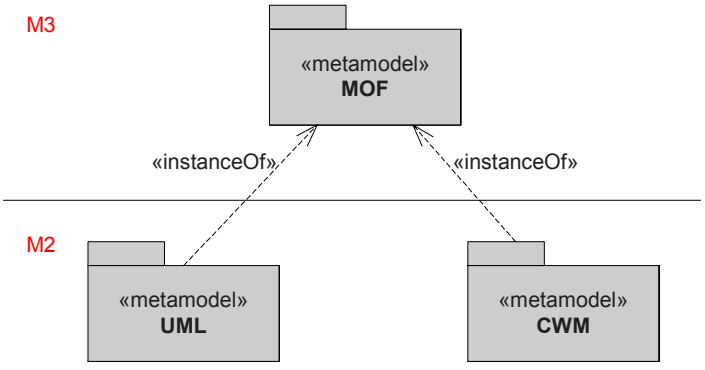

Figure 7.4 - UML and MOF are at different metalevels

How these metalevel hierarchies work is explained in more detail in 7.7, "Superstructure Architecture," on page 17. An important aspect that deserves mentioning here is that every model element of UML is an instance of exactly one model element in MOF. Note that the *InfrastructureLibrary* is used at both the M2 and M3 metalevels, since it is being reused by UML and MOF, respectively, as was shown in Figure 7.2. In the case of MOF, the metaclasses of the *InfrastructureLibrary* are used as is, while in the case of UML these model elements are given additional properties. The reason for these differences is that the requirements when metamodeling differ slightly from the requirements when modeling applications of a very diverse nature.

© ISO/IEC 2012 - All rights reserved
AVA Documents
Document Sharing Hub
Full document details: BS ISO/IEC 19505-1:2012

MOF defines for example how UML models are interchanged between tools using XML Metadata Interchange (XMI). MOF also defines reflective interfaces (MOF::Reflection) for introspection that work for not only MOF itself, but also for CWM, UML, and for any other metamodel that is an instance of MOF. It further defines an extension mechanism that can be used to extend metamodels as an alternative to or in conjunction with profiles (as described in Clause 13, "Core::Profiles"). In fact, profiles are defined to be a subset of the MOF extension mechanism.

7.7 Superstructure Architecture

The UML Superstructure metamodel is specified by the *UML* package, which is divided into a number of packages that deal with structural and behavioral modeling, as shown in Figure 7.5.

Each of these areas is described in a separate clause of the *UML 2: Superstructure* specification. Note that there are some packages that are dependent on each other in circular dependencies. This is because the dependencies between the top-level packages show a summary of all relationships between their subpackages; there are no circular dependencies between subpackages of those packages.

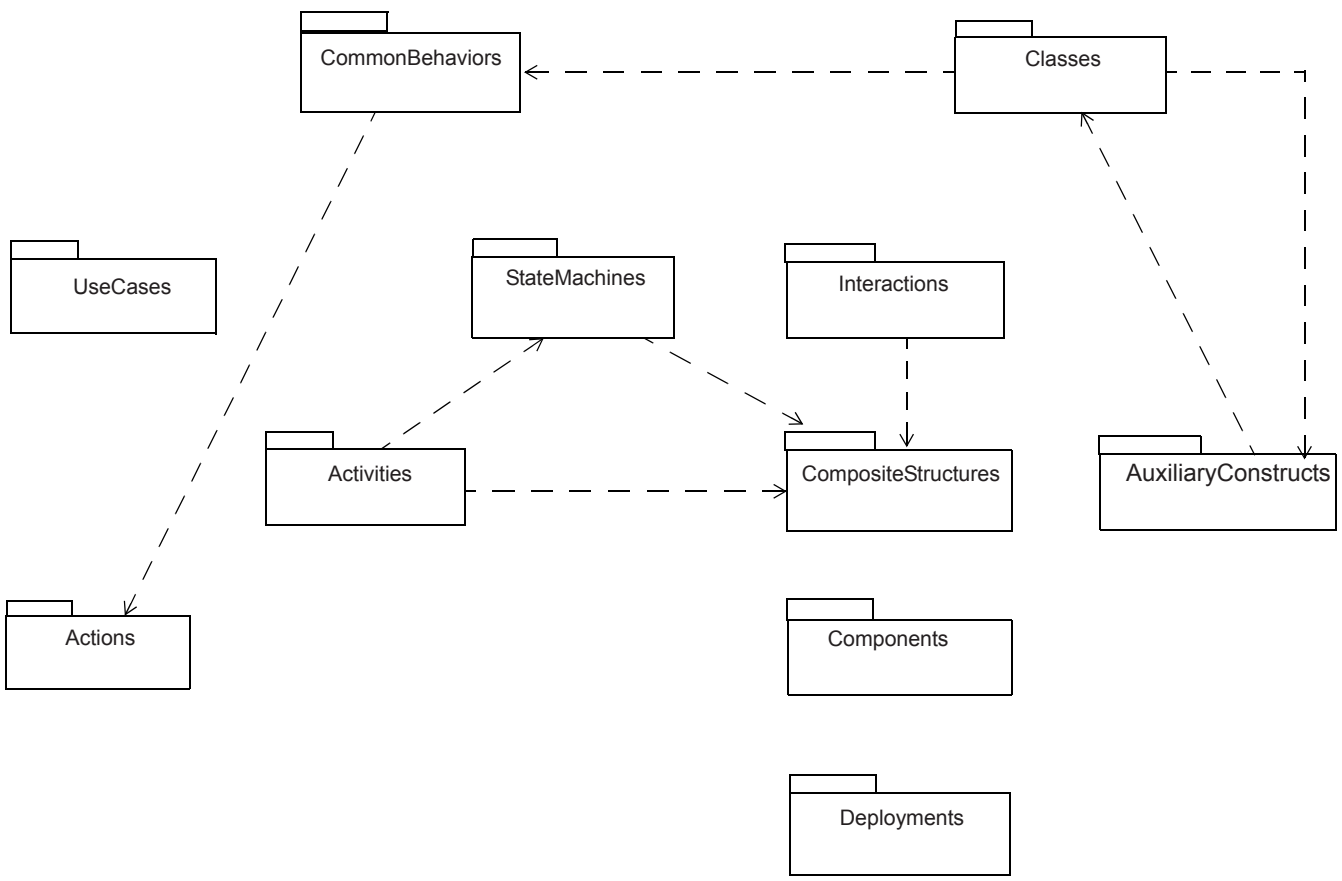

Figure 7.5 - The top-level package structure of the UML 2 Superstructure

© ISO/IEC 2012 - All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 19505-1:2012

7.8 Reusing Infrastructure

One of the primary uses of the UML 2 Infrastructure specification is that it should be reused when creating other metamodels. The UML metamodel reuses the *InfrastructureLibrary* in two different ways:

- All of the UML metamodel is instantiated from meta-metaclasses that are defined in the *InfrastructureLibrary.*
- The UML metamodel imports and specializes all metaclasses in the *InfrastructureLibrary.*

As was discussed earlier, it is possible for a model to be used as a metamodel, and here we make use of this fact. The *InfrastructureLibrary* is in one capacity used as a meta-metamodel and in the other aspect as a metamodel, and is thus reused in two dimensions.

7.9 The Kernel Package

The *InfrastructureLibrary* is primarily reused in the *Kernel* package of *Classes* in *UML 2: Superstructure*; this is done by bringing together the different packages of the Infrastructure using package merge. The *Kernel* package is at the very heart of UML, and the metaclasses of every other package are directly or indirectly dependent on it. The *Kernel* package is very similar to the *Constructs* package of the *InfrastructureLibrary*, but adds more capabilities to the modeling constructs that were not necessary to include for purposes of reuse or alignment with MOF.

Because the Infrastructure has been designed for reuse, there are metaclasses—particularly in *Abstractions*—that are partially defined in several different packages. These different aspects are for the most part brought together into a single metaclass already in *Constructs*, but in some cases this is done only in *Kernel*. In general, if metaclasses with the same name occur in multiple packages, they are meant to represent the same metaclass, and each package where it is defined (specialized) represents a specific factorization. This same pattern of partial definitions also occurs in Superstructure, where some aspects of, for example, the metaclass *Class* are factored out into separate packages to form compliance points (see below).

7.10 Metamodel Layering

The architecture that is centered around the *Core* package is a complementary view of the four-layer metamodel hierarchy on which the UML metamodel has traditionally been based. When dealing with meta-layers to define languages there are generally three layers that always have to be taken into account:

1. the language specification, or the metamodel,
2. the user specification, or the model, and
3. objects of the model.

This structure can be applied recursively many times so that we get a possibly infinite number of meta-layers; what is a metamodel in one case can be a model in another case, and this is what happens with UML and MOF. UML is a language specification (metamodel) from which users can define their own models. Similarly, MOF is also a language specification (metamodel) from which users can define their own models. From the perspective of MOF, however, UML is viewed as a user (i.e., the members of the OMG that have developed the language) specification that is based on MOF as a language specification. In the four-layer metamodel hierarchy, MOF is commonly referred to as a meta-metamodel, even though strictly speaking it is a metamodel.

© ISO/IEC 2012 – All rights reserved
AVA Documents
Document Sharing Hub
Full document details: BS ISO/IEC 19505-1:2012

7.11 The Four-layer Metamodel Hierarchy

The meta-metamodeling layer forms the foundation of the metamodeling hierarchy. The primary responsibility of this layer is to define the language for specifying a metamodel. The layer is often referred to as M3, and MOF is an example of a meta-metamodel. A meta-metamodel is typically more compact than a metamodel that it describes, and often defines several metamodels. It is generally desirable that related metamodels and meta-metamodels share common design philosophies and constructs. However, each layer can be viewed independently of other layers, and needs to maintain its own design integrity.

A metamodel is an instance of a meta-metamodel, meaning that every element of the metamodel is an instance of an element in the meta-metamodel. The primary responsibility of the metamodel layer is to define a language for specifying models. The layer is often referred to as M2; UML and the OMG Common Warehouse Metamodel (CWM) are examples of metamodels. Metamodels are typically more elaborate than the meta-metamodels that describe them, especially when they define dynamic semantics. The UML metamodel is an instance of the MOF (in effect, each UML metaclass is an instance of an element in *InfrastructureLibrary*).

A model is an instance of a metamodel. The primary responsibility of the model layer is to define languages that describe semantic domains, i.e., to allow users to model a wide variety of different problem domains, such as software, business processes, and requirements. The things that are being modeled reside outside the metamodel hierarchy. This layer is often referred to as M1. A user model is an instance of the UML metamodel. Note that the user model contains both model elements and snapshots (illustrations) of instances of these model elements.

The metamodel hierarchy bottoms out at M0, which contains the run-time instances of model elements defined in a model. The snapshots that are modeled at M1 are constrained versions of the M0 run-time instances.

When dealing with more than three meta-layers, it is usually the case that the ones above M2 gradually get smaller and more compact the higher up they are in the hierarchy. In the case of MOF, which is at M3, it consequently only shares some of the metaclasses that are defined in UML. A specific characteristic about metamodeling is the ability to define languages as being reflective, i.e., languages that can be used to define themselves. The *InfrastructureLibrary* is an example of this, since it contains all the metaclasses required to define itself. MOF is reflective since it is based on the *InfrastructureLibrary*. This allows it to be used to define itself. For this reason, no additional meta-layers above MOF are defined.

7.12 Metamodeling

When metamodeling, we primarily distinguish between metamodels and models. As already stated, a model that is instantiated from a metamodel can in turn be used as a metamodel of another model in a recursive manner. A model typically contains model elements. These are created by instantiating model elements from a metamodel, i.e., metamodel elements.

The typical role of a metamodel is to define the semantics for how model elements in a model get instantiated. As an example, consider Figure 7.6, where the metaclasses Association and Class are both defined as part of the UML metamodel. These are instantiated in a user model in such a way that the classes Person and Car are both instances of the metaclass Class, and the association Person.car between the classes is an instance of the metaclass Association. The semantics of UML defines what happens when the user defined model elements are instantiated at M0, and we get an instance of Person, an instance of Car, and a link (i.e., an instance of the association) between them.

© ISO/IEC 2012 - All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 19505-1:2012

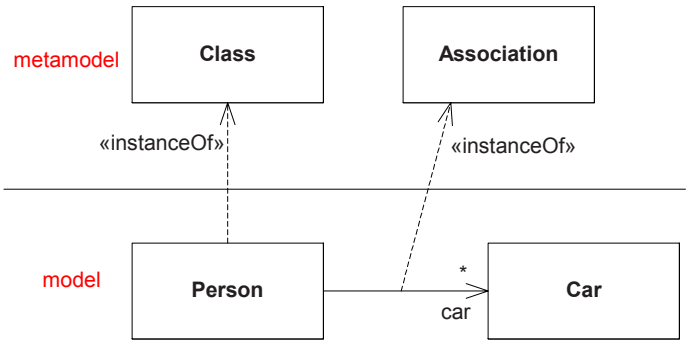

Figure 7.6 - An example of metamodeling; note that not all instance-of relationships are shown

The instances, which are sometimes referred to as “run-time” instances, that are created at M0 from for example Person should not be confused with instances of the metaclass InstanceSpecification that are also defined as part of the UML metamodel. An instance of an InstanceSpecification is defined in a model at the same level as the model elements that it illustrates, as is depicted in Figure 7.7, where the instance specification Mike is an illustration (or a snapshot) of an instance of class Person.

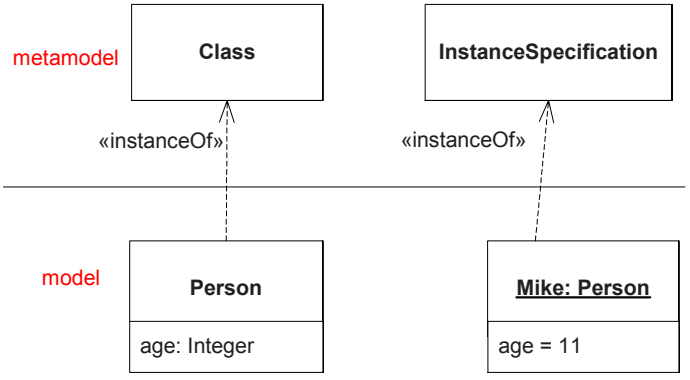

Figure 7.7 - Giving an illustration of a class using an instance specification

7.13 An Example of the Four-level Metamodel Hierarchy

An illustration of how these meta-layers relate to each other is shown in Figure 7.8. It should be noted that we are by no means restricted to only these four meta-layers, and it would be possible to define additional ones. As is shown, the meta-layers are usually numbered from M0 and upwards, depending on how many meta-layers are used. In this particular case, the numbering goes up to M3, which corresponds to MOF.

© ISO/IEC 2012 - All rights reserved
AVA Documents
Document Sharing Hub
Full document details: BS ISO/IEC 19505-1:2012

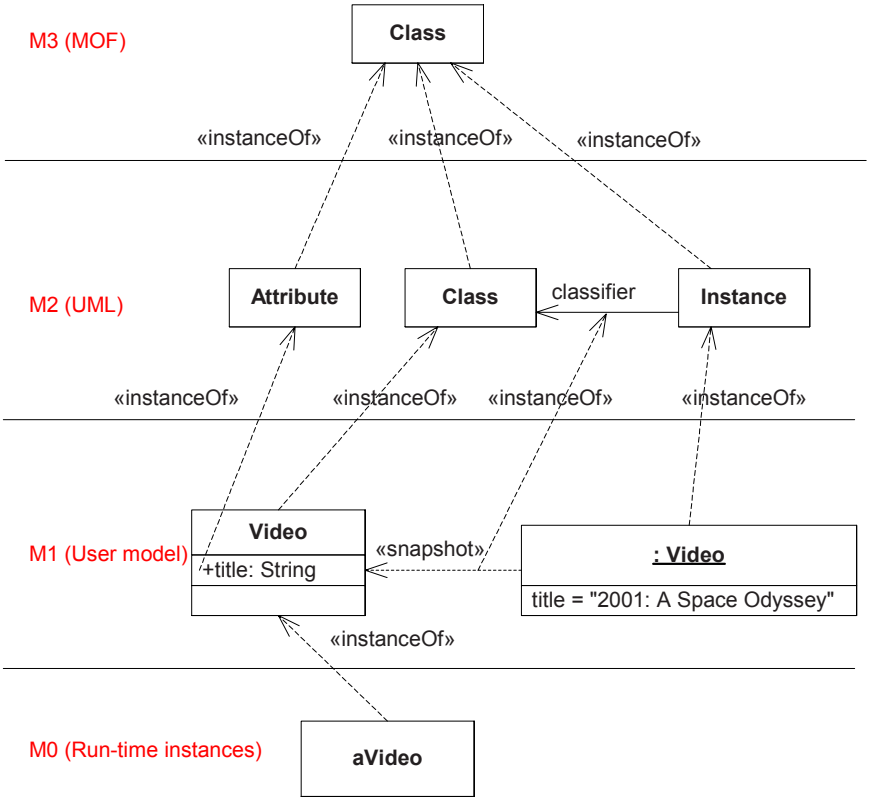

Figure 7.8 - An example of the four-layer metamodel hierarchy

© ISO/IEC 2012 - All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 19505-1:2012

© ISO/IEC 2012 – All rights reserved
Document Sharing Hub
Full document details: BS ISO/IEC 19505-1:2012

8 Language Formalism

8.1 General

The UML specification is defined by using a metamodeling approach that adapts formal specification techniques. The formal specification techniques are used to increase the precision and correctness of the specification. This clause explains the specification techniques used to define UML.

The following are the goals of the specification techniques used to define UML:

- Correctness — The specification techniques should improve the correctness of the metamodel by helping to validate it. For example, the well-formedness rules should help validate the abstract syntax and help identify errors.
- Precision — The specification techniques should increase the precision of both the syntax and semantics. The precision should be sufficient so that there is no syntactic nor semantic ambiguity for either implementors or users.[1]
- Conciseness — The specification techniques should be parsimonious, so that the precise syntax and semantics are defined without superfluous detail.
- Consistency — The specification techniques should complement the metamodeling approach by adding essential detail in a consistent manner.
- Understandability — While increasing the precision and conciseness, the specification techniques should also improve the readability of the specification. For this reason a less than strict formalism is applied, since a strict formalism would require formal techniques.

The specification technique used describes the metamodel in three views using both text and graphic presentations.

It is important to note that the current description is not a completely formal specification of the language because to do so would have added significant complexity without clear benefit.

The structure of the language is nevertheless given a precise specification, which is required for tool interoperability. The detailed semantics are described using natural language, although in a precise way so they can easily be understood. Currently, the semantics are not considered essential for the development of tools; however, this will probably change in the future.

8.2 Levels of Formalism

A common technique for specification of languages is to first define the syntax of the language and then to describe its static and dynamic semantics. The syntax defines what constructs exist in the language and how the constructs are built up in terms of other constructs. Sometimes, especially if the language has a graphic syntax, it is important to define the syntax in a notation independent way (i.e., to define the abstract syntax of the language). The concrete syntax is then defined by mapping the notation onto the abstract syntax.

The static semantics of a language define how an instance of a construct should be connected to other instances to be meaningful, and the dynamic semantics define the meaning of a well formed construct. The meaning of a description written in the language is defined only if the description is well formed (i.e., if it fulfills the rules defined in the static semantics).

---

1. By definition semantic variation points are an exception to this.

© ISO/IEC 2012 - All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 19505-1:2012

The specification uses a combination of languages - a subset of UML, an object constraint language, and precise natural language to describe the abstract syntax and semantics of the full UML. The description is self-contained; no other sources of information are needed to read the document[2]. Although this is a metacircular description[3], understanding this document is practical since only a small subset of UML constructs are needed to describe its semantics.

In constructing the UML metamodel different techniques have been used to specify language constructs, using some of the capabilities of UML. The main language constructs are reified into metaclasses in the metamodel. Other constructs, in essence being variants of other ones, are defined as stereotypes of metaclasses in the metamodel. This mechanism allows the semantics of the variant construct to be significantly different from the base metaclass. Another more "lightweight" way of defining variants is to use metaattributes. As an example, the aggregation construct is specified by an attribute of the metaclass Property, which is used to indicate if an association is an ordinary aggregate, a composite aggregate, or a common association.

8.3 Package Specification Structure

This sub clause provides information for each package and each class in the UML metamodel. Each package has one or more of the following sub clauses.

8.3.1 Class Descriptions

The sub clause contains an enumeration of the classes specifying the constructs defined in the package. It begins with one diagram or several diagrams depicting the abstract syntax of the constructs (i.e., the classes and their relationships) in the package, together with some of the well-formedness requirements (multiplicity and ordering). Then follows a specification of each class in alphabetic order (see below).

8.3.2 Diagrams

If a specific kind of diagram usually presents the constructs that are defined in the package, a sub clause describing this kind of diagram is included.

8.3.3 Instance Model

An example may be provided to show how an instance model of the contained classes may be populated. The elements in the example are instances of the classes contained in the package (or in an imported package).

8.4 Class Specification Structure

The specification of a class starts with a presentation of the general meaning of the concept that sets the context for the definition.

---

2. Although a comprehension of the UML's four-layer metamodel architecture and its underlying meta-metamodel is helpful, it is not essential to understand the UML semantics.
3. In order to understand the description of the UML semantics, you must understand some UML semantics.

© ISO/IEC 2012 - All rights reserved
AVA Documents
Document Sharing Hub
Full document details: BS ISO/IEC 19505-1:2012

8.4.1 Description

The sub clause includes an informal definition of the metaclass specifying the construct in UML. The sub clause states if the metaclass is abstract. This sub clause, together with the following two, constitutes a description of the abstract syntax of the construct.

8.4.2 Attributes

Each of the attributes of the class are enumerated together with a short explanation. The sub clause states if the attribute is derived, or if it is a specialization of another attribute. The multiplicity of the attribute is suppressed it defaults to '1' (default in UML).

8.4.3 Associations

The member ends of associations connected to the class are also listed in the same way. The sub clause states if the property is derived, or if it subsets or redefines another end.

8.4.4 Constraints

The well-formedness rules of the metaclass, except for multiplicity and ordering constraints that are defined in the diagram at the beginning of the package sub clause, are defined as a (possibly empty) set of invariants for the metaclass, which must be satisfied by all instances of that metaclass for the model to be meaningful. The rules thus specify constraints over attributes and associations defined in the metamodel. Most invariants are defined by OCL expressions together with an informal explanation of the expression, but in some cases invariants are expressed by other means (in exceptional cases with natural language). The statement 'No additional constraints' means that all well-formedness rules are expressed in the superclasses together with the multiplicity and type information expressed in the diagrams.

8.4.5 Additional Operations (optional)

In many cases, additional operations on the classes are needed for the OCL expressions. These are then defined in a separate sub clause after the constraints for the construct, using the same approach as the Constraints sub clause: an informal explanation followed by the OCL expression defining the operation.

8.4.6 Semantics

The meaning of a well formed construct is defined using natural language.

8.4.7 Semantic Variation Points (optional)

The term *semantic variation point* is used throughout this document to denote a part of the UML specification whose purpose in the overall specification is known but whose form or semantics may be varied in some way. The objective of a semantic variation point is to enable specialization of that part of UML for a particular situation or domain.

There are several forms in which semantic variation points appear in the standard:

- *Changeable default* — in this case, a single default specification for the semantic variation point is provided in the standard but it may be replaced. For example, the standard provides a default set of rules for specializing state

© ISO/IEC 2012 - All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 19505-1:2012

machines, but this default can be overridden (e.g., in a profile) by a different set of rules (the choice typically depends on which definition of behavioral compatibility is used).

- *Multiple choice* — in this case, the standard explicitly specifies a number of possible mutually exclusive choices, one of which may be marked as the default. Language designers may either select one of those alternatives or define a new one. An example of this type of variation point can be found in the handling of unexpected events in state machines; the choices include (a) ignoring the event (the default), (b) explicitly rejecting it, or (c) deferring it.
- *Undefined* — in this case, the standard does not provide any pre-defined specifications for the semantic variation point. For instance, the rules for selecting the method to be executed when a polymorphic operation is invoked are not defined in the standard.

8.4.8 Notation

The notation of the construct is presented in this sub clause.

8.4.9 Presentation Options (optional)

If there are different ways to show the construct (e.g., it is not necessary to show all parts of the construct in every occurrence), these possibilities are described in this sub clause.

8.4.10 Style Guidelines (optional)

Often non-normative conventions are used in representing some part of a model. For example, one such convention is to always have the name of a class in bold and centered within the class rectangle.

8.4.11 Examples (optional)

In this sub clause, examples of how the construct is to be depicted are given.

8.4.12 Rationale (optional)

If there is a reason why a construct is defined like it is, or why its notation is defined as it is, this reason is given in this sub clause.

8.4.13 Changes from UML 1.4

Here, changes compared with UML 1.4 are described and a migration approach from 1.4 to 2 is specified.

8.5 Use of a Constraint Language

The specification uses the Object Constraint Language (OCL), as defined in Clause 6, “Object Constraint Language Specification” of the UML 1.4 specification, for expressing well-formedness rules. The following conventions are used to promote readability:

- Self — which can be omitted as a reference to the metaclass defining the context of the invariant, has been kept for clarity.

© ISO/IEC 2012 – All rights reserved
Document Sharing Hub
Full document details: BS ISO/IEC 19505-1:2012

- In expressions where a collection is iterated, an iterator is used for clarity, even when formally unnecessary. The type of the iterator is usually omitted, but included when it adds to understanding.
- The 'collect' operation is left implicit where this is practical.
- The context part of an OCL constraint is not included explicitly, as it is well defined in the sub clause where the constraint appears.

8.6 Use of Natural Language

We strove to be precise in our use of natural language, in this case English. For example, the description of UML semantics includes phrases such as "X provides the ability to…" and "X is a Y." In each of these cases, the usual English meaning is assumed, although a deeply formal description would demand a specification of the semantics of even these simple phrases.

The following general rules apply:

- When referring to an instance of some metaclass, we often omit the word "instance." For example, instead of saying "a Class instance" or "an Association instance," we just say "a Class" or "an Association." By prefixing it with an "a" or "an," assume that we mean "an instance of." In the same way, by saying something like "Elements" we mean "a set (or the set) of instances of the metaclass Element."
- Every time a word coinciding with the name of some construct in UML is used, that construct is meant.
- Terms including one of the prefixes sub, super, or meta are written as one word (e.g., metamodel, subclass).

8.7 Conventions and Typography

In the description of UML, the following conventions have been used:

- When referring to constructs in UML, not their representation in the metamodel, normal text is used.
- Metaclass names that consist of appended nouns/adjectives, initial embedded capitals are used (e.g., 'ModelElement,' 'StructuralFeature').
- Names of metaassociations are written in the same manner as metaclasses (e.g., 'ElementReference').
- Initial embedded capital is used for names that consist of appended nouns/adjectives (e.g., 'ownedElement,' 'allContents').
- Boolean metaattribute names always start with 'is' (e.g., 'isAbstract').
- Enumeration types always end with "Kind" (e.g., 'AggregationKind').
- While referring to metaclasses, metaassociations, metaattributes, etc. in the text, the exact names as they appear in the model are always used.
- No visibilities are presented in the diagrams, as all elements are public.
- If a mandatory section does not apply for a metaclass, the text 'No additional XXX' is used, where 'XXX' is the name of the heading. If an optional section is not applicable, it is not included.

For textual notations a variant of the Backus-Naur Form (BNF) is often used to specify the legal formats. The conventions of this BNF are:

© ISO/IEC 2012 - All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 19505-1:2012

- All non-terminals are in italics and enclosed between angle brackets (e.g. ,*<non-terminal>*).
- All terminals (keywords, strings, etc.), are enclosed between single quotes (e.g., *'or'*).
- Non-terminal production rule definitions are signified with the '::=' operator.
- Repetition of an item is signified by an asterisk placed after that item: '*'
- Alternative choices in a production are separated by the '|' symbol; e.g., *<alternative-A>* | *<alternative-B>*.
- Items that are optional are enclosed in square brackets (e.g., *[<item-x>]*).
- Where items need to be grouped they are enclosed in simple parenthesis. For example, *(<item-1>* | *<item-2>*) * signifies a sequence of one or more items, each of which is *<item-1>* or *<item-2>*.

© ISO/IEC 2012 – All rights reserved
Documents
Document Sharing Hub
Full document details: BS ISO/IEC 19505-1:2012

Subpart II - Infrastructure Library

© ISO/IEC 2012 - All rights reserved

Documents
Document Sharing Hub

Full document details: BS ISO/IEC 19505-1:2012

© ISO/IEC 2012 – All rights reserved
Document Sharing Hub
Full document details: BS ISO/IEC 19505-1:2012

9 Core::Abstractions

The Abstractions package of InfrastructureLibrary::Core is divided into a number of finer-grained packages to facilitate flexible reuse when creating metamodels.

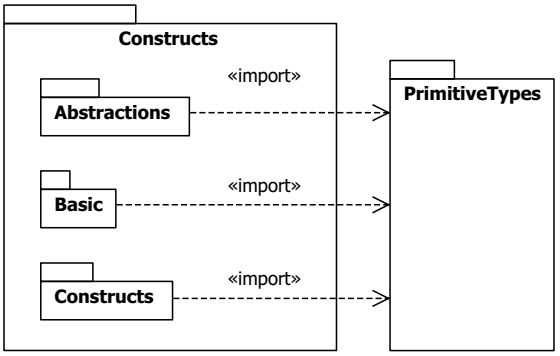

Figure 9.1 - The Core package is owned by the InfrastructureLibrary pack and contains several subpackages

The subpackages of Abstractions are all shown in Figure 9.2.

© ISO/IEC 2012 - All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 19505-1:2012

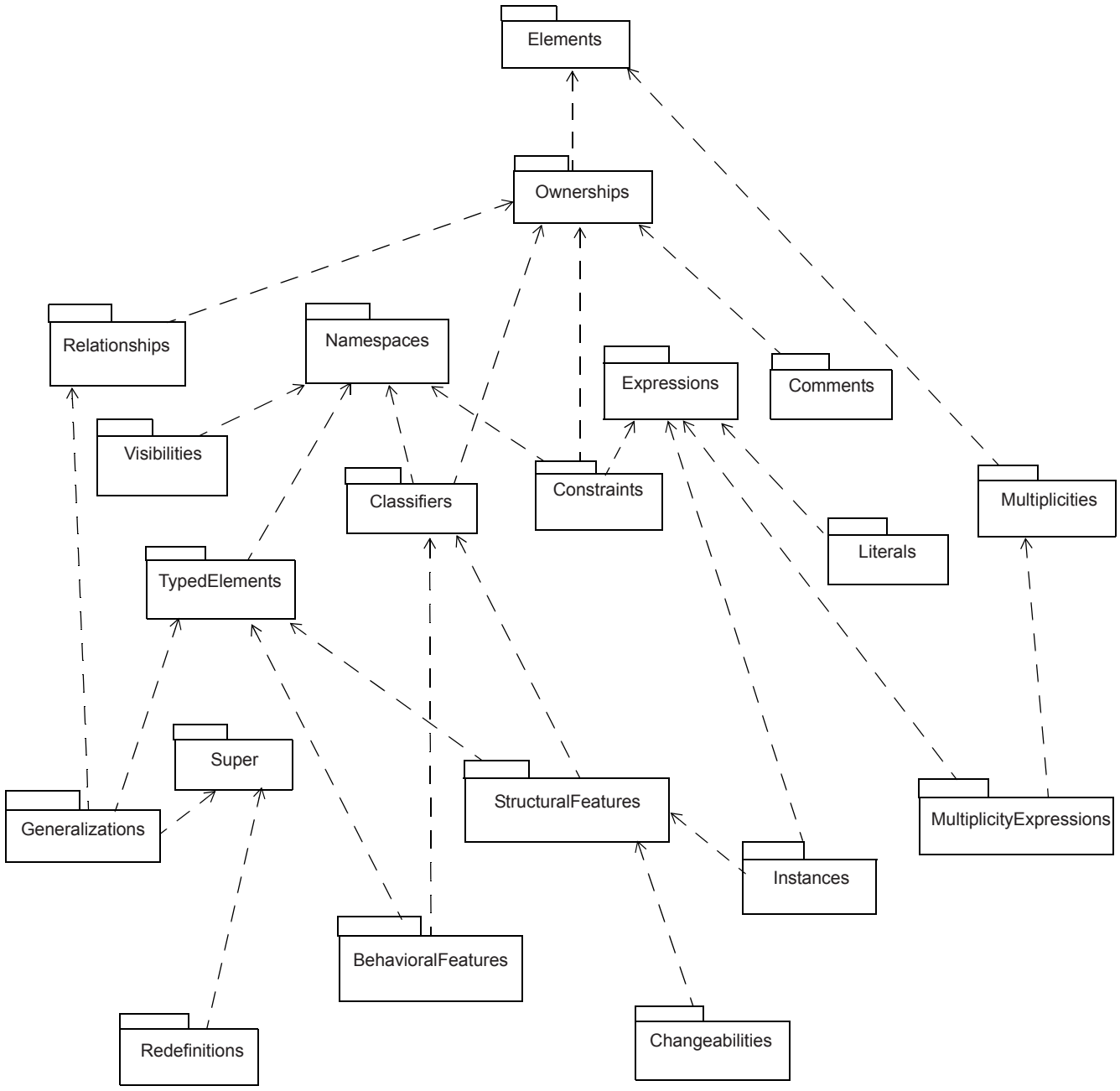

Figure 9.2 - The Abstractions package contains several subpackages, all of which are specified in this clause

The contents of each subpackage of Abstractions is described in a separate sub clause below.

© ISO/IEC 2012 - All rights reserved
Document Sharing Hub
Full document details: BS ISO/IEC 19505-1:2012

9.1 BehavioralFeatures Package

The BehavioralFeatures subpackage of the Abstractions package specifies the basic classes for modeling dynamic features of model elements.

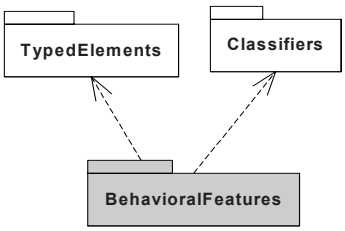

Figure 9.3 - The BehavioralFeatures package

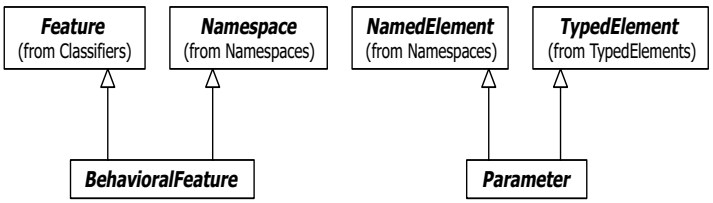

Figure 9.4 - The elements defined in the BehavioralFeatures package

9.1.1 BehavioralFeature

A behavioral feature is a feature of a classifier that specifies an aspect of the behavior of its instances.

Description

A behavioral feature is a feature of a classifier that specifies an aspect of the behavior of its instances. BehavioralFeature is an abstract metaclass specializing *Feature* and *Namespace*. Kinds of behavioral aspects are modeled by subclasses of BehavioralFeature.

Generalizations

- "Feature" on page 38
- "Namespace" on page 75

Attributes

No additional attributes

Constraints

No additional constraints

© ISO/IEC 2012 - All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 19505-1:2012

Additional Operations

[1] The query isDistinguishableFrom() determines whether two BehavioralFeatures may coexist in the same Namespace. It specifies that they have to have different signatures.

```
BehavioralFeature::isDistinguishableFrom(n: NamedElement, ns: Namespace): Boolean;
isDistinguishableFrom =
    if n.oclIsKindOf(BehavioralFeature)
    then
        if ns.getNamesOfMember(self)->intersection(ns.getNamesOfMember(n))->notEmpty()
        then Set{}->including(self)->including(n)->isUnique( bf | bf.parameter->collect(type))
        else true
        endif
    else true
    endif
```

Semantics

The list of parameters describes the order and type of arguments that can be given when the BehavioralFeature is invoked.

Notation

No additional notation

9.2 Parameter

A parameter is a specification of an argument used to pass information into or out of an invocation of a behavioral feature.

Description

Parameter is an abstract metaclass specializing *TypedElement* and *NamedElement*.

Generalizations

- "TypedElement" on page 89
- "NamedElement" on page 73

Attributes

No additional attributes

Associations

No additional associations

Constraints

No additional constraints

© ISO/IEC 2012 – All rights reserved
AVA Documents
Document Sharing Hub
Full document details: BS ISO/IEC 19505-1:2012

Semantics

A parameter specifies arguments that are passed into or out of an invocation of a behavioral element like an operation. A parameter's type restricts what values can be passed.

A parameter may be given a name, which then identifies the parameter uniquely within the parameters of the same behavioral feature. If it is unnamed, it is distinguished only by its position in the ordered list of parameters.

Notation

No general notation. Specific subclasses of BehavioralFeature will define the notation for their parameters.

Style Guidelines

A parameter name typically starts with a lowercase letter.

9.3 Changeabilities Package

The Changeabilities subpackage of the Abstractions package defines when a structural feature may be modified by a client.

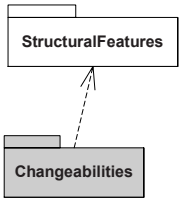

Figure 9.5 - The Changeabilities package

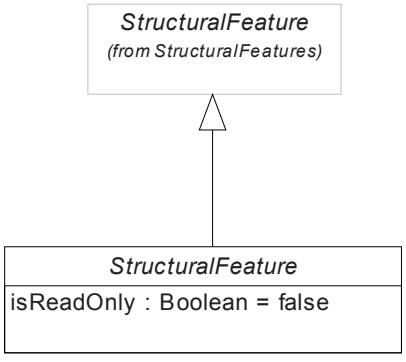

Figure 9.6 - The elements defined in the Changeabilities package

© ISO/IEC 2012 - All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 19505-1:2012

9.3.1 StructuralFeature (as specialized)

Description

StructuralFeature is specialized to add an attribute that determines whether a client may modify its value.

Generalizations

- "StructuralFeature" on page 83

Attributes

- isReadOnly: Boolean — States whether the feature's value may be modified by a client. Default is false.

Associations

No additional associations

Constraints

No additional constraints

Semantics

No additional semantics

Notation

A read only structural feature is shown using {readOnly} as part of the notation for the structural feature. This annotation may be suppressed, in which case it is not possible to determine its value from the diagram.

Presentation Option

It is possible to only allow suppression of this annotation when isReadOnly=false. In this case it is possible to assume this value in all cases where {readOnly} is not shown.

9.4 Classifiers Package

The Classifiers package in the Abstractions package specifies an abstract generalization for the classification of instances according to their features.

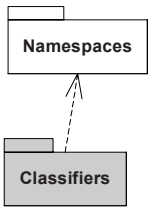

Figure 9.7 - The Classifiers package

© ISO/IEC 2012 - All rights reserved
Documents
Document Sharing Hub
Full document details: BS ISO/IEC 19505-1:2012

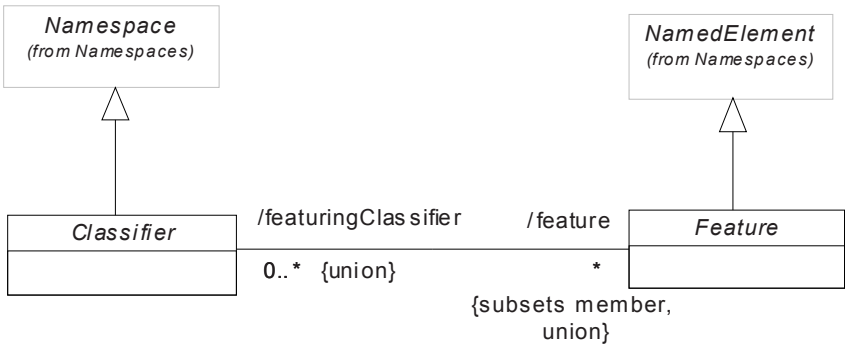

Figure 9.8 - The elements defined in the Classifiers package

9.4.1 Classifier

A classifier is a classification of instances — it describes a set of instances that have features in common.

Description

A classifier is a namespace whose members can include features. Classifier is an abstract metaclass.

Generalizations

- "Namespace" on page 75

Attributes

No additional attributes

Associations

- / feature : Feature [*]
  Specifies each feature defined in the classifier. Subsets *Namespace::member*. This is a derived union.

Additional Operations

[1] The query allFeatures() gives all of the features in the namespace of the classifier. In general, through mechanisms such as inheritance, this will be a larger set than feature.

```
Classifier::allFeatures(): Set(Feature);
allFeatures = member->select(oclIsKindOf(Feature))
```

Constraints

No additional constraints

Semantics

A classifier is a classification of instances according to their features.

© ISO/IEC 2012 - All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 19505-1:2012

Notation

The default notation for a classifier is a solid-outline rectangle containing the classifier's name, and optionally with compartments separated by horizontal lines containing features or other members of the classifier. The specific type of classifier can be shown in guillemets above the name. Some specializations of Classifier have their own distinct notations.

Presentation Options

Any compartment may be suppressed. A separator line is not drawn for a suppressed compartment. If a compartment is suppressed, no inference can be drawn about the presence or absence of elements in it. Compartment names can be used to remove ambiguity, if necessary.

9.4.2 Feature

A feature declares a behavioral or structural characteristic of instances of classifiers.

Description

A feature declares a behavioral or structural characteristic of instances of classifiers. Feature is an abstract metaclass.

Generalizations

- "NamedElement" on page 73

Attributes

No additional attributes

Associations

- / featuringClassifier: Classifier [0..*]
  The Classifiers that have this Feature as a feature. This is a derived union.

Constraints

No additional constraints

Semantics

A Feature represents some characteristic for its featuring classifiers. A Feature can be a feature of multiple classifiers.

Notation

No general notation. Subclasses define their specific notation.

© ISO/IEC 2012 – All rights reserved
AVA Documents
Document Sharing Hub
Full document details: BS ISO/IEC 19505-1:2012