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

**Part 2:
Superstructure**

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

Partie 2: Superstructure



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 Language Units	2
2.2 Compliance Levels	2
2.3 Meaning and Types of Compliance	6
2.4 Compliance Level Contents	8
3. Normative References	9
4. Terms and Definitions	10
5. Notational Conventions	10
5.1 Keywords for Requirement Statements	10
5.2 Annotations on Example Diagrams	10
6. Additional Information	11
6.1 Architectural Alignment and MDA Support	11
6.2 On the Run-Time Semantics of UML	11
6.2.1 The Basic Premises	11
6.2.2 The Semantics Architecture	11
6.2.3 The Basic Causality Model	12
6.2.4 Semantics Descriptions in the Specification	13
6.3 The UML Metamodel	14
6.3.1 Models and What They Model	14
6.3.2 Semantic Levels and Naming	14
6.4 How to Read this Proceed	15
6.4.1 Specification format	15
6.4.2 Diagram format	18
6.4.3 Contents of Subparts	19
Subpart I - Supplement	23
7. Classes	25
7.1 Overview	25

7.2	Abstract Syntax	26
7.3	Class Descriptions	39
7.3.1	Abstraction (from Dependencies)	39
7.3.2	AggregationKind (from Kernel)	39
7.3.3	Association (from Kernel)	40
7.3.4	AssociationClass (from AssociationClasses)	48
7.3.5	BehavioralFeature (from Kernel)	51
7.3.6	BehavioredClassifier (from Interfaces)	52
7.3.7	Class (from Kernel)	52
7.3.8	Classifier (from Kernel, Dependencies, PowerTypes, Interfaces)	55
7.3.9	Comment (from Kernel)	60
7.3.10	Constraint (from Kernel)	61
7.3.11	DataType (from Kernel)	64
7.3.12	Dependency (from Dependencies)	65
7.3.13	DirectedRelationship (from Kernel)	67
7.3.14	Element (from Kernel)	68
7.3.15	ElementImport (from Kernel)	69
7.3.16	Enumeration (from Kernel)	71
7.3.17	EnumerationLiteral (from Kernel)	72
7.3.18	Expression (from Kernel)	73
7.3.19	Feature (from Kernel)	74
7.3.20	Generalization (from Kernel, PowerTypes)	75
7.3.21	GeneralizationSet (from PowerTypes)	79
7.3.22	InstanceSpecification (from Kernel)	87
7.3.23	InstanceValue (from Kernel)	90
7.3.24	Interface (from Interfaces)	91
7.3.25	InterfaceRealization (from Interfaces)	94
7.3.26	LiteralBoolean (from Kernel)	94
7.3.27	LiteralInteger (from Kernel)	95
7.3.28	LiteralNull (from Kernel)	96
7.3.29	LiteralReal	97
7.3.30	LiteralSpecification (from Kernel)	98
7.3.31	LiteralString (from Kernel)	99
7.3.32	LiteralUnlimitedNatural (from Kernel)	99
7.3.33	MultiplicityElement (from Kernel)	100
7.3.34	NamedElement (from Kernel, Dependencies)	104
7.3.35	Namespace (from Kernel)	105
7.3.36	OpaqueExpression (from Kernel)	108
7.3.37	Operation (from Kernel, Interfaces)	109
7.3.38	Package (from Kernel)	113
7.3.39	PackageableElement (from Kernel)	116
7.3.40	PackageImport (from Kernel)	117
7.3.41	PackageMerge (from Kernel)	118
7.3.42	Parameter (from Kernel)	127
7.3.43	ParameterDirectionKind (from Kernel)	129
7.3.44	PrimitiveType (from Kernel)	129
7.3.45	Property (from Kernel, AssociationClasses, Interfaces)	130
7.3.46	Realization (from Dependencies)	136
7.3.47	RedefinableElement (from Kernel)	137
7.3.48	Relationship (from Kernel)	139

7.3.49 Slot (from Kernel)	140
7.3.50 StructuralFeature (from Kernel)	140
7.3.51 Substitution (from Dependencies)	141
7.3.52 Type (from Kernel)	142
7.3.53 TypedElement (from Kernel)	143
7.3.54 Usage (from Dependencies)	144
7.3.55 ValueSpecification (from Kernel)	145
7.3.56 VisibilityKind (from Kernel)	146
7.4 Diagrams	147
8. Components	151
8.1 Overview	151
8.2 Abstract Syntax	151
8.3 Class Descriptions	155
8.3.1 Component (from BasicComponents, PackagingComponents)	155
8.3.2 ComponentRealization (from BasicComponents)	164
8.3.3 ConnectableElement (from BasicComponents)	165
8.3.4 Connector (from BasicComponents)	165
8.3.5 ConnectorEnd (from BasicComponents)	169
8.3.6 ConnectorKind (from BasicComponents)	169
8.4 Diagrams	170
9. Composite Structures	173
9.1 Overview	173
9.2 Abstract Syntax	173
9.3 Class Descriptions	178
9.3.1 Class (from StructuredClasses, InternalStructures)	178
9.3.2 Classifier (from InternalStructures, Collaborations)	179
9.3.3 Collaboration (from Collaborations)	180
9.3.4 CollaborationUse (from Collaborations)	183
9.3.5 ConnectableElement (from InternalStructures)	186
9.3.6 Connector (from InternalStructures)	186
9.3.7 ConnectorEnd (from InternalStructures, Ports)	188
9.3.8 EncapsulatedClassifier (from Ports)	190
9.3.9 Feature (from InternalStructures)	190
9.3.10 InvocationAction (from InvocationActions)	191
9.3.11 Parameter (from Collaborations)	191
9.3.12 Port (from Ports)	192
9.3.13 Property (from InternalStructures)	196
9.3.14 StructuredClassifier (from InternalStructures)	198
9.3.15 Trigger (from InvocationActions)	202
9.3.16 Variable (from StructuredActivities)	203
9.4 Diagrams	203

10. Deployments	205
10.1 Overview	205
10.2 Abstract Syntax	205
10.3 Class Descriptions	209
10.3.1 Artifact (from Artifacts, Nodes)	209
10.3.2 CommunicationPath (from Nodes)	211
10.3.3 DeployedArtifact (from Nodes)	212
10.3.4 Deployment (from ComponentDeployments, Nodes)	213
10.3.5 DeploymentSpecification (from ComponentDeployments)	215
10.3.6 DeploymentTarget (from Nodes)	217
10.3.7 Device (from Nodes)	218
10.3.8 ExecutionEnvironment (from Nodes)	219
10.3.9 InstanceSpecification (from Nodes)	220
10.3.10 Manifestation (from Artifacts)	221
10.3.11 Node (from Nodes)	222
10.3.12 Property (from Nodes)	224
10.4 Diagrams	225
Subpart II - Behavior	229
11. Actions	231
11.1 Overview	231
11.2 Abstract Syntax	233
11.3 Class Descriptions	247
11.3.1 AcceptCallAction (from CompleteActions)	247
11.3.2 AcceptEventAction (from CompleteActions)	248
11.3.3 Action (from BasicActions)	250
11.3.4 ActionInputPin (from StructuredActions)	251
11.3.5 AddStructuralFeatureValueAction (from IntermediateActions)	252
11.3.6 AddVariableValueAction (from StructuredActions)	254
11.3.7 BroadcastSignalAction (from IntermediateActions)	255
11.3.8 CallAction (from BasicActions)	257
11.3.9 CallBehaviorAction (from BasicActions)	257
11.3.10 CallOperationAction (from BasicActions)	259
11.3.11 ClearAssociationAction (from IntermediateActions)	260
11.3.12 ClearStructuralFeatureAction (from IntermediateActions)	261
11.3.13 ClearVariableAction (from StructuredActions)	262
11.3.14 CreateLinkAction (from IntermediateActions)	263
11.3.15 CreateLinkObjectAction (from CompleteActions)	265
11.3.16 CreateObjectAction (from IntermediateActions)	266
11.3.17 DestroyLinkAction (from IntermediateActions)	267
11.3.18 DestroyObjectAction (from IntermediateActions)	268
11.3.19 InputPin (from BasicActions)	269
11.3.20 InvocationAction (from BasicActions)	270
11.3.21 LinkAction (from IntermediateActions)	270

11.3.22 LinkEndCreationData (from IntermediateActions)	272
11.3.23 LinkEndData (from IntermediateActions, CompleteActions)	273
11.3.24 LinkEndDestructionData (from IntermediateActions)	275
11.3.25 MultiplicityElement (from BasicActions)	276
11.3.26 OpaqueAction (from BasicActions)	276
11.3.27 OutputPin (from BasicActions)	277
11.3.28 Pin (from BasicActions)	278
11.3.29 QualifierValue (from CompleteActions)	278
11.3.30 RaiseExceptionAction (from StructuredActions)	279
11.3.31 ReadExtentAction (from CompleteActions)	280
11.3.32 ReadIsClassifiedObjectAction (from CompleteActions)	281
11.3.33 ReadLinkAction (from IntermediateActions)	282
11.3.34 ReadLinkObjectEndAction (from CompleteActions)	284
11.3.35 ReadLinkObjectEndQualifierAction (from CompleteActions)	285
11.3.36 ReadSelfAction (from IntermediateActions)	286
11.3.37 ReadStructuralFeatureAction (from IntermediateActions)	288
11.3.38 ReadVariableAction (from StructuredActions)	289
11.3.39 ReclassifyObjectAction (from CompleteActions)	290
11.3.40 ReduceAction (from CompleteActions)	291
11.3.41 RemoveStructuralFeatureValueAction (from IntermediateActions)	292
11.3.42 RemoveVariableValueAction (from StructuredActions)	294
11.3.43 ReplyAction (from CompleteActions)	295
11.3.44 SendObjectAction (from IntermediateActions)	296
11.3.45 SendSignalAction (from BasicActions)	297
11.3.46 StartClassifierBehaviorAction (from CompleteActions)	298
11.3.47 StartObjectBehaviorAction (from CompleteActions)	299
11.3.48 StructuralFeatureAction (from IntermediateActions)	300
11.3.49 TestIdentityAction (from IntermediateActions)	302
11.3.50 UnmarshalAction (from CompleteActions)	303
11.3.51 ValuePin (from BasicActions)	304
11.3.52 ValueSpecificationAction (from IntermediateActions)	305
11.3.53 VariableAction (from StructuredActions)	306
11.3.54 WriteLinkAction (from IntermediateActions)	306
11.3.55 WriteStructuralFeatureAction (from IntermediateActions)	307
11.3.56 WriteVariableAction (from StructuredActions)	308
11.4 Diagrams	309
12. Activities	311
12.1 Overview	311
12.2 Abstract Syntax	313
12.3 Class Descriptions	325
12.3.1 AcceptEventAction (as specialized)	325
12.3.2 Action (from CompleteActivities, FundamentalActivities, StructuredActivities, CompleteStructuredActivities)	327
12.3.3 ActionInputPin (as specialized)	331
12.3.4 Activity (from BasicActivities, CompleteActivities, FundamentalActivities, StructuredActivities)	332

12.3.5 ActivityEdge (from BasicActivities, CompleteActivities, CompleteStructuredActivities, IntermediateActivities)	342
12.3.6 ActivityFinalNode (from BasicActivities, IntermediateActivities)	347
12.3.7 ActivityGroup (from BasicActivities, FundamentalActivities, IntermediateActivities, StructuredActivities, CompleteActivities, CompleteStructuredActivities)	350
12.3.8 ActivityNode (from BasicActivities, CompleteActivities, FundamentalActivities, IntermediateActivities, CompleteStructuredActivities)	351
12.3.9 ActivityParameterNode (from BasicActivities)	354
12.3.10 ActivityPartition (from IntermediateActivities)	358
12.3.11 AddVariableValueAction (as specialized)	363
12.3.12 Behavior (from CompleteActivities)	364
12.3.13 BehavioralFeature (from CompleteActivities)	365
12.3.14 CallBehaviorAction (as specialized)	366
12.3.15 CallOperationAction (as specialized)	368
12.3.16 CentralBufferNode (from IntermediateActivities)	369
12.3.17 Clause (from CompleteStructuredActivities, StructuredActivities)	370
12.3.18 ConditionalNode (from CompleteStructuredActivities, StructuredActivities)	371
12.3.19 ControlFlow (from BasicActivities)	374
12.3.20 ControlNode (from BasicActivities)	375
12.3.21 DataStoreNode (from CompleteActivities)	377
12.3.22 DecisionNode (from IntermediateActivities)	378
12.3.23 ExceptionHandler (from ExtraStructuredActivities)	381
12.3.24 ExecutableNode (from ExtraStructuredActivities, StructuredActivities)	384
12.3.25 ExpansionKind (from ExtraStructuredActivities)	385
12.3.26 ExpansionNode (from ExtraStructuredActivities)	385
12.3.27 ExpansionRegion (from ExtraStructuredActivities)	386
12.3.28 FinalNode (from IntermediateActivities)	392
12.3.29 FlowFinalNode (from IntermediateActivities)	394
12.3.30 ForkNode (from IntermediateActivities)	395
12.3.31 InitialNode (from BasicActivities)	397
12.3.32 InputPin (from CompleteStructuredActivities)	398
12.3.33 InterruptibleActivityRegion (from CompleteActivities)	399
12.3.34 JoinNode (from CompleteActivities, IntermediateActivities)	401
12.3.35 LoopNode (from CompleteStructuredActivities, StructuredActivities)	404
12.3.36 MergeNode (from IntermediateActivities)	406
12.3.37 ObjectFlow (from BasicActivities, CompleteActivities)	408
12.3.38 ObjectNode (from BasicActivities, CompleteActivities)	413
12.3.39 ObjectNodeOrderingKind (from CompleteActivities)	416
12.3.40 OutputPin (from CompleteStructuredActivities, StructuredActivities)	417
12.3.41 Parameter (from CompleteActivities)	417
12.3.42 ParameterEffectKind (from CompleteActivities)	419
12.3.43 ParameterSet (from CompleteActivities)	419
12.3.44 Pin (from BasicActivities, CompleteActivities)	421
12.3.45 SendObjectAction (as specialized)	428
12.3.46 SendSignalAction (as specialized)	429
12.3.47 SequenceNode (from StructuredActivities)	430
12.3.48 StructuredActivityNode (from CompleteStructuredActivities, StructuredActivities)	431
12.3.49 UnmarshallAction (as specialized)	434
12.3.50 ValuePin (as specialized)	435
12.3.51 ValueSpecificationAction (as specialized)	435
12.3.52 Variable (from StructuredActivities)	436

12.4 Diagrams	438
13. Common Behaviors	443
13.1 Overview	443
13.2 Abstract Syntax	447
13.3 Class Descriptions	452
13.3.1 AnyReceiveEvent (from Communications)	452
13.3.2 Behavior (from BasicBehaviors)	453
13.3.3 BehavioralFeature (from BasicBehaviors, Communications)	456
13.3.4 BehavoredClassifier (from BasicBehaviors, Communications)	457
13.3.5 CallConcurrencyKind (from Communications)	458
13.3.6 CallEvent (from Communications)	459
13.3.7 ChangeEvent (from Communications)	460
13.3.8 Class (from Communications)	461
13.3.9 Duration (from SimpleTime)	462
13.3.10 DurationConstraint (from SimpleTime)	462
13.3.11 DurationInterval (from SimpleTime)	464
13.3.12 DurationObservation (from SimpleTime)	465
13.3.13 Event (from Communications)	465
13.3.14 FunctionBehavior (from BasicBehaviors)	466
13.3.15 Interface (from Communications)	467
13.3.16 Interval (from SimpleTime)	467
13.3.17 IntervalConstraint (from SimpleTime)	468
13.3.18 MessageEvent (from Communications)	469
13.3.19 Observation (from SimpleTime)	469
13.3.20 OpaqueBehavior (from BasicBehaviors)	470
13.3.21 OpaqueExpression (from BasicBehaviors)	471
13.3.22 Operation (from Communications)	471
13.3.23 Reception (from Communications)	472
13.3.24 Signal (from Communications)	473
13.3.25 SignalEvent (from Communications)	474
13.3.26 TimeConstraint (from SimpleTime)	475
13.3.27 TimeEvent (from SimpleTime)	476
13.3.28 TimeExpression (from SimpleTime)	477
13.3.29 TimeInterval (from SimpleTime)	478
13.3.30 TimeObservation (from SimpleTime)	479
13.3.31 Trigger (from Communications)	479
14. Interactions	481
14.1 Overview	481
14.2 Abstract Syntax	482
14.3 Class Descriptions	488
14.3.1 ActionExecutionSpecification (from BasicInteractions)	488
14.3.2 BehaviorExecutionSpecification (from BasicInteractions)	489
14.3.3 CombinedFragment (from Fragments)	490
14.3.4 ConsiderIgnoreFragment (from Fragments)	495

14.3.5 Continuation (from Fragments)	496
14.3.6 DestructionOccurrenceSpecification(from BasicInteractions)	499
14.3.7 ExecutionOccurrenceSpecification (from BasicInteractions)	499
14.3.8 ExecutionSpecification (from BasicInteractions)	500
14.3.9 Gate (from Fragments)	501
14.3.10 GeneralOrdering (from BasicInteractions)	502
14.3.11 Interaction (from BasicInteraction, Fragments)	503
14.3.12 InteractionConstraint (from Fragments)	506
14.3.13 InteractionFragment (from BasicInteractions, Fragments)	507
14.3.14 InteractionOperand (from Fragments)	507
14.3.15 InteractionOperatorKind (from Fragments)	508
14.3.16 InteractionUse (from Fragments)	509
14.3.17 Lifeline (from BasicInteractions, Fragments)	512
14.3.18 Message (from BasicInteractions)	513
14.3.19 MessageEnd (from BasicInteractions)	516
14.3.20 MessageKind (from BasicInteractions)	516
14.3.21 MessageOccurrenceSpecification (from BasicInteractions)	517
14.3.22 MessageSort (from BasicInteractions)	518
14.3.23 OccurrenceSpecification (from BasicInteractions)	518
14.3.24 PartDecomposition (from Fragments)	519
14.3.25 StateInvariant (from BasicInteractions)	522
14.4 Diagrams	523
15. State Machines	543
15.1 Overview	543
15.2 Abstract Syntax	543
15.3 Class Descriptions	546
15.3.1 ConnectionPointReference (from BehaviorStateMachines)	546
15.3.2 FinalState (from BehaviorStateMachines)	549
15.3.3 Interface (from ProtocolStateMachines)	550
15.3.4 Port (from ProtocolStateMachines)	551
15.3.5 ProtocolConformance (from ProtocolStateMachines)	551
15.3.6 ProtocolStateMachine (from ProtocolStateMachines)	552
15.3.7 ProtocolTransition (from ProtocolStateMachines)	554
15.3.8 Pseudostate (from BehaviorStateMachines)	557
15.3.9 PseudostateKind (from BehaviorStateMachines)	564
15.3.10 Region (from BehaviorStateMachines)	565
15.3.11 State (from BehaviorStateMachines, ProtocolStateMachines)	567
15.3.12 StateMachine (from BehaviorStateMachines)	581
15.3.13 TimeEvent (from BehaviorStateMachines)	588
15.3.14 Transition (from BehaviorStateMachines)	589
15.3.15 TransitionKind (from BehaviorStateMachines)	597
15.3.16 Vertex (from BehaviorStateMachines)	600
15.4 Diagrams	600
16. Use Cases	605

16.1 Overview	605
16.2 Abstract Syntax	605
16.3 Class Descriptions	606
16.3.1 Actor (from UseCases)	606
16.3.2 Classifier (from UseCases)	608
16.3.3 Extend (from UseCases)	609
16.3.4 ExtensionPoint (from UseCases)	611
16.3.5 Include (from UseCases)	612
16.3.6 UseCase (from UseCases)	614
16.4 Diagrams	619
Subpart III - Supplement	625
17. Auxiliary Constructs	627
17.1 Overview	627
17.2 InformationFlows	627
17.2.1 InformationFlow (from InformationFlows)	628
17.2.2 InformationItem (from InformationFlows)	630
17.3 Models	633
17.3.1 Model (from Models)	633
17.4 Templates	635
17.4.1 ParameterableElement (from Templates)	637
17.4.2 TemplateableElement (from Templates)	639
17.4.3 TemplateBinding (from Templates)	641
17.4.4 TemplateParameter (from Templates)	642
17.4.5 TemplateParameterSubstitution (from Templates)	644
17.4.6 TemplateSignature (from Templates)	644
17.4.7 Classifier (from Templates)	646
17.4.8 ClassifierTemplateParameter (from Templates)	651
17.4.9 RedefinableTemplateSignature (from Templates)	652
17.4.10 Package (from Templates)	653
17.4.11 PackageableElement (from Templates)	655
17.4.12 NamedElement (from Templates)	656
17.4.13 StringExpression (from Templates)	658
17.4.14 Operation (from Templates)	659
17.4.15 Operation (from Templates)	661
17.4.16 OperationTemplateParameter (from Templates)	661
17.4.17 ConnectableElement (from Templates)	662
17.4.18 ConnectableElementTemplateParameter (from Templates)	663
17.4.19 Property (from Templates)	664
17.4.20 ValueSpecification (from Templates)	665
18. Profiles	667
18.1 Overview	667

18.1.1 Positioning profiles versus metamodels, MOF and UML	667
18.1.2 Profiles History and design requirements	667
18.2 Abstract Syntax	669
18.3 Class Descriptions	670
18.3.1 Class (from Profiles)	670
18.3.2 Extension (from Profiles)	671
18.3.3 ExtensionEnd (from Profiles)	674
18.3.4 Image (from Profiles)	675
18.3.5 Package (from Profiles)	676
18.3.6 PackageableElement (from Profiles)	677
18.3.7 Profile (from Profiles)	678
18.3.8 ProfileApplication (from Profiles)	685
18.3.9 Stereotype (from Profiles)	687
18.4 Diagrams	694
Subpart IV - Annexes	697
Annex A: Diagrams	699
Annex B: Keywords	705
Annex C: Standard Stereotypes	711
Annex D: Component Profile Examples	719
Annex E: Tabular Notations	723
Annex F: Classifiers Taxonomy	727
Annex G: XMI Serialization and Schema	729
Annex H: UML Compliance Level XMI Documents	731
INDEX	733

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 2.

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), Superstructure

1 Scope

1.1 General

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 four compliance levels defined for the whole of UML:

- *Level 0 (L0)*. This compliance level is formally defined in the UML Infrastructure. It 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.
- *Level 1 (L1)*. This level adds new language units and extends the capabilities provided by Level 0. Specifically, it adds language units for use cases, interactions, structures, actions, and activities.
- *Level 2 (L2)*. This level extends the language units already provided in Level 1 and adds language units for deployment, state machine modeling, and profiles.
- *Level 3 (L3)*. This level represents the complete UML. It extends the language units provided by Level 2 and adds new language units for modeling information flows, templates, and model packaging.

The contents of language units are defined by corresponding top-tier packages of the UML metamodel, while the contents of their various increments are defined by second-tier packages within language unit packages. Therefore, the contents of a compliance level are defined by the set of metamodel packages that belong to that level.

As noted, compliance levels build on supporting compliance levels. The principal mechanism used in this International Standard for achieving this is *package merge* (see “PackageMerge (from Kernel)” on page 118). 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” on page 6.

For this reason, all compliance levels are ultimately merged into a single core “UML” model package that defines the common namespace shared by all the compliance levels. Level 0 is defined by the top-level metamodel shown in Figure 2.1. In this model, “L0” is originally an empty package that simply merges in the contents of the *Basic* package from the *UML Infrastructure*. This package is then merged into the UML model. Package L0 contains elementary concepts such as Class, Package, DataType, Operation, etc. merged in from *Basic* (see the *Unified Modeling Language: Infrastructure specification* for the complete list of contents of this package).

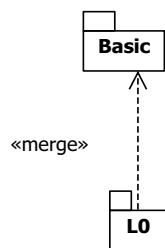


Figure 2.1 - Level 0 package diagram

At the next level (Level 1) the packages merged into Level 0 and their contents are extended with additional packages as shown in Figure 2.2 on page 4. Note that each of the four packages shown in the figure merges in additional packages that are not shown in the diagram. They are defined in the corresponding package diagrams in this Part of ISO/IEC 19505. Consequently, the set of language units that results from this model is more than is indicated by the top-level model in the diagram. The specific packages included at this level are listed in Table 2.3 on page 8.

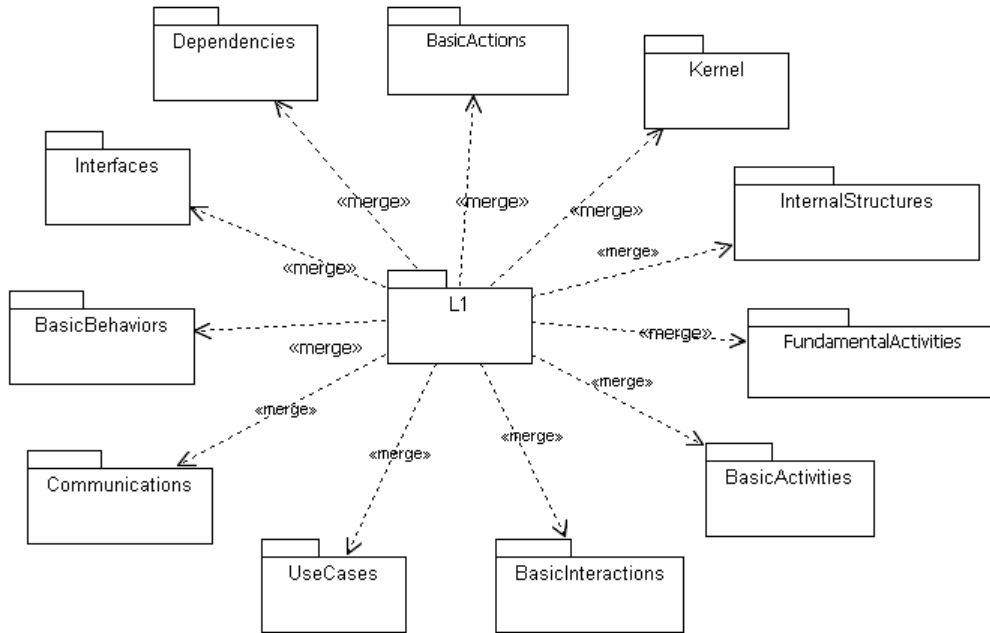


Figure 2.2 - Level 1 top-level package merges

Level 2 adds further language units and extensions to those provided by the Level 1. The actual language units and packages included at this level of compliance are listed in Table 2.4 on page 8.

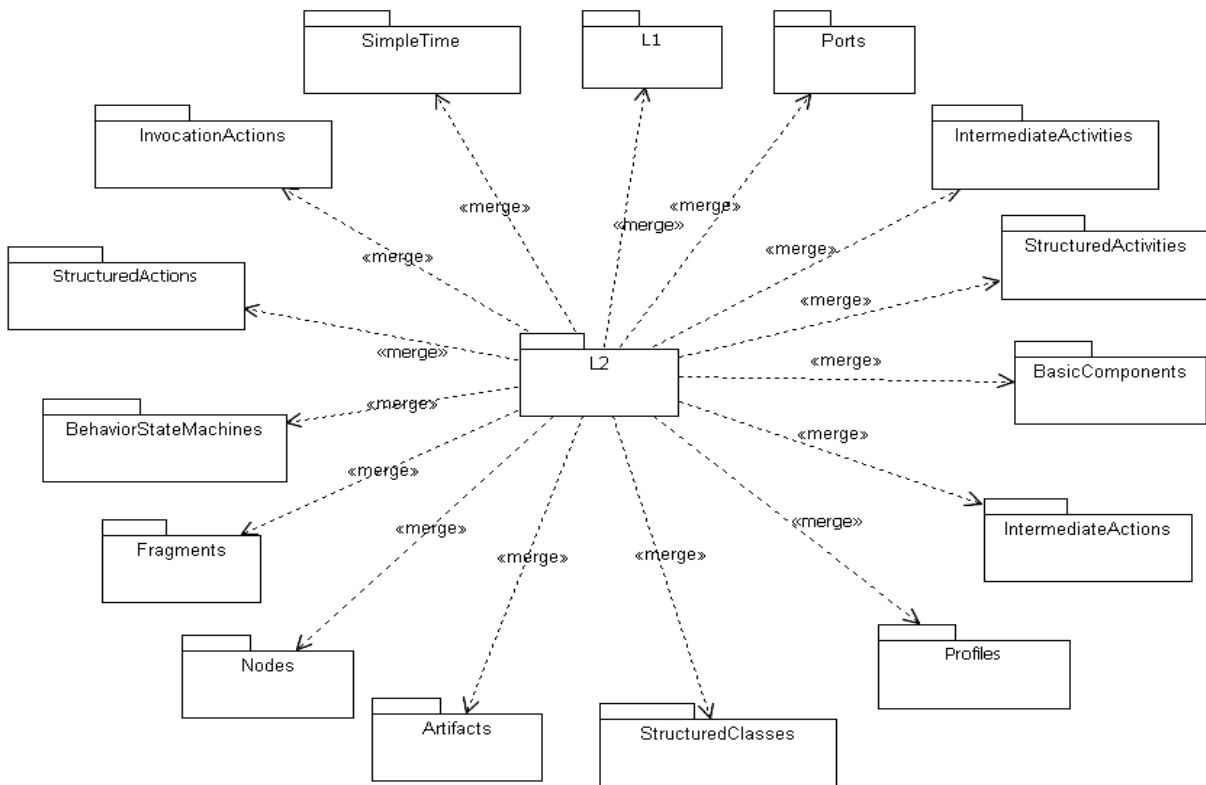


Figure 2.3 - Level 2 top-level package merges

Finally, Level3, incorporating the full UML definition, is shown in Figure 2.4 on page 6. Its contents are described in Table 2.5 on page 9.

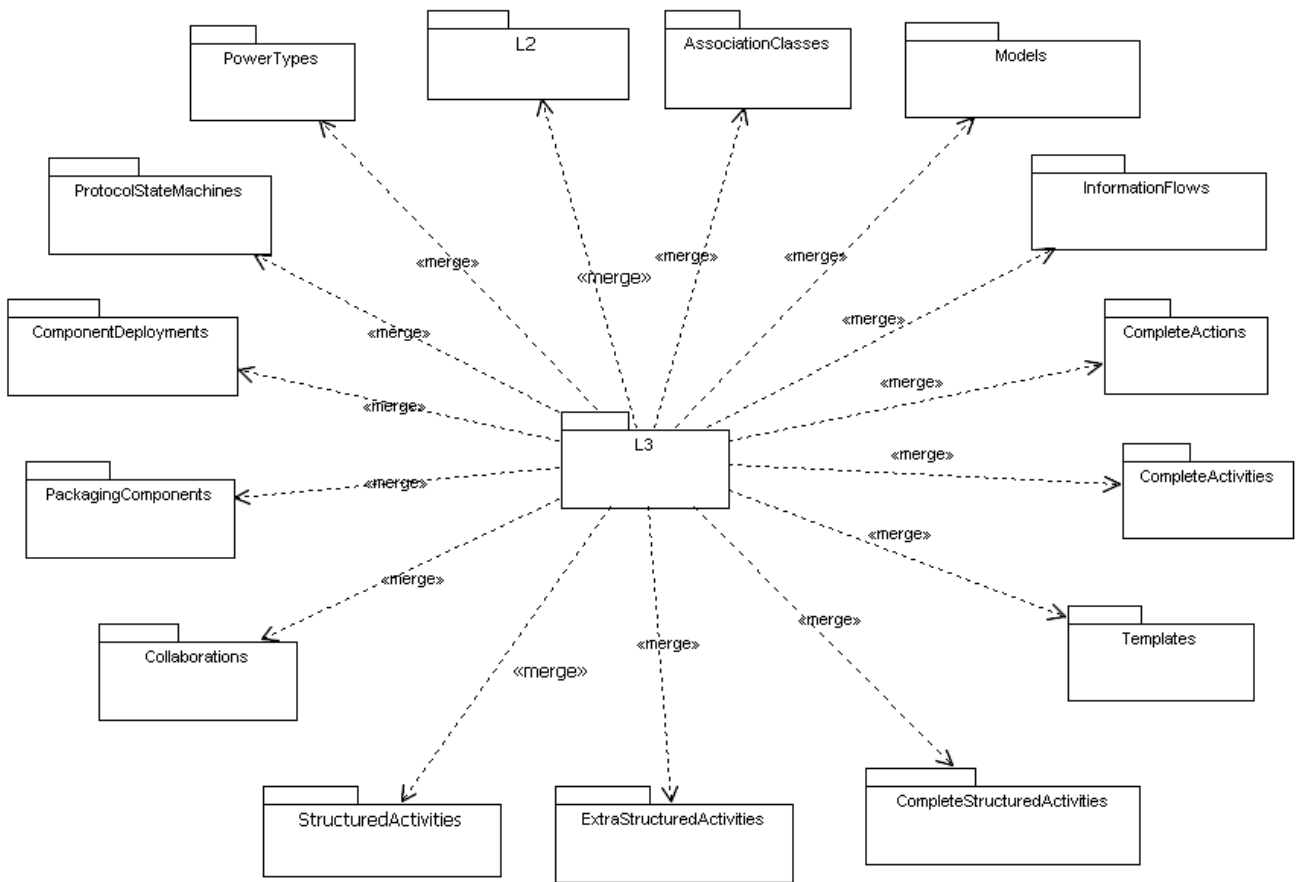


Figure 2.4 - Level 3 top-level package merges

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:

1. *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.

2. *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
Level 0	YES	YES	YES
Level 1	YES	YES	NO
Level 2	YES	NO	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 implementors may make different choices on which ones to support. Finally, it is recognized that some implementors 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, implementors 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	Packages	Abstract Syntax	Concrete Syntax	Semantics	Presentation Options
Deployments	Deployments::Artifacts (L2) Deployments::Nodes (L2)	YES	YES	Note (4)	Note (5)
State Machines	StateMachines::BehaviorStateMachines (L2) StateMachines::ProtocolStateMachines (L3)	Note (1)	YES	Note (2)	Note (3)

Note (1): States and state machines are limited to a single region
Shallow history pseudostates not supported

Note (2): FIFO queueing in event pool

Note (3): Inherited elements indicated using grey-toned lines, etc.

2.5 Compliance Level Contents

Table 2.3 - Metamodel packages added in Level 1

Language Unit	Metamodel Packages
Actions	Actions::BasicActions
Activities	Activities::FundamentalActivities
	Activities::BasicActivities
Classes	Classes::Kernel
	Classes::Dependencies
	Classes::Interfaces
General Behavior	CommonBehaviors::BasicBehaviors
	CommonBehaviors::Communications
Structures	CompositeStructure::InternalStructures
Interactions	Interactions::BasicInteractions
UseCases	UseCases

Table 2.4 - Metamodel packages added in Level 2

Language Unit	Metamodel Packages
Actions	Actions::StructuredActions
	Actions::IntermediateActions
Activities	Activities::IntermediateActivities
	Activities::StructuredActivities
Components	Components::BasicComponents

Table 2.4 - Metamodel packages added in Level 2

Language Unit	Metamodel Packages
Deployments	Deployments::Artifacts
	Deployments::Nodes
General Behavior	CommonBehaviors::SimpleTime
Interactions	Interactions::Fragments
Profiles	AuxilliaryConstructs::Profiles
Structures	CompositeStructures::InvocationActions
	CompositeStructures::Ports
	CompositeStructures::StructuredClasses
State Machines	StateMachines::BehaviorStateMachines

Table 2.5 - Metamodel packages added in Level 3

Language Unit	Metamodel Packages
Action	Actions::CompleteActions
Activities	Activities::CompleteActivities
	Activities::CompleteStructuredActivities
	Activities::ExtraStructuredActivities
Classes	Classes::AssociationClasses
	Classes::PowerTypes
Components	Components::PackagingComponents
Deployments	Deployments::ComponentDeployments
Information Flows	AuxilliaryConstructs::InformationFlows
Models	AuxilliaryConstructs::Models
State Machines	StateMachines::ProtocolStateMachines
Structures	CompositeStructures::Collaborations
	CompositeStructures::StructuredActivities
Templates	AuxilliaryConstructs::Templates

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 19505-2:2012(E)

- ISO/IEC 19505-1, Information technology — OMG Unified Modeling Language (OMG UML) Version 2.4.1 — Part 1: Infrastructure (pas/2011-08-11)
- OMG Specification formal/2011-08-05, UML Infrastructure, 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
- OMG Specification formal/2006-04-04, 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) which are compatible with ISO/IEC 19501 UML version 1.4.1.