Date: April 2006

Diagram Interchange

version 1.0
formal/06-04-04

OBJECT MANAGEMENT GROUP

USE OF SPECIFICATION - TERMS, CONDITIONS & NOTICES

The material in this document details an Object Management Group specification in accordance with the terms, conditions and notices set forth below. This document does not represent a commitment to implement any portion of this specification in any company's products. The information contained in this document is subject to change without notice.

LICENSES

The companies listed above have granted to the Object Management Group, Inc. (OMG) a nonexclusive, royalty-free, paid up, worldwide license to copy and distribute this document and to modify this document and distribute copies of the modified version. Each of the copyright holders listed above has agreed that no person shall be deemed to have infringed the copyright in the included material of any such copyright holder by reason of having used the specification set forth herein or having conformed any computer software to the specification.

Subject to all of the terms and conditions below, the owners of the copyright in this specification hereby grant you a fully-paid up, non-exclusive, nontransferable, perpetual, worldwide license (without the right to sublicense), to use this specification to create and distribute software and special purpose specifications that are based upon this specification, and to use, copy, and distribute this specification as provided under the Copyright Act; provided that: (1) both the copyright notice identified above and this permission notice appear on any copies of this specification; (2) the use of the specifications is for informational purposes and will not be copied or posted on any network computer or broadcast in any media and will not be otherwise resold or transferred for commercial purposes; and (3) no modifications are made to this specification. This limited permission automatically terminates without notice if you breach any of these terms or conditions. Upon termination, you will destroy immediately any copies of the specifications in your possession or control.

PATENTS

The attention of adopters is directed to the possibility that compliance with or adoption of OMG specifications may require use of an invention covered by patent rights. OMG shall not be responsible for identifying patents for which a license may be required by any OMG specification, or for conducting legal inquiries into the legal validity or scope of those patents that are brought to its attention. OMG specifications are prospective and advisory only. Prospective users are responsible for protecting themselves against liability for infringement of patents.

GENERAL USE RESTRICTIONS

Any unauthorized use of this specification may violate copyright laws, trademark laws, and communications regulations and statutes. This document contains information which is protected by copyright. All Rights Reserved. No part of this work covered by copyright herein may be reproduced or used in any form or by any means--graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems--without permission of the copyright owner.

DISCLAIMER OF WARRANTY

WHILE THIS PUBLICATION IS BELIEVED TO BE ACCURATE, IT IS PROVIDED "AS IS" AND MAY CONTAIN ERRORS OR MISPRINTS. THE OBJECT MANAGEMENT GROUP AND THE COMPANIES LISTED ABOVE MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS PUBLICATION, INCLUDING BUT NOT LIMITED TO ANY WARRANTY OF TITLE OR OWNERSHIP, IMPLIED WARRANTY OF MERCHANTABILITY OR WARRANTY OF FITNESS FOR A PARTICULAR PURPOSE OR USE.
IN NO EVENT SHALL THE OBJECT MANAGEMENT GROUP OR ANY OF THE COMPANIES LISTED ABOVE BE LIABLE FOR ERRORS CONTAINED HEREIN OR FOR DIRECT, INDIRECT, INCIDENTAL, SPECIAL, CONSEQUENTIAL, RELIANCE OR COVER DAMAGES, INCLUDING LOSS OF PROFITS, REVENUE, DATA OR USE, INCURRED BY ANY USER OR ANY THIRD PARTY IN CONNECTION WITH THE FURNISHING, PERFORMANCE, OR USE OF THIS MATERIAL, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

The entire risk as to the quality and performance of software developed using this specification is borne by you. This disclaimer of warranty constitutes an essential part of the license granted to you to use this specification.

RESTRICTED RIGHTS LEGEND

Use, duplication or disclosure by the U.S. Government is subject to the restrictions set forth in subparagraph (c) (1) (ii) of The Rights in Technical Data and Computer Software Clause at DFARS 252.227-7013 or in subparagraph (c)(1) and (2) of the Commercial Computer Software - Restricted Rights clauses at 48 C.F.R. 52.227-19 or as specified in 48 C.F.R. 227-7202-2 of the DoD F.A.R. Supplement and its successors, or as specified in 48 C.F.R. 12.212 of the Federal Acquisition Regulations and its successors, as applicable. The specification copyright owners are as indicated above and may be contacted through the Object Management Group, 250 First Avenue, Needham, MA 02494, U.S.A.

TRADEMARKS

The OMG Object Management Group Logo®, CORBA®, CORBA Academy®, The Information Brokerage®, XMI® and IIOP® are registered trademarks of the Object Management Group. OMG™, Object Management Group™, CORBA logos™, OMG Interface Definition Language (IDL)™, The Architecture of Choice for a Changing World™, CORBAservices™, CORBAfacilities™, CORBAmed™, CORBAnet™, Integrate 2002™, Middleware That's Everywhere™, UML™, Unified Modeling Language™, The UML Cube logo™, MOF™, CWM™, The CWM Logo™, Model Driven Architecture™, Model Driven Architecture Logos™, MDA™, OMG Model Driven Architecture™, OMG MDA™ and the XMI Logo™ are trademarks of the Object Management Group. All other products or company names mentioned are used for identification purposes only, and may be trademarks of their respective owners.

COMPLIANCE

The copyright holders listed above acknowledge that the Object Management Group (acting itself or through its designees) is and shall at all times be the sole entity that may authorize developers, suppliers and sellers of computer software to use certification marks, trademarks or other special designations to indicate compliance with these materials.

Software developed under the terms of this license may claim compliance or conformance with this specification if and only if the software compliance is of a nature fully matching the applicable compliance points as stated in the specification. Software developed only partially matching the applicable compliance points may claim only that the software was based on this specification, but may not claim compliance or conformance with this specification. In the event that testing suites are implemented or approved by Object Management Group, Inc., software developed using this specification may claim compliance or conformance with the specification only if the software satisfactorily completes the testing suites.

OMG's Issue Reporting Procedure

All OMG specifications are subject to continuous review and improvement. As part of this process we encourage readers to report any ambiguities, inconsistencies, or inaccuracies they may find by completing the Issue Reporting Form listed on the main web page *http://www.omg.org*, under Documents, Report a Bug/Issue (http://www.omg.org/technology/agreement.htm).

Table of Contents

Preface

About the Object Management Group

OMG

Founded in 1989, the Object Management Group, Inc. (OMG) is an open membership, not-for-profit computer industry standards consortium that produces and maintains computer industry specifications for interoperable, portable and reusable enterprise applications in distributed, heterogeneous environments. Membership includes Information Technology vendors, end users, government agencies and academia.

OMG member companies write, adopt, and maintain its specifications following a mature, open process. OMG's specifications implement the Model Driven Architecture® (MDA®), maximizing ROI through a full-lifecycle approach to enterprise integration that covers multiple operating systems, programming languages, middleware and networking infrastructures, and software development environments. OMG's specifications include: UML® (Unified Modeling Language™); CORBA® (Common Object Request Broker Architecture); CWM™ (Common Warehouse Metamodel); and industry-specific standards for dozens of vertical markets.

More information on the OMG is available at *http://www.omg.org*

OMG Specifications

As noted, OMG specifications address middleware, modeling and vertical domain frameworks. A catalog of all OMG Specifications Catalog is available from the OMG website at:

http://www.omg.org/technology/documents/spec_catalog.htm

Specifications within the Catalog are organized by the following categories:

OMG Modeling Specifications

- UML
- MOF
- XMI
- CWM
- Profile specifications.

OMG Middleware Specifications

- CORBA/IIOP
- IDL/Language Mappings
- Specialized CORBA specifications
- CORBA Component Model (CCM).

Platform Specific Model and Interface Specifications

- CORBAservices
- CORBAfacilities
- OMG Domain specifications
- OMG Embedded Intelligence specifications
- OMG Security specifications.

All of OMG's formal specifications may be downloaded without charge from our website. (Products implementing OMG specifications are available from individual suppliers.) Copies of specifications, available in PostScript and PDF format, may be obtained from the Specifications Catalog cited above or by contacting the Object Management Group, Inc. at:

OMG Headquarters
140 Kendrick Street
Building A, Suite 300
Needham, MA 02494
USA
Tel: +1-781-444-0404
Fax: +1-781-444-0320
Email: *pubs@omg.org*

Certain OMG specifications are also available as ISO standards. Please consult *http://www.iso.org*

Typographical Conventions

The type styles shown below are used in this document to distinguish programming statements from ordinary English. However, these conventions are not used in tables or section headings where no distinction is necessary.

Times/Times New Roman - 10 pt.: Standard body text

Helvetica/Arial - 10 pt. Bold: OMG Interface Definition Language (OMG IDL) and syntax elements.

Courier - 10 pt. Bold: Programming language elements.

Helvetica/Arial - 10 pt: Exceptions

Note – Terms that appear in *italics* are defined in the glossary. Italic text also represents the name of a document, specification, or other publication.

Issues

The reader is encouraged to report any technical or editing issues/problems with this specification to *http://www.omg.org/technology/agreement.htm.*

1 Scope

The goal of this specification is to enable a smooth and seamless exchange of documents compliant to the UML standard (in the following referred to as UML models) between different software tools. While this certainly includes tools for developing UML models, it also includes tools such as whiteboard tools, code generators, word processing tools, and desktop publishing tools. Also, special attention is given to the Internet as a medium for exchanging and presenting UML models.

Although a mechanism for exchanging UML models had been specified for UML 1.x using XMI, namely the XML Metadata Interchange (in the following referred to as XMI[UML]) this mechanism did not fully fulfill the goal of a model interchange. Foremost, it did not include the exchange of diagram information. This mechanism was only capable of transporting information on what elements are contained in a UML model but not the information on how these elements are represented and laid out in diagrams. Thus, if a UML model is stored in one UML tool and then loaded in a different UML tool (or even the same tool) using XMI[UML], all diagram information is lost. This limitation is not due to XMI itself but instead to the fact that the UML metamodel does not define a standard way of representing diagram definitions.

The solution extends the UML metamodel by a supplementary package for graph-oriented information while leaving the current UML metamodel fully intact. Moreover, it is compliant with the UML 2.0 metamodel and should also be unaffected by any subsequent changes to the UML metamodel. A MOF-compliant metamodel for UML diagram information is provided as extension to the UML metamodel, allowing the DTD for the XMI to be extended. The resulting XMI can then be used to exchange UML models between various tools without information loss.

To assure the exchange to tools that do not have a notion of model elements but of lines, text, and graphics, a transformation mechanism from XMI to SVG is provided. SVG is an XML-based format for scalable vector graphics that has been adopted as a W3C Recommendation. Well-suited to express any diagram of the UML, it will be a commonly used format for a wide variety of tools (graphical, desktop publishing, etc.) and was created to be fit for the web.

In combination with a tighter definition of XMI[UML] in the Infrastructure and Superstructure for UML 2.0, this specification will make a smooth and seamless exchange mechanism for UML models available.

2 Conformance

There are no optional compliance points for the Diagram Interchange. To be compliant with Diagram Interchange means to be compliant to its abstract syntax, well-formedness rules, semantics, notation, and XMI interchange.

3 Normative References

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

- UML
- MOF
- XMI

4 Terms and Definitions

This specification defines no additional terms.

5 Symbols

This specification defines no additional symbols.

6 Additional information

6.1 Statement of Proof of Concept

The metamodel described has been fully implemented and tested using a set of different diagrams. The concepts presented are regarded as proven. However, this mechanism has thus far not been employed for interchange between different vendors and it might be necessary to tune some of the details to ensure interoperability.

A more detailed evaluation of whether all the aspects of UML 2.0 have been taken into account is needed. In the current version, UML 1.4, XMI 1.2, and MOF 1.4 are used.

The example provided in Annex B and used throughout the document has so far been transformed into its XMI representation by hand rather than automatically. However, this example has been validated against the XSD provided and translated to SVG using the XSLT made available.

The specification will be fully implemented by several parties and the interoperability between different tools will be tested.

6.2 Design Rationale

UML is a modeling language for object-oriented software systems with a strong emphasis on a graphical representation. It is deployed throughout the software development process and there is a wide variety of tools that can be utilized during this process. Tools vary greatly: there is an extensive range geared to design the diagrams, to check the consistency of models, to store them for persistence or for versioning, for generating code, for preparing demonstrations, presentations or documentation, and many more applications. The ability to seamlessly use and combine all of these various tools trouble-free is highly valuable and desirable. Accordingly, a mechanism for representing (and hence exchanging) model information was included in the first UML standard. However, the mechanism laid out in UML 1.x merely supports the definition of elements in a model. While this is essential for tools that check the consistency of a model or generate code, this information is not sufficient for graphically oriented tools. This thus excludes a wide variety of tools that make use of graphical information, including UML tools themselves. In this respect, the model interchange mechanism of UML 1.x falls short and the need to correct this deficiency has been recognized and addressed by the OMG.

The general mechanism applied within the OMG to transport meta-information is XMI. XMI is an application of XML, which lends itself to transporting information that is highly internally referential. Both object-oriented models and the models of such models fall into this category but are only one example. XMI was applied to transport UML models by generating a special XSD through applying the rules of XMI to the concrete UML metamodel. To distinguish between XMI in general and its application to UML, we will refer to the latter as XMI[UML] in this document. This mechanism has proved to be highly useful despite the fact that graphical information was not included.

UML diagrams, just like UML models, can be described using a metamodel. This document suggests a separate metamodel for diagram information that can simply be added as a separate package to the existing metamodel of UML. And just as with the metamodel of UML, XMI can be applied to transport instances of this diagram metamodel. The corresponding XSD (which simply extends the XSD of XMI[UML]) is generated directly from the metamodel and provided in this document. This format will be referred to as XMI[DI] here. The conjunction of both, which makes up the complete model interchange mechanism, will be denoted as XMI[UML+DI], or simply as XMI for brevity.

In the current version, one XSD is generated, describing the XMI[UML] and the XMI[DI] part. Using XLink for referencing other ModelElements, both parts can be placed in different files, allowing different XSDs for each part to be generated.

The metamodel in this document conforms to the MOF metamodel facility. XMI defines how to create a DTD or a schema from a MOF-compliant metamodel and how this is to be applied to XMI. Thus, once a MOF-compliant metamodel has been agreed on, its representation in XMI and the corresponding DTD is taken care of.

The metamodel itself was developed with two key objectives in mind. First, it should flexibly extend the existing UML metamodel (as well as future revisions) without interfering with it and without modifying it. At the same time it should carry as little redundant information as possible and instead reference the UML metamodel to access this information. Second, it should allow any tool to easily render the diagram from the given information. This, of course, includes UML tools but also web browsers, office suites, graphical editors, etc.

The tools addressed are very different in their needs. While UML tools usually enable the rendering of model elements to lines and text themselves, text editing tools have no knowledge at all about model elements and require a diagram to be in a common graphical format. And graphic tools typically require a rich vector-oriented format for manipulating and scaling.

SVG, the scalable vector graphic format, is a new W3C standard that promises to be supported by a wide variety of these tools. It is based on XML and can easily be read in, processed, and transformed into many other formats. It is equally suited for text tools, office suites, and graphic tools. Moreover, it is suitable for web browsers that will soon support it directly (but in the interim can use a freely available Adobe plugin).

However, SVG is not a well-suited mechanism for UML tools. UML tools do not require solely the lines and text but need information at a higher level for they do not merely display the diagrams graphically they also call for a semantic understanding of the ModelElements that are represented by the graphical primitives.

Yet one of the great advantages of XML is that data expressed in one XML data format can easily be transformed into a different XML data format as long as all required information is present. Therefore this specification suggests a metamodel transported using XMI[DI] and provides a transformation from XMI[UML+DI] to SVG using XSLT. This approach satisfies the needs of the widest range of tools. All other required formats can then be produced from this.

So, what is the abstraction that commonly expresses the additional information to allow interchange for all diagrams in UML? While one alternative would be to introduce special classes for each and every kind of shape that UML diagrams consist of, if UML is to be extended or its scope broadened or if the core mechanisms are to be reusable for other modeling notations, this approach seems too inflexible. The diagram interchange should not restrict the extensibility of UML. If possible, the diagram interchange mechanism should not have a notion of concrete shapes or other elements. The drawing of the concrete forms of the shapes used lie in the responsibility of the UML tools or of an SVG renderer.

It can be observed that most diagrams in UML follow the graph schema as known from graph theory: they consist of nodes (which may be rectangles, ovals, circles, or other shapes) and edges (connecting lines between the nodes with different arrows and shapes at the ends). Nodes can contain compartments and annotations; edges can have annotations attached to them. Some nodes can be nested in others, with edges connecting two nodes (in some cases they may also connect edges).

The graph schema is a very powerful, well understood, abstraction that is employed in many areas of visual modeling. It is the abstraction used by this specification since it proves to be fully sufficient, as will be discussed in this document. Most UML diagrams have a natural and straightforward mapping to a graph. This is also the case for class diagrams, use case diagrams, collaboration diagrams, object diagrams, component diagrams, and deployment diagrams. In the current form of the UML, it is only sequence diagrams that do not have such a natural mapping to nodes and edges; yet a mapping can clearly be found. It is argued that, although the graph based approach may not be the obvious choice for all diagram types, it is well suited for most and fully sufficient to represent any current and future UML diagram types.

6.3 Acknowledgements

The following companies submitted and/or supported parts of this specification:

- Adaptive
- Compuware Corporation
- DaimlerChrysler AG
- Gentleware AG
- Hamburg University
- I-Logix
- IBM Rational Software
- Softeam
- Sun Microsystems
- Telelogic AB
- TogetherSoft Corporation (now part of Borland)
- University College London

7 Architectural Overview

The extension of XMI[UML] to XMI[UML+DI] employs several technologies to create both its metamodel and the instantiated objects of the mentioned model. The technologies involved have been published as standards by the OMG and the W3C and anyone is free to utilize them. The basic technology, serving as foundation for all the others involved in this process, is XML. It consists of fundamental rules (e.g., well-formedness) for how to create documents, describing their content by tagging. This commonly accepted mechanism is supported by many tools all around the world.

7.1 XMI[DI] DTD Creation Overview

The following diagram provides an overview of all the technologies that are involved in the creation process of the metamodel and DTD.

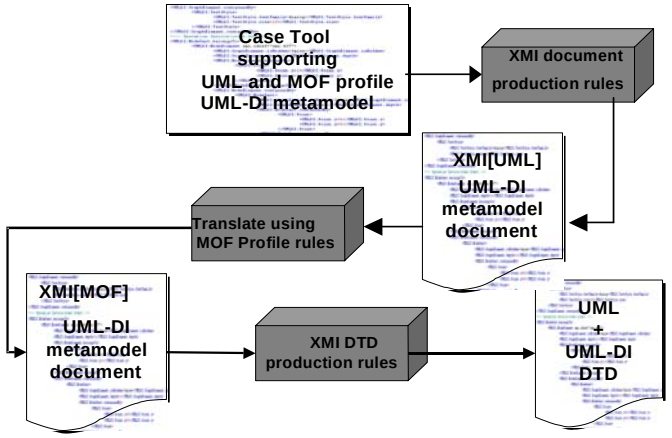


Figure 7.1 - XMI[DI] DTD creation overview

The XMI[DI] metamodel is created with a UML modeling tool. With this tool, a MOF-compliant metamodel is created using the UML profile for MOF, describing the UML M2 extension. For conformance to the XMI specification, this tool needs the capability to create an XMI[UML] document. This is a document following the rules defined in XML and an XMI[UML] DTD belonging to it. This step is represented by the XMI document production rules. As mentioned, it produces an XMI-conformant document containing the content of the metamodel created with the help of the UML tool. In order to generate the new XSD for XMI[UML+DI], the new M2 must be expressed in MOF form by translating the XMI[UML] document to an XMI[MOF] document. The rules for the mapping from the profile to MOF are contained in the UML profile for MOF. Another option would be to generate the XMI[MOF] document directly from the CASE tool. Finally the XMI XSD production rules can be applied to the MOF representation of the M2. Using the rules in the XMI specification, an XML schema can also be generated.

Based on this extended XMI[UML+DI] metamodel, it is now possible to include the graphical information of an XMI[UML] model when exchanging it. Furthermore several representations of a model can be created. One option is to create a representation in SVG.

SVG (Scalable Vector Graphics) is a technology geared to describe vectorized graphics in a clear text (XML-based) format and produce a visualization out of this. It was published as a Recommendation by the W3C. In contrast to plain graphics such as bitmaps, SVG enables, just to mention some of the possibilities, graphics to be scaled, rotated, or methods invoked on single elements of the technology. It is also capable of handling user interaction, which offers many alternatives to work with such SVG-based graphics.

XSLT (eXtensible Stylesheet Language) is another W3C Recommendation that defines how to create stylesheets (themselves XML documents) for the transformation of an XML document into another (usually also XML) document. In this case the stylesheet is used to transform the XMI[UML+DI] XML document containing the UML model and diagramming information into an SVG XML document containing full graphical rendition information that can be displayed by a browser. Note that XSLT engines to perform stylesheet-driven transformations are commonly available including in open source (most notably Xalan from the Apache project).

7.1.1 SVG Graphic Creation Overview

The following picture provides an overview of the technologies involved in the creation of an SVG document from a UML modeling tool.

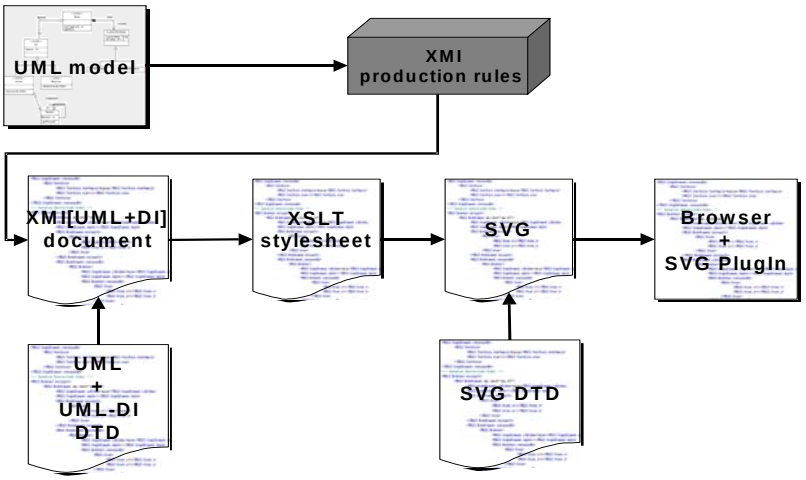


Figure 7.2 - SVG graphic creation overview

As when creating the XMI[DI] extension, the starting point is a UML modeling tool to describe a model. Based on this model, an XMI document is created using the XMI production rules. The result is again an XMI document containing the content and, in contrast to past approaches, the graphical information of the model. The document can be validated against the XMI[UML] XSD containing the XMI[DI] extension in order to check the syntactic correctness of the XMI document. Apart from this, the well-formedness of the document is always checked. The next step is to create an SVG document out of the XMI source document. This is done by using an XSLT stylesheet, which only needs to be produced once and is then applicable to all XMI documents containing both the model and the diagram information. Applying this stylesheet to the XMI[UML+DI] source document generated results in an SVG document. The SVG document can be displayed as a diagram using an SVG viewer to visualize the model by rendering the input document. Different viewers exist, including integrated plug-ins for common internet browsers. Thus, it is possible to view and navigate UML diagrams in a browser with a high level of user interaction, which can be designed to be as intuitive and easy as UML tools can currently be used.

8 Metamodel Extension

This chapter describes the metamodel for diagram information that the diagram interchange mechanism relies on. It is an extension of the UML metamodel and is currently based on UML. The existing mechanism of XMI[UML] for exchanging models includes only the model information but not the graphical information. The diagram interchange extension allows graphical information to be included for diagrams used in UML models.

This extension adds a new package to the current UML metamodel packages. Yet the existing standard is not changed in any way. Also, changes to the UML metamodel due to version updates should not affect this model as long as the high-level notions of Core::Element (as used in UML 1.x) and Elements::Element (as used in UML 2.0 as well as all metamodels based on the Common Core) are maintained. The extension and the UML metamodel are kept largely independent such that solely links from the extension to the UML metamodel are included. Thus, graphical and model information are cleanly separated.

In addition, conflicts with tools supporting the current standard are avoided and full backward compatibility is maintained. Flexibility for future extensions to UML itself is provided.

The package contains elements to reflect the diagram information of any diagram element of the standard UML. Tool-specific extensions can be defined in additional packages. For example, if a tool adds drawing capabilities for additional shapes, then an additional package to describe these can be provided. The purpose of this is to guarantee that tools not supporting these additional extensions are nonetheless able to generate a graphical representation by simply ignoring the information from an extra package. Figure 8.1 depicts the metamodel for representation of diagram information.

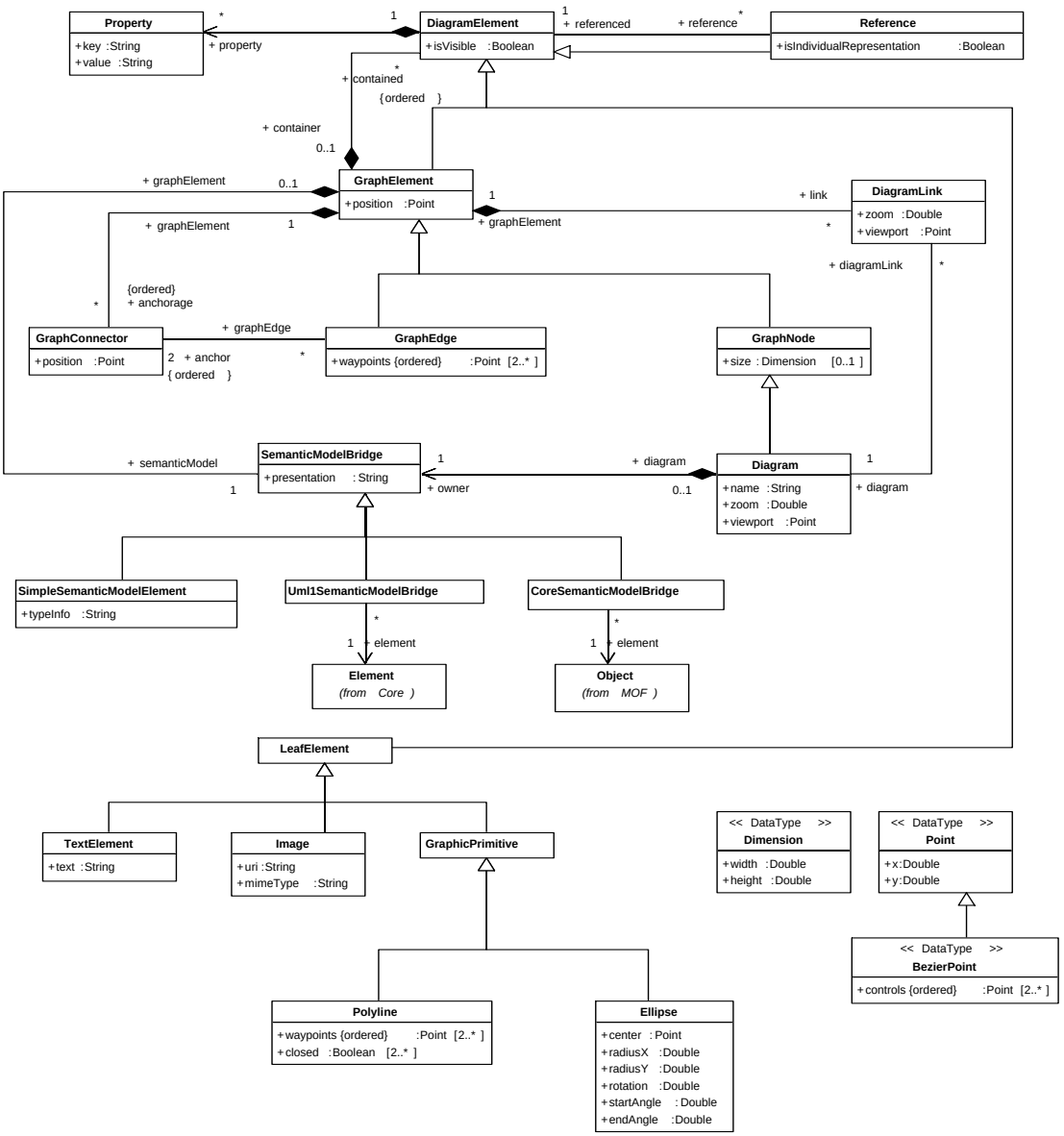


Figure 8.1 - Diagram Interchange Metamodel

8.1 Extended Graph Model

The underlying concept of this metamodel extension is based on the idea of modeling the contents of the UML diagrams as graphs. The core classes are GraphNode and GraphEdge. Every visible model element is represented either by a GraphNode or by a GraphEdge. The base class of the graph elements is GraphElement. Graph elements are linked via a class called GraphConnector. This allows linking of a GraphEdge with a GraphNode or another GraphEdge. The latter case is an extension to the concept of a pure mathematical graph. A GraphConnector does not permit two GraphNodes to be linked.

A GraphElement can own any number of GraphConnectors, called anchorages. They permit any number of GraphEdges to connect to them. A GraphEdge references two GraphConnectors, which are its connection end points. From the perspective of the GraphEdge these are called anchors. The two GraphConnectors are ordered in the same way as the waypoints of the corresponding GraphEdge. The first GraphConnector corresponds to the first waypoint of the GraphEdge and the second GraphConnector corresponds to the last waypoint.

A GraphConnector is not obliged to have the same position as the end points of the GraphEdges that reference it. When being referenced by more than one GraphEdge, it serves as a virtual collection point for GraphEdges, e.g., in the middle of a node or at another specialized point of a node. All linked GraphEdges point straight to the GraphConnector while its waypoints may, for example, be defined as ending at the border of the shape of the connected node (Figure 8.2). A UML tool that reads this information may interpret it as clipping, which might be helpful for further editing.

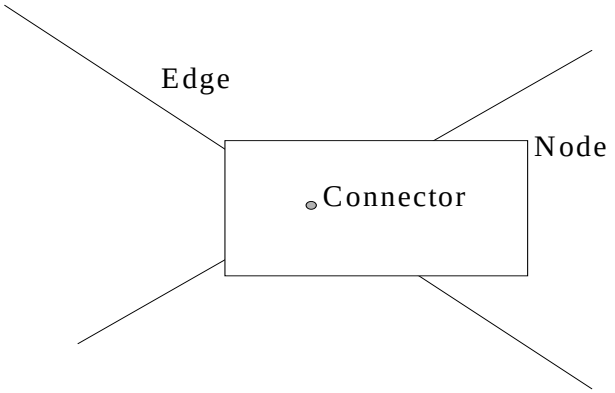


Figure 8.2 - Example of a GraphConnector with radial edges

8.2 Nesting

The association container—contained between GraphElement and DiagramElement—constitutes a further extension to the model of a pure mathematical graph. It allows a hierarchy of nested elements to be built. Each GraphElement is able to contain an unlimited number of DiagramElements, so that it may contain an entire subgraph. The nested hierarchy is especially useful when modeling the representation of complex model elements.

For example, classes are represented by a GraphNode that owns nested GraphNodes as contained DiagramElements to represent the compartments, e.g., operation compartment, attribute compartment. Such a compartment GraphNode, e.g., an attribute compartment, owns further nested GraphNodes as contained DiagramElements that represent attributes. The attribute GraphNode again has nested contained DiagramElements that represent parts of the attribute, e.g., visibility, name, type, or initial value.

Figure 8.3 sets out a simple class as an example of the nesting of GraphElements, which are GraphNodes in this case. A more complex class may contain more nested GraphNodes to be represented completely.

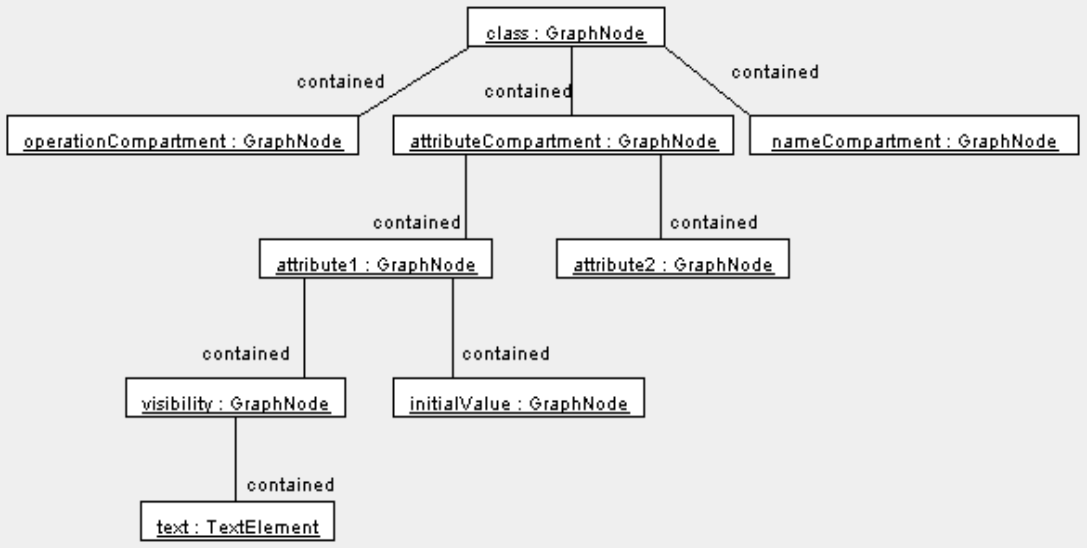


Figure 8.3 - Example of nested GraphElements in the representation of a class

8.3 Leaves

The graphical representation of a GraphElement is largely derived from the semantic model (see below) and not explicitly stored in the DI metamodel to avoid redundancy. For example, the knowledge of drawing a class as a rectangle is not stored in the DI metamodel. Note that its position and its size, if needed, are stored. A drawing tool and the XSL stylesheet must have the semantic knowledge of drawing these elements correctly.

LeafElements can be utilized to represent model elements with special optional representation, e.g., actors. If a GraphNode representing an actor has the presentation of the SemanticModelBridge set to UserDefined (see below), it must have a LeafElement as child. This LeafElement represents the user-defined visualization of the actor.

A LeafElement can be a TextElement to show a simple text, a GraphicPrimitive to show a simple drawing such as an ellipse, or an Image to show an external resource.

Two basic specialized graphic primitives are part of this specification: Polyline and Ellipse. Other graphic primitives can be defined in an extra package and inherit from the class GraphicPrimitive or its subclasses. TextElements are employed to represent parts of attributes, operations, names, and other texts that are part of model elements. For example, the visibility of an attribute can be a text such as “public” or “+”. This text is stored in the attribute text of the TextElement. The container GraphNode of this TextElement has a link to the semantic model in the form of a SimpleSemanticModelElement. This indicates the part of the attribute that is being referenced. The attribute typeInfo of the SimpleSemanticModelElement contains visibility in this example. Naming rules for the attribute typeInfo are described in section ‘additional semantic information’ below. Figure 8.4 illustrates the relevant part of the objects involved. The SimpleSemanticModelElement of the GraphNode name is not shown in this figure.

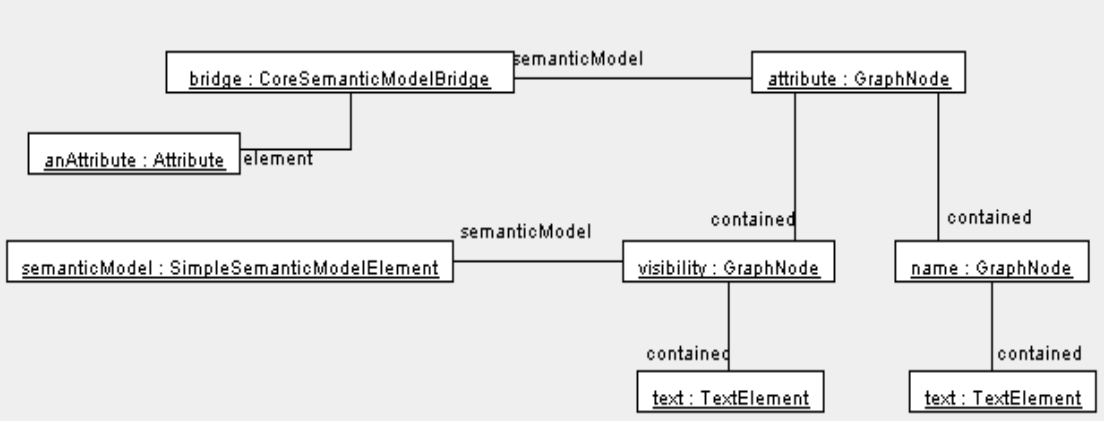


Figure 8.4 - Example of a TextElement

If the visibility is to be represented by an icon, the TextElement is replaced by another LeafElement. If the icon is stored as an image in another file, the LeafElement is an Image referencing that file.

LeafElements that are used many times are only needed to be instantiated once. These LeafElements are directly contained by the Diagram. Every time such a LeafElement is used, a Reference referencing this LeafElement is instantiated instead of a copy of the LeafElement. The isIndividualRepresentation flag is set to true to distinguish this usage of the Reference from the usages described below. This avoids multiple copies of the same visual component.

8.4 Diagram

A special node is the Diagram. It is the topmost GraphElement of any graph or diagram in this terminology and recursively contains all other GraphicElements. A Diagram has a name and a viewport. The viewport is a point that indicates the top left-hand corner of the current visible part of the Diagram. It also has a zoom factor, which allows it to be shown at a different scale. The type of the Diagram is stored in a SimpleSemanticModelElement, e.g., StateDiagram. The Diagram references it through its semanticModel. A graph element can be linked to diagrams via a DiagramLink. Such a link could be employed if a graph element can be represented by other diagrams, e.g., on a more detailed level or if the graph element has a special semantic relation to other diagrams. A DiagramLink has its own zoom factor and viewport, so each diagram can be displayed with a different zoom factor and viewport in each context. An example for the use of a diagram link is a state diagram that displays the behavior of a class or a class diagram, which displays details of a package.

A simple example of a Diagram is given through the object diagram in Figure 8.5. It shows a class diagram with two classes that are linked through an association. The diagram and the classes are represented through GraphNodes, while the association is represented by a GraphEdge. This object diagram also shows the role that GraphConnectors assume in linking GraphNodes and GraphEdges.

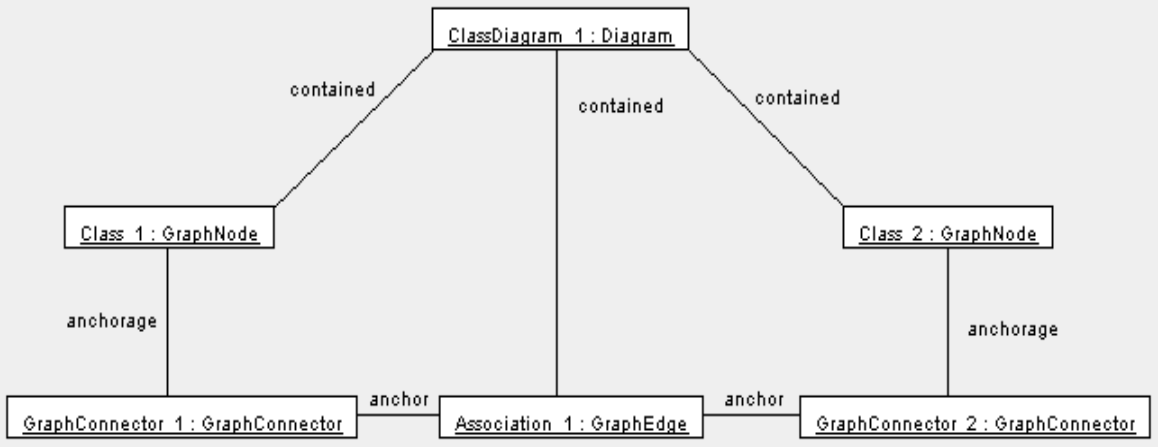


Figure 8.5 - Example of a class diagram (only the UML[DI] part depicted)

8.4.1 The Owner of a Diagram

In general, a UML Element is assigned to a UML Namespace while a DiagramElement is not. A DiagramElement does not need a namespace as long as its SemanticModelBridge (see below) references a UML Element that has one. If this is not the case, a surrounding GraphElement serving as container will have a namespace. However, there is an exception to this rule. The Diagram does not have a namespace and there is no GraphElement containing it because the Diagram itself is the topmost container.

Therefore, the Diagram is in need of a namespace. The namespace of the Diagram is assigned by the reference of a namespace element of the semantic model via the SemanticModelBridge. The Element referenced by this way from the diagram must be a subclass of Namespace or in case of state- and activity diagrams the state machine. This element is called the owner of a diagram.

8.4.2 Showing a cut-out of a Diagram in another Diagram

A Diagram may be used to show an extract of an existing Diagram without the need of creating the used DiagramElements more often than once. For example, a large Diagram may be shown through several smaller Diagrams for a better representation. To achieve this, the large Diagram would own all its containing DiagramElements while the cut-out Diagrams would only contain References to the original DiagramElements. Only those DiagramElements are shown in a cut-out Diagram that are being pointed to by a Reference. References and stand-alone DiagramElements may be used independently in the same diagram.

8.4.3 Visibility of Diagram Elements

The attribute visible specifies whether a DiagramElement is being shown or not. Its value is applied recursively to all contained DiagramElements. If only one DiagramElement in the hierarchy has set its visible attribute to false, all nested elements are also hidden, regardless of their own visibility.

While a hidden element is not shown, it still exists. This means that if it is shown by setting visible to true, the element appears in the same way as before. This provides a comfortable way to fade out a compartment, for example, while keeping its representation. This makes it easy to show a previously hidden DiagramElement again later.

8.5 Properties

The appearance of DiagramElements is specified by properties. Properties can be added to a DiagramElement, influencing its appearance in different ways. The table below defines a standard set of optional properties: it comprises font family and size as well as line style, thickness, and color. Each property overwrites any existing property of the same type of an enclosing GraphElement. If a property is not set, the DiagramElement utilizes the property of its container GraphElement. Non-standard properties could be added but are not part of any standardization. Properties provide an extension mechanism for adding non-standardized elements to the diagram interchange metamodel.

Table 8.1 - Standardized properties

Key	Domain	Example	Description
FontFamily	string	‘Times’, ‘Courier’	font name
FontSize	double	11.0	font size in pixel
LineStyle	string	‘solid’, ‘dotted’	line style
LineThickness	double	2.0	line thickness in pixel
FontColor, ForegroundColor, BackgroundColor	integer	FF00FF for magenta	24-bit color value in RGB format
Translucent	boolean	true, false	see text

The property ‘Translucent’ is set to *true* for GraphElements to indicate that their contained elements can be seen through the background of the containing GraphElement (see sequence diagram).

8.6 Coordinate System

The coordinate system used in the diagram interchange extension defines that the x-axis points to the east and the y-axis to the south. Position and size are defined through double values that employ pixels as the unit of measurement. The floating point type *double* allows sub-pixel accuracy. This prevents a possible loss of information when exchanging scaled or otherwise manipulated diagrams.

8.7 Position

Generally speaking, all position values used in this model are relative, i.e., any position value of a GraphElement is relative to its surrounding container. This means that the container's position is the origin for any child's position. To obtain the absolute position of a DiagramElement, it is necessary to navigate recursively through the container GraphElement until a GraphElement with no container, the root, is encountered. This root node must be a Diagram since it is only Diagrams that do not have a container.

The position of a GraphElement is specified through the StructureType point. This indicates the top left-hand corner of a GraphNode or Diagram. The x and y coordinates of a position are allowed to be negative.

For a GraphEdge, the position serves as an origin for all the DiagramElements contained. Since a GraphEdge is defined through its waypoints, position does not affect the graphical representation of the GraphEdge itself. It is drawn as a Bezier curve through all of its waypoints, which are of type BezierPoint. The control points are defined relative to base. Thus, if all control points are zero (0, 0), the result is a simple polygon line.

A Diagram does not need to have a surrounding container element. Its position is (0, 0) by default. The viewport specifies the top left-hand coordinate of a currently shown view within the Diagram. The viewport is different from (0, 0) if the view has been scrolled.

The relative coordinates allow hierarchies of nested nodes to be moved easily by changing the position of the root node. For example, moving a class can be done by changing the position of the GraphNode representing it. All the GraphElements contained automatically move with the class since they are positioned relative to it.

8.8 Size

For GraphNodes, an optional size can be provided using the DataType Dimension, which encapsulates width and height of type *double*. None of these are allowed to be negative. If the size is not given, it is determined by its semantic model and the nested DiagramElements within this GraphNode. The following two cases are to be considered:

1. If the semantic model indicates that a graphical representation for the given GraphNode is necessary, its size must be calculated from its nested DiagramElements. For example, if not given, the size of a GraphNode representing a class must be determined by recursively examining the extent of the contained compartments. If the size of a compartment is not given, its contained DiagramElements, e.g., attributes, must be analyzed and so on.

2. If the semantic model indicates an Element of the semantic model with no or a fixed graphical representation, this calculation is not necessary. An example for such a case is an AssociationEnd. Although, for example, GraphNodes containing TextElements for multiplicity and role name may be nested as contained into such a GraphNode, the AssociationEnd's GraphNode itself has no size—it is simply being used as a container. Even if the AssociationEnd is visible in a sense, e.g., a filled rhombus to indicate a composition, this is considered a fixed graphical representation which therefore does not have an extent.

8.9 Rendering Order

The general rendering order of DiagramElements is defined through the following two mechanisms:

1. The nesting tree of DiagramElements within GraphElements has the highest priority. To render the DiagramElements, the tree is processed using a preorder traversal. This means that the most deeply nested DiagramElement of a path is drawn last and thus is shown on top in z-order.

2. The ordered association container—contained between GraphElement and DiagramElement—specifies the rendering sequence of DiagramElements contained within the same GraphElement. The first DiagramElement is drawn first, the second next, and so on.

There is a less common situation in which this straightforward rendering concept alone is not sufficient. LeafElements are employed to enrich a diagram with elements not belonging to UML. It may be useful to show such a LeafElement as a background for more than one GraphElement. An example for this case is an ellipse that is drawn through an Image. The ellipse contains two classes A and B from two different packages a and b, while being drawn on top of the two packages.

Figure 8.6 illustrates this situation.

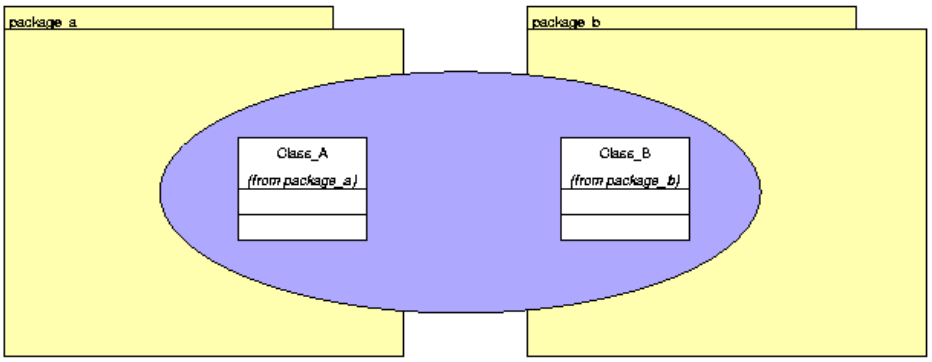


Figure 8.6 : - An ellipse contained within two paths in the tree of DiagramElements

Rendering Figure 8.6 is impossible with the two mechanisms described above, since any DiagramElement, i.e., the Image, may only be part of a single path in the tree of DiagramElements. In order to be shown in this way, it must be part of both paths. To accomplish this, the Reference is used. The Image is included in a single path as a DiagramElement. For any other path in which it is wanted to be displayed, a Reference is included. Its reference points to the Image, which indicates that it is part of both paths and is being rendered accordingly.

8.10 Semantic Model

This diagram interchange metamodel allows diagrams to be represented with an additional semantic meaning by linking an element of an existing semantic model to a GraphElement via the abstract SemanticModelBridge. Each GraphElement has a link to a concrete subclass of SemanticModelBridge. If it is not a SimpleSemanticModelElement, it references the topmost super-class of the linked semantic model. For UML 1.x this is the class Element from the package Core, which is referenced by Uml1SemanticModelBridge. For UML 2.0 and later, this can be the class Object from the package MOF, referenced by CoreSemanticModelBridge. This link allows the addition of UML-specific information to the graph. Other semantic models might be added as well, e.g., ER diagrams.

The model is designed to minimize the amount of redundant information. Due to this motivation, there is no extra attribute to indicate the semantic type of an element. In order to find out the semantic type of an element, the SemanticModelBridge must be examined. If the concrete subclass has a link to an element of the semantic model, all available semantic information about this element can be obtained.

8.11 Pre-defined Presentations of Elements

Some elements allow more than one standard presentation. For example, an actor may be shown as a rectangle or as a stickman graphic. In order to distinguish these cases, the attribute 'presentation' defined in SemanticModelBridge indicates the desired presentation. This avoids the need to create complex DiagramElements such as Images to display standard elements.

To achieve the standard visualization of an element, presentation must be set to '', the empty string. For a non-standard treatment, it must be set to 'UserDefined'. This means that it is shown exactly as specified through the DiagramElements contained in the GraphElement associated with the Element. As most elements have exactly one standard presentation, Table 8.2 lists only those explicitly that have more than one. The others are collected under the term [Single Presentation].

Table 8.2 - Permissible values for SemanticModelBridge.presentation

Element	SemanticModelBridge.presentation
[Single Presentation]	‘’ (Default), ‘UserDefined’
Actor	‘’ = ‘Stickman’, ‘Rectangle’, ‘UserDefined’
Interface	‘’ = ‘Rectangle’, ‘Circle’, ‘Lollipop’, ‘UserDefined’

8.12 Additional Semantic Information

GraphElements that have no corresponding semantic model element are linked to the subclass SimpleSemanticModelElement. Its attribute typeInfo contains semantic information for the related GraphElement. The following rules define the usage of the SimpleSemanticModelElement:

- A compartment generally does not have a corresponding model element in UML but is visualized by a GraphNode. For example, the GraphNode of an attribute compartment is linked to a SimpleSemanticModelElement with the typeInfo AttributeCompartment. For an operation compartment this would be OperationCompartment and so on.
- For an attribute part, the representing GraphElement is linked to a SimpleSemanticModelElement where its typeInfo is set to the name of the attribute. For example, for the attribute visibility of the class Feature in package Core the typeInfo is visibility.
- Diagrams have a typeInfo set to the name of the corresponding UML diagram. For example, a class diagram has the typeInfo ClassDiagram; a state diagram has the typeInfo StateDiagram, etc.

Table 8.3 contains a list of special GraphElements representing elements that do not have a model element in UML and their corresponding typeInfos.

Table 8.3 - Permissible values for SimpleSemanticModelElement.typeInfo

GraphElement	SimpleSemanticModelElement.typeInfo
Active part of a lifeline	active
Cross at the end of a lifeline	destroy
Header of a lifeline	header

8.13 Representing the Different Diagram Types of UML

The diagram interchange model can be used with the UML metamodel to represent any UML diagram type through a graph with semantic links to the UML metamodel. For most of the UML diagram types there is an intuitive way to represent them as a graph. Sequence diagrams, however, are somewhat different as discussed below.

In a class diagram, for example, classes, interfaces, and packages are represented by GraphNodes, while associations, generalizations, and dependencies are represented by GraphEdges. These GraphNodes and GraphEdges have links to the related model elements of the UML metamodel. A class may contain multiple compartments. These are represented through nested nodes of the class node with a link to a SimpleSemanticModelElement, as compartments are not part of the UML metamodel, being only part of its representation. An attribute or operation is a nested node of the compartment node with a link to the corresponding attribute or operation of the UML metamodel. An attribute itself has attributes such as visibility and type. These parts of the attribute are represented by LeafElements. If an attribute part is represented as text, the subclass TextElement is employed, otherwise an Image or GraphicPrimitive is used.

The appearance of the end of an edge, e.g., the composite of an association end, generalization, etc., is determined by the corresponding UML metamodel elements.

A special case is the n-ary association. It consists of a GraphNode representing the diamond that links the parts of the association. These parts again are shown as GraphEdges. Each of these is linked via a SemanticModelBridge to the same corresponding association. This means that several DiagramInterchange model elements (the GraphEdges) are utilized to visualize a single UML model element (the association). Figure 8.7 illustrates this mapping.

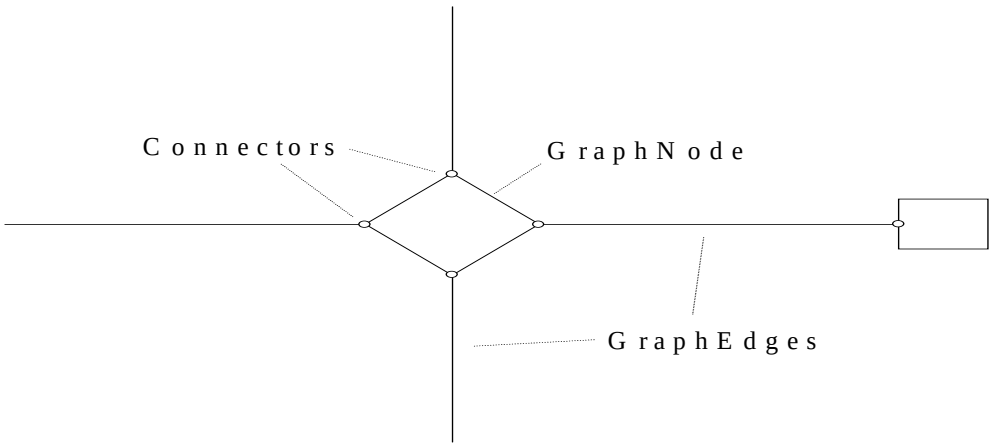


Figure 8.7 - An n-ary association

Other diagrams are represented similarly. The tool that makes use of this metamodel is responsible for the exact representation of elements that refer to semantic model elements. For example, the tool has to know that a node representing a class should be visualized through a rectangle. Its position, size, line style, etc. are determined by the elements of the metamodel.

An object in a sequence diagram is modeled as a GraphNode with a semantic model bridge to the corresponding instance. The height of this GraphNode is the entire height from the lifeline including the object at the top of the lifeline. The dashed lifeline is the only visible element of this GraphNode. The contained DiagramElements of the GraphNode are GraphNodes that represent the activated section of the lifeline. Such a GraphNode has a corresponding SimpleSemanticModelElement with the typeInfo active. Only one child of the top GraphNode has a corresponding SimpleSemanticModelElement with the typeInfo header to represent the rectangle at the top of the lifeline. The GraphNode with the TextElement is nested into this GraphNode. Another child of the top GraphNode can be a GraphNode to represent the cross at the end of a lifeline (see Figure 8.8). The cross is represented by a LeafElement. The arrows between the lifelines are GraphEdges with the stimulus as corresponding model element. The arrowhead of such a GraphEdge is not modeled explicitly, yet the type of the arrowhead can be found out through the corresponding action.

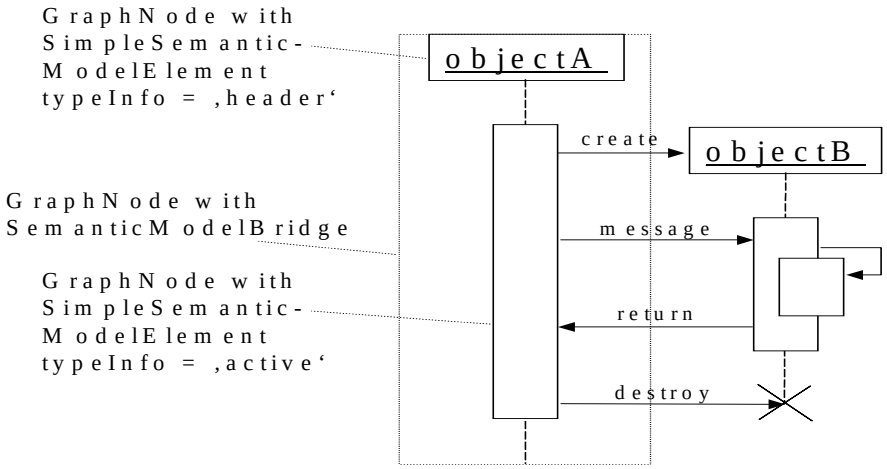


Figure 8.8 - Sequence diagram

InteractionOccurrences can overlap multiple lifelines and other elements of sequence diagrams. To indicate that the elements below the InteractionOccurrence are visible through the InteractionOccurrence, the Translucent property of the GraphNode of the InteractionOccurrence is set to true.

8.13.1 Components with Ports and Interfaces

Components are able to contain ports and interfaces that may be required or provided. Components and their contained elements are represented as GraphNodes. The interfaces are contained either in a port if one should exist or directly in the component itself. Whether the circle showing the interface is open or closed is determined solely through the semantic model, i.e., it depends upon whether the interface is required or provided. Interfaces visualized through a line with an open or closed circle are distinguishable from those shown as a rectangle by setting the attribute representation of the SemanticModelBridge to lollipop. The GraphNode of an interface shown as lollipop contains a GraphEdge for the line of the Lollipop and a GraphNode for the circle as shown in Figure 8.9.

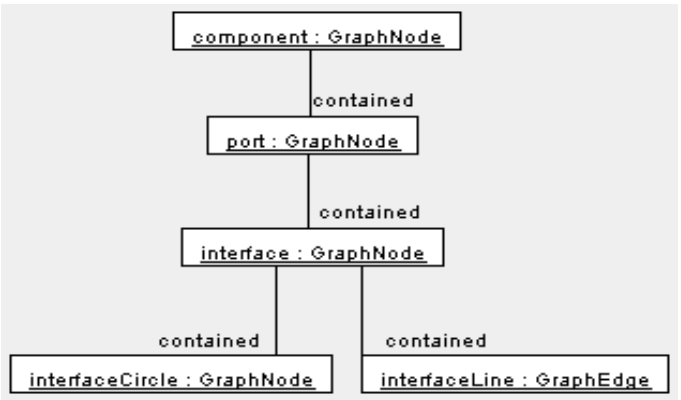


Figure 8.9 - Component

9 Derivation of Presentational Views

This chapter provides a short overview on how the meta-information for diagram interchange is represented.

Expressed in XMI, a model can be interchanged between tools that are aware of model elements. However, this format is not very consumable for the human reader neither is it well suited for tools that are purely graphically oriented. Therefore a transformation into a graphical format is needed. The format most promising for this purpose is SVG. The format used in this specification is well prepared for a transformation into SVG and was explicitly designed to make this transformation as straightforward as possible. Both data formats, XMI and SVG, are applications of XML and the common set of XML tools can be used to manipulate them. XSLT is such a mechanism that is designed to transform one XML format into another. This specification includes a set of XSLT scripts to transform a model given in XMI[UML+DI] into SVG. At the current stage this is applied to information regarding class diagrams only.

As set out in the following paragraphs, these stylesheets extract information from an XMI file with model and DI data and build a new SVG document out of this. The resulting SVG file contains the representation of a single class diagram in UML notation.

The next section describes the transformation concept for class diagrams. This is followed by a section containing the discussion about further extensions of the stylesheets. This includes, for instance, a suggestion how to deal with DI files that store several diagrams and a discussion about transformation of other diagram types. The stylesheets are designed to generate different notations with small modifications. More about this issue can be found in the last section.

9.1 General Transformation Concept

A transformation using XSLT scripts is based on a set of templates. In our implementation we need one template that matches the first class diagram in a diagram interchange file. All other information such as model data is used later on to obtain single values such as class or attribute names (script: xmi2svg.xsl).

Once a class diagram is matched, a named template is implemented to generate the SVG header on some standard elements that can be reused for the diagram elements to be created later on (script: build_diagramm.xsl).

With the usage of <apply-templates> the XSLT processor will iterate over all child elements of the diagram. These children can be node elements such as a class or an association-end or edge elements such as associations or generalizations. Templates that match these elements can be found in build_nodeelem.xsl and build_edgeelem.xsl.

The templates of build_nodeelem.xsl and build_edgeelem.xsl need to know about the kind of model element that is associated with the node or edge element matched. To obtain this information, we need to follow the link attribute. This is done by means of a couple of named templates in util_modellookup.xsl.

The model element type (e.g., class or generalization) is used as condition for calling other named templates for generating SVG elements. These named templates can be found in build_class.xsl, build_assozend.xsl, and build_edge.xsl. Each of them uses util_modellookup to get the text elements, e.g., class or operation name, which are to be represented.

Another kind of information that needs to be extracted is the model information on the association ends if an edge is represented in the diagram. In this case the scripts call a lookup routine for each of the edge ends and generate an SVG element based on the attributes returned. For instance, a navigable flag might be set for an edge end. This will lead to a generation of a solid arrow if the other edge end does not have a navigation flag set.

The representation of an edge is very simple when using the <path> element of SVG. The coordinates of the points of an edge need to be written into an attribute of the path element. To do so, we created a template that matches all points and generates a string (util_createpath.xsl).

As set out in the paragraphs above, all information for generating the SVG can be taken from diagram interchange or model interchange data. The linkage of corresponding elements makes it easy to write XSLT scripts for a general transformation to SVG.

9.2 Extensions to the Style Sheets

In the process of developing the XSLT scripts for this proof of concept we thought about extensibility of the scripts to support the following features:

1. Support of other diagram types and further elements such as interfaces, packages, parameterized classes, association classes, and dependencies (a complete list of all the elements that need to be supported can be found in the UML Notation Guide).

   This means that the model lookup routines would need to be extended for the new model element types and for the handling routines for these types. Also additional named templates will have to be written for generation of a graphical representation in SVG for these new elements.

2. Support of XLink. In the current implementation a linkage between model and diagram interchange elements is only possible through usage of simple IDRefs. For this reason, only XMI files that contain both model interchange and diagram interchange data can be transformed. This is not acceptable for a final solution.

   XSLT allows access of different files using the <xsl:for-each> command (<xsl:for-each select="document('other-document.xml')"). For a support of special cases of XLinks where only a document and an IDRef is used, an implementation could employ the string functions of XPath to extract link information. A general solution of XLink should be possible with the usage of an XLink library by an XSLT extension. All templates that need the XLink support can be found in util_modellookup.xsl.

3. Last but not least there is a need for support of transformation for files that contain more than one diagram, which is the most common case.

To accomplish this, one of the following solutions might be implemented:

- A parameterized set of scripts which still only extract one diagram is a viable alternative. An external application controls the calls of these scripts for the different diagrams. One parameter selects the extracted diagram.
- With the use of XSLT extensions, a generation of several output files can be implemented. With this feature an SVG file for each diagram could be generated.
- A solution could be a dynamic SVG that contains all diagrams and a user interface to switch between the diagrams in the SVG browser.

9.3 Transformation to Other Notations

For the proof of concept we implemented only the transformation to one graphical representation as defined in UML Notation Guide. Generally any notation is possible as long as no renderer algorithm is required. This allows modeling tools to provide XSLT scripts that generate SVG with their own notation. For example, some tools draw shadows for classes and objects or make use of colors.

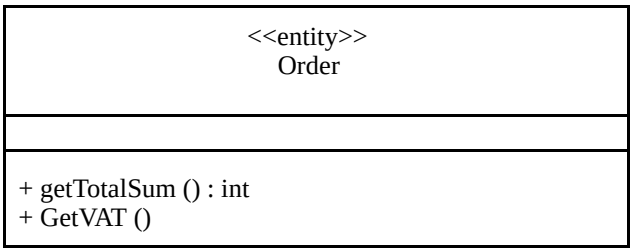
For a notation where position of graphical elements change, a renderer is needed. This renderer might be connected to the stylesheets via XSLT extensions. However, in this case, XSLT might not be the best solution.

With the following example we wish to show how simple a change to the scripts for a different notation might be. In the example given we change the representation of a class element.

For the generation of the class element the script build_class is responsible. We extracted the lines of interest from this stylesheet:

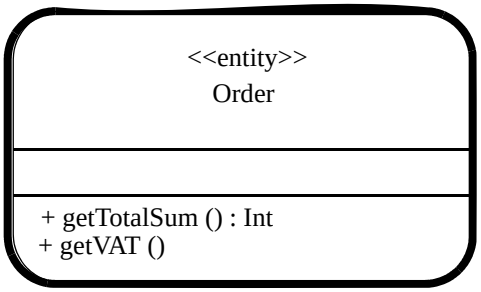
```
<g style="fill:none;stroke:black;stroke-width:1">
    <rect height="{$height}" width="{$width}" style="fill:white"/>
    <xsl:if test="$draw_attributes">
        <line y1="{$Attrib_PartStart}"y2="{$Attrib_PartStart}" x2="{$width}" style="fill:none;stroke:black;stroke-
width:1"/>
    </xsl:if>
    <xsl:if test="$draw_operations">
        <line y1="{$Operation_PartStart}" y2="{$Operation_PartStart}" x2="{$width}"
style="fill:none;stroke:black;stroke-width:1"/>
</xsl:if>
</g>
```

With this code, a class looks the way it is defined in the UML Notation Guide. A solid-outline rectangle with three compartments separated by horizontal lines will be viewed in an SVG browser.



With only a small change in this code, a class with shadow and round edges can be viewed.

```
<g style="fill:none;stroke:black;stroke-width:1">
    <rect x="2" y="2" rx="4" ry="4" height="{$height}" width="{$width}" style="fill:grey"/>
    <rect rx="4" ry="4" height="{$height}" width="{$width}" style="fill:white"/>
    <xsl:if test="$draw_attributes">
        <line y1="{$Attrib_PartStart}"  y2="{$Attrib_PartStart}" x2="{$width}" style="fill:none;stroke:black;stroke-
width:1"/>
    </xsl:if>
    <xsl:if test="$draw_operations">
<line y1="{$Operation_PartStart}" y2="{$Operation_PartStart}" x2="{$width}"
style="fill:none;stroke:black;stroke-width:1"/>
    </xsl:if>
</g>
```


<<entity>>
Order
+ getTotalSum () : Int
+ getVAT ()

10 Representation in SVG Packaging Meta-information into SVG Diagrams

As defined in the UML Notation Guide, a class diagram is a graph of *Classifier elements* connected by their various static relationships (see UML 1.4 - Chapter 3 - Notation Guide – Section 3.19.2). In our implementation, this includes classes, associations, and generalizations. Each of these elements are represented by a group in the generated SVG file. A group has no graphical representation but is used to embrace the graphical elements of a model element. For instance, a class group contains rectangles, text elements, and lines.

Following the notation guide, a class is drawn as a solid-outline rectangle with three compartments separated by horizontal lines (see UML 1.4 - Chapter 3 - Notation Guide - Section 3.22.2).

10.1 SVG

```
<!--Class Position-->
<g style="font-size:9;font-family:dialog;" onmousemove="removePopup()" onclick="class_click(evt,'Class Position')"
transform="translate(184,458)">
        <g style="fill:none;stroke:black;stroke-width:1">
                <rect style="fill:white" width="88" height="61"/>
            <line style="fill:none;stroke:black;stroke-width:1" x2="88" y2="20" y1="20"/>
            <line style="fill:none;stroke:black;stroke-width:1" x2="88" y2="41" y1="41"/>
        </g>
        <!-- top name compartment -->
        <text style="text-anchor:middle" dx="88" dy="9">Position</text>
        <!-- attribute compartment -->
        <g transform="translate(0,20)">
            <text dx="2" dy="9">#<tspan x="12">posnum</tspan> : int</text>
        </g>
        <!-- operation compartment -->
        <g transform="translate(0,41)">
            <text class="standardfont" dx="2" dy="9">+
            <tspan x="12">getPosnum</tspan>()
            </text>
        </g>
</g>
```

The top name compartment holds the class name and stereotype. Class name and stereotype are aligned in the middle of the name compartment. The stereotype value is extracted from the model and enclosed in guillemets «». Additional icons for special stereotypes and a list of properties for the class are not supported by our current XSLT scripts, extracting this information from an external icon file and the model is straightforward.

In our implementation only attribute and operation compartments are displayed, yet there is no restriction to extend the representation with compartments for exceptions or requirements. If the compartment attribute *hidden* is set, no elements of this compartment will be generated into the SVG document.

An attribute compartment is defined in the UML Notation Guide - Section 3.25.2 as represented as a list of attributes. We decided not to display a compartment name or group properties. This representation can be found in most UML tools. Not only the order but also the graphical positioning and visibility of the attributes are taken from the diagram interchange data. Attribute visibility symbol, name, type, and initial value are represented in the SVG document. Each attribute is encapsulated in separate SVG text elements.

Example: `#posnum: int`

SVG: <text dx="2" dy="9">#<tspan x="12">posnum</tspan> : int</text>

The SVG representation of the operation compartment is effected similar to the attribute compartment as a list of text elements. Each text element contains operation visibility, name, a list of parameters in braces, and the return type as defined in the UML Notation Guide - Section 3.26.2.

Example: `getPosnum( )`

SVG: <text dx="2" dy="9">+ <tspan x="12">getPosnum</tspan>() </text>

Apart from the class element we decided, for the proof of concept, to represent associations and generalizations to show how path elements are handled. As defined in the UML Notation Guide, paths consist of a series of line segments. Associations and generalizations are paths with an optional attached name and special symbols at the ends of the path.

For associations in our current implementation of XSLT scripts we allow the representation of navigability and aggregation indicators following the UML Notation Guide - Section 3.43.2. Textual elements such as role name and multiplicity are encapsulated in a group for each association end. This group contains only text elements as given by the diagram interchange data.

We define and make use of SVG elements for the aggregation symbol, navigability (an arrow), and generalization (large hollow triangle). These elements can be reused at any association or generalization end, necessitating only that a rotation angle be calculated according to the direction of the last segment they are placed on. To do so, we needed an XSLT extension to calculate the square root, which is not possible in XPath expressions and XSLT functions.

In this chapter we described how class diagrams are represented in SVG as defined in the UML Notation Guide. We have employed the grouping mechanism to embrace graphical elements that belong to the same model element. This allows us to add features such as a tooltip on classes that are shown in an SVG browser.

Annex A: Assignment of Diagram Elements

(informative)

The following table is based on the superstructure of UML 2.0. It describes which element is represented by a GraphNode or by a GraphEdge. For some elements with a more complex representation, the nested elements are set in brackets '[]'.

Element	DiagramElement	Diagram
ActionOccurrence	GraphNode	Interaction
Actor	GraphNode	Use Case
Aggregation	GraphEdge	Class/Package/Object
Artifact	GraphNode	Deployment
Association	GraphEdge	Class/Package/Object
Class	GraphNode	Class/Package/Object
Class template	GraphNode	Class/Package/Object
Collaboration	GraphNode	Composite
CollaborationOccurrence	GraphNode	Composite
CombinedFragment	GraphNode	Interaction
Component	GraphNode	Component
Component has Port	GraphNode[GraphNode]	Component
Composition	GraphEdge	Class/Package/Object
Connector	GraphEdge	Composite
Connector (Assembly)	GraphEdge[GraphNode]	Component
Coregion	GraphNode	Interaction
Dependency	GraphEdge	Class/Package/Object
Dependency	GraphEdge	Deployment
DeploymentSpecification	GraphNode	Deployment
Extend	GraphEdge	Use Case
ExtensionPoint	GraphNode	Use Case
FinalState	GraphNode	State
Frame	GraphNode	Interaction
Generalization	GraphEdge	Class/Package/Object

Generalization	GraphEdge	Deployment
GeneralOrdering	GraphEdge	Interaction
Include	GraphNode	Use Case
InstanceSpecification	GraphNode	Class/Package/Object
Instantiation	GraphEdge	Deployment
InteractionOccurrence	GraphNode	Interaction
Interface	GraphNode	Class/Package/Object
Lifeline	GraphNode	Interaction
Message	GraphEdge	Interaction
Node	GraphNode	Deployment
Package	GraphNode	Class/Package/Object
PackageExtension	GraphEdge	Class/Package/Object
PackageImport (private/public)	GraphEdge	Class/Package/Object
Part	GraphNode	Composite
Port	GraphNode	Component
Realization	GraphEdge	Class/Package/Object
Role binding	GraphEdge	Composite
State	GraphNode	State
Stop	GraphNode	Interaction
Transition	GraphEdge	State
Use Case	GraphNode	Use Case

The table below lists diagram elements which are not modeled in the UML metamodel but have a explicit representation in diagrams. They are identified by their typeInfo in the SimpleSemanticModelElement.

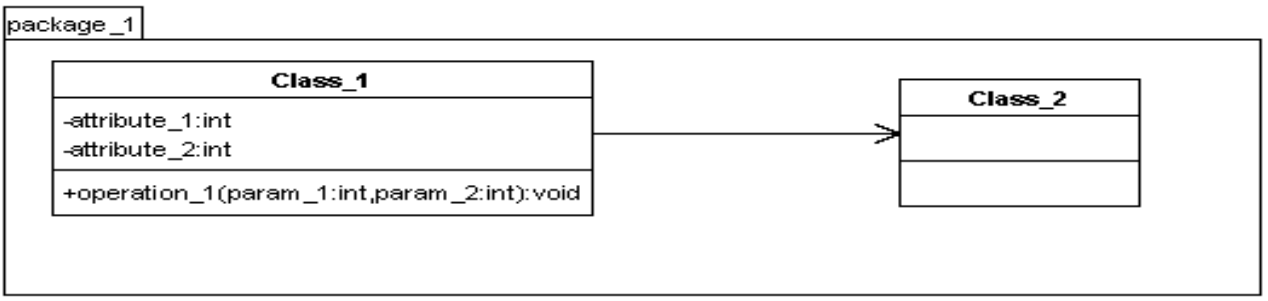
SemanticModelElement.typeInfo	**DiagramElement**
NameCompartment	GraphNode
AttributeCompartment	GraphNode
OperationCompartment	GraphNode
ClassDiagram, StateDiagram, …	Diagram
Name	GraphNode
Visibility	GraphNode
TypeSeparator	GraphNode

InitialValue	GraphNode
Multiplicity	GraphNode
Ordering	GraphNode
InterfaceCircle	GraphNode
InterfaceLine	GraphEdge

Annex B: XMI[DI] Examples

(informative)

To get an impression of the saving format described in this specification, this annex contains an excerpt from an example file with the XMI[DI] information. As example, the UML model of the diagram interchange metamodel itself is used as depicted in Figure 8.3 on page 10. The structure types of the metamodel are modeled as classes in this example.



B.1 A Simple Example

This example shows a very simple diagram and the resulting XMI file containing the semantic and diagram interchange models.

This results in the following XMI file:

```
<?xml version = '1.0' encoding = 'UTF-8' ?>
<XMI xmi.version = '1.2' xmlns:UML = 'org.omg.xmi.namespace.UML' xmlns:UML2 =
'org.omg.xmi.namespace.UML2'
 timestamp = 'Wed Jun 15 18:24:46 CEST 2005'>
 <XMI.header><XMI.documentation>
  <XMI.exporter>Netbeans XMI Writer</XMI.exporter>
  <XMI.exporterVersion>1.0</XMI.exporterVersion>
  <XMI.metaModelVersion>1.4.1</XMI.metaModelVersion></XMI.documentation>
 </XMI.header>
 <XMI.content>
  <UML:Model xmi.id = 'I11b8a6bm10480c06e65mm715e' name = 'model 2' isSpecification = 'false'
   isRoot = 'false' isLeaf = 'false' isAbstract = 'false'>
   <UML:Namespace.ownedElement>
    <UML:Package xmi.id = 'I11b8a6bm10480c06e65mm7147' name = 'package_1' visibility = 'public'
     isSpecification = 'false' isRoot = 'false' isLeaf = 'false' isAbstract = 'false'>
     <UML:Namespace.ownedElement>
      <UML:Class xmi.id = 'I11b8a6bm10480c06e65mm715a' name = 'Class_1' visibility = 'public'
```

```
isSpecification = ‘false’ isRoot = ‘false’ isLeaf = ‘false’ isAbstract = ‘false’
isActive = ‘false’>
<UML:Classifier.feature>
 <UML:Attribute xmi.id = ‘I11b8a6bm10480c06e65mm7102’ name = ‘attribute_1’
  visibility = ‘private’ isSpecification = ‘false’ ownerScope = ‘instance’
  changeability = ‘changeable’>
  <UML:StructuralFeature.type>
   <UML:DataType xmi.idref = ‘I11b8a6bm10480c06e65mm7106’/>
  </UML:StructuralFeature.type>
 </UML:Attribute>
 <UML:Attribute xmi.id = ‘I11b8a6bm10480c06e65mm70f1’ name = ‘attribute_2’
  visibility = ‘private’ isSpecification = ‘false’ ownerScope = ‘instance’
  changeability = ‘changeable’>
  <UML:StructuralFeature.type>
   <UML:DataType xmi.idref = ‘I11b8a6bm10480c06e65mm7106’/>
  </UML:StructuralFeature.type>
 </UML:Attribute>
 <UML:Operation xmi.id = ‘I11b8a6bm10480c06e65mm70e0’ name = ‘operation_1’
  visibility = ‘public’ isSpecification = ‘false’ ownerScope = ‘instance’
  isQuery = ‘false’ concurrency = ‘sequential’ isRoot = ‘false’ isLeaf = ‘false’
  isAbstract = ‘false’>
  <UML:BehavioralFeature.parameter>
   <UML:Parameter xmi.id = ‘I11b8a6bm10480c06e65mm70dd’ name = ‘return’ isSpecification = ‘false’
    kind = ‘return’>
    <UML:Parameter.type>
     <UML:DataType xmi.idref = ‘I11b8a6bm10480c06e65mm7103’/>
    </UML:Parameter.type>
   </UML:Parameter>
 <UML:Parameter xmi.id = ‘I11b8a6bm10480c06e65mm70c8’ name = ‘param_1’ isSpecification = ‘false’
    kind = ‘in’>
    <UML:Parameter.type>
     <UML:DataType xmi.idref = ‘I11b8a6bm10480c06e65mm7106’/>
    </UML:Parameter.type>
   </UML:Parameter>
 <UML:Parameter xmi.id = ‘I11b8a6bm10480c06e65mm70b1’ name = ‘param_2’ isSpecification = ‘false’
    kind = ‘in’>
    <UML:Parameter.type>
     <UML:DataType xmi.idref = ‘I11b8a6bm10480c06e65mm7106’/>
    </UML:Parameter.type>
   </UML:Parameter>
  </UML:BehavioralFeature.parameter>
 </UML:Operation>
 <UML:Method xmi.id = ‘I11b8a6bm10480c06e65mm70de’ isSpecification = ‘false’
  isQuery = ‘false’>
  <UML:Method.body>
   <UML:ProcedureExpression xmi.id = ‘I11b8a6bm10480c06e65mm70df’ language = ‘java’
    body = ‘’/>
  </UML:Method.body>
  <UML:Method.specification>
```

```
          <UML:Operation xmi.idref = 'I11b8a6bm10480c06e65mm70e0'/>
        </UML:Method.specification>
      </UML:Method>
    </UML:Classifier.feature>
  </UML:Class>
  <UML:Class xmi.id = 'I11b8a6bm10480c06e65mm713c' name = 'Class_2' visibility = 'public'
    isSpecification = 'false' isRoot = 'false' isLeaf = 'false' isAbstract = 'false'
    isActive = 'false'/>
  <UML:Association xmi.id = 'I11b8a6bm10480c06e65mm7123' isSpecification = 'false'
    isRoot = 'false' isLeaf = 'false' isAbstract = 'false'>
    <UML:Association.connection>
      <UML:AssociationEnd xmi.id = 'I11b8a6bm10480c06e65mm7129' visibility = 'public'
        isSpecification = 'false' isNavigable = 'false' ordering = 'unordered' aggregation = 'none'
        targetScope = 'instance' changeability = 'changeable'>
        <UML:AssociationEnd.multiplicity>
          <UML:Multiplicity xmi.id = 'I11b8a6bm10480c06e65mm7127'>
            <UML:Multiplicity.range>
              <UML:MultiplicityRange xmi.id = 'I11b8a6bm10480c06e65mm7128' lower = '1'
                upper = '1'/>
            </UML:Multiplicity.range>
          </UML:Multiplicity>
        </UML:AssociationEnd.multiplicity>
        <UML:AssociationEnd.participant>
          <UML:Class xmi.idref = 'I11b8a6bm10480c06e65mm715a'/>
        </UML:AssociationEnd.participant>
      </UML:AssociationEnd>
      <UML:AssociationEnd xmi.id = 'I11b8a6bm10480c06e65mm7126' visibility = 'public'
        isSpecification = 'false' isNavigable = 'true' ordering = 'unordered' aggregation = 'none'
        targetScope = 'instance' changeability = 'changeable'>
        <UML:AssociationEnd.multiplicity>
          <UML:Multiplicity xmi.id = 'I11b8a6bm10480c06e65mm7124'>
            <UML:Multiplicity.range>
              <UML:MultiplicityRange xmi.id = 'I11b8a6bm10480c06e65mm7125' lower = '1'
                upper = '1'/>
            </UML:Multiplicity.range>
          </UML:Multiplicity>
        </UML:AssociationEnd.multiplicity>
        <UML:AssociationEnd.participant>
          <UML:Class xmi.idref = 'I11b8a6bm10480c06e65mm713c'/>
        </UML:AssociationEnd.participant>
      </UML:AssociationEnd>
    </UML:Association.connection>
  </UML:Association>
 </UML:Namespace.ownedElement>
</UML:Package>
<UML:Package xmi.id = 'I11b8a6bm10480c06e65mm7104' name = 'java' isSpecification = 'false'
 isRoot = 'false' isLeaf = 'false' isAbstract = 'false'>
 <UML:Namespace.ownedElement>
  <UML:Package xmi.id = 'I11b8a6bm10480c06e65mm7105' name = 'lang' isSpecification = 'false'
```

```
        isRoot = 'false' isLeaf = 'false' isAbstract = 'false'>
        <UML:Namespace.ownedElement>
          <UML:DataType xmi.id = 'I11b8a6bm10480c06e65mm7106' name = 'int' isSpecification = 'false'
            isRoot = 'false' isLeaf = 'false' isAbstract = 'false'/>
          <UML:DataType xmi.id = 'I11b8a6bm10480c06e65mm7103' name = 'void' isSpecification = 'false'
            isRoot = 'false' isLeaf = 'false' isAbstract = 'false'/>
        </UML:Namespace.ownedElement>
      </UML:Package>
    </UML:Namespace.ownedElement>
  </UML:Package>
 </UML:Namespace.ownedElement>
</UML:Model>
<UML:Diagram xmi.id = 'I11b8a6bm10480c06e65mm715d' isVisible = 'true' name = 'Class Diagram_1'
 zoom = '1.0'>
 <UML:GraphElement.position>
  <XMI.field>0.0</XMI.field>
  <XMI.field>0.0</XMI.field>
 </UML:GraphElement.position>
 <UML:GraphNode.size>
  <XMI.field>580.0</XMI.field>
  <XMI.field>220.0</XMI.field>
 </UML:GraphNode.size>
 <UML:Diagram.viewport>
  <XMI.field>0.0</XMI.field>
  <XMI.field>0.0</XMI.field>
 </UML:Diagram.viewport>
 <UML:GraphElement.semanticModel>
  <UML:SimpleSemanticModelElement xmi.id = 'I11b8a6bm10480c06e65mm715c' presentation = ''
   typeInfo = 'ClassDiagram'/>
 </UML:GraphElement.semanticModel>
 <UML:GraphElement.contained>
  <UML:GraphNode xmi.id = 'I11b8a6bm10480c06e65mm7146' isVisible = 'true'>
   <UML:GraphElement.position>
    <XMI.field>20.0</XMI.field>
    <XMI.field>40.0</XMI.field>
   </UML:GraphElement.position>
   <UML:GraphNode.size>
    <XMI.field>520.0</XMI.field>
    <XMI.field>160.0</XMI.field>
   </UML:GraphNode.size>
   <UML:DiagramElement.property>
    <UML:Property xmi.id = 'I11b8a6bm10480c06e65mm713e' key = 'gentleware-custom-width'
     value = '520.0'/>
    <UML:Property xmi.id = 'I11b8a6bm10480c06e65mm713d' key = 'gentleware-custom-height'
     value = '160.0'/>
   </UML:DiagramElement.property>
   <UML:GraphElement.semanticModel>
    <UML:Uml1SemanticModelBridge xmi.id = 'I11b8a6bm10480c06e65mm7145' presentation = ''>
     <UML:Uml1SemanticModelBridge.element>
```

```
      <UML:Package xmi.idref = 'I11b8a6bm10480c06e65mm7147'/>
    </UML:Uml1SemanticModelBridge.element>
  </UML:Uml1SemanticModelBridge>
</UML:GraphElement.semanticModel>
<UML:GraphElement.contained>
  <UML:GraphNode xmi.id = 'I11b8a6bm10480c06e65mm7144' isVisible = 'true'>
    <UML:GraphElement.position>
      <XMI.field>0.0</XMI.field>
      <XMI.field>0.0</XMI.field>
    </UML:GraphElement.position>
    <UML:GraphNode.size>
      <XMI.field>57.8237</XMI.field>
      <XMI.field>19.0</XMI.field>
    </UML:GraphNode.size>
    <UML:GraphElement.semanticModel>
      <UML:SimpleSemanticModelElement xmi.id = 'I11b8a6bm10480c06e65mm7143' presentation = ''
        typeInfo = 'NameCompartment'/>
    </UML:GraphElement.semanticModel>
    <UML:GraphElement.contained>
      <UML:GraphNode xmi.id = 'I11b8a6bm10480c06e65mm7142' isVisible = 'true'>
        <UML:GraphElement.position>
          <XMI.field>2.0</XMI.field>
          <XMI.field>2.0</XMI.field>
        </UML:GraphElement.position>
        <UML:GraphNode.size>
          <XMI.field>53.8237</XMI.field>
          <XMI.field>15.0</XMI.field>
        </UML:GraphNode.size>
        <UML:GraphElement.semanticModel>
          <UML:SimpleSemanticModelElement xmi.id = 'I11b8a6bm10480c06e65mm7141' presentation = ''
            typeInfo = 'Name'/>
        </UML:GraphElement.semanticModel>
      </UML:GraphNode>
    </UML:GraphElement.contained>
  </UML:GraphNode>
  <UML:GraphNode xmi.id = 'I11b8a6bm10480c06e65mm7140' isVisible = 'true'>
    <UML:GraphElement.position>
      <XMI.field>0.0</XMI.field>
      <XMI.field>18.0</XMI.field>
    </UML:GraphElement.position>
    <UML:GraphNode.size>
      <XMI.field>520.0</XMI.field>
      <XMI.field>142.0</XMI.field>
    </UML:GraphNode.size>
    <UML:GraphElement.semanticModel>
      <UML:SimpleSemanticModelElement xmi.id = 'I11b8a6bm10480c06e65mm713f' presentation = ''
        typeInfo = 'BodyCompartment'/>
    </UML:GraphElement.semanticModel>
    <UML:GraphElement.contained>
```

```
<UML:GraphNode xmi.id = 'I11b8a6bm10480c06e65mm7159' isVisible = 'true'>
 <UML:GraphElement.position>
  <XMI.field>20.0</XMI.field>
  <XMI.field>12.0</XMI.field>
 </UML:GraphElement.position>
 <UML:GraphNode.size>
  <XMI.field>224.3228</XMI.field>
  <XMI.field>86.0</XMI.field>
 </UML:GraphNode.size>
 <UML:GraphElement.semanticModel>
  <UML:Uml1SemanticModelBridge xmi.id = 'I11b8a6bm10480c06e65mm7158' presentation = ''>
   <UML:Uml1SemanticModelBridge.element>
    <UML:Class xmi.idref = 'I11b8a6bm10480c06e65mm715a'/>
   </UML:Uml1SemanticModelBridge.element>
  </UML:Uml1SemanticModelBridge>
 </UML:GraphElement.semanticModel>
 <UML:GraphElement.contained>
  <UML:GraphNode xmi.id = 'I11b8a6bm10480c06e65mm7157' isVisible = 'true'>
   <UML:GraphElement.position>
    <XMI.field>1.0</XMI.field>
    <XMI.field>1.0</XMI.field>
   </UML:GraphElement.position>
   <UML:GraphNode.size>
    <XMI.field>222.3228</XMI.field>
    <XMI.field>19.0</XMI.field>
   </UML:GraphNode.size>
   <UML:GraphElement.semanticModel>
    <UML:SimpleSemanticModelElement xmi.id = 'I11b8a6bm10480c06e65mm7156' presentation = ''
      typeInfo = 'NameCompartment'/>
   </UML:GraphElement.semanticModel>
   <UML:GraphElement.contained>
    <UML:GraphNode xmi.id = 'I11b8a6bm10480c06e65mm7155' isVisible = 'true'>
     <UML:GraphElement.position>
      <XMI.field>90.3672</XMI.field>
      <XMI.field>2.0</XMI.field>
     </UML:GraphElement.position>
     <UML:GraphNode.size>
      <XMI.field>41.5884</XMI.field>
      <XMI.field>15.0</XMI.field>
     </UML:GraphNode.size>
     <UML:GraphElement.semanticModel>
     <UML:SimpleSemanticModelElement xmi.id = 'I11b8a6bm10480c06e65mm7154' presentation = ''
        typeInfo = 'Name'/>
     </UML:GraphElement.semanticModel>
    </UML:GraphNode>
   </UML:GraphElement.contained>
  </UML:GraphNode>
  <UML:GraphNode xmi.id = 'I11b8a6bm10480c06e65mm7153' isVisible = 'true'>
   <UML:GraphElement.position>
```

```
    <XMI.field>1.0</XMI.field>
    <XMI.field>20.0</XMI.field>
   </UML:GraphElement.position>
   <UML:GraphNode.size>
    <XMI.field>222.3228</XMI.field>
    <XMI.field>1.0</XMI.field>
   </UML:GraphNode.size>
   <UML:GraphElement.semanticModel>
    <UML:SimpleSemanticModelElement xmi.id = 'I11b8a6bm10480c06e65mm7152' presentation = ''
      typeInfo = 'CompartmentSeparator'/>
   </UML:GraphElement.semanticModel>
  </UML:GraphNode>
  <UML:GraphNode xmi.id = 'I11b8a6bm10480c06e65mm7151' isVisible = 'true'>
   <UML:GraphElement.position>
    <XMI.field>1.0</XMI.field>
    <XMI.field>21.0</XMI.field>
   </UML:GraphElement.position>
   <UML:GraphNode.size>
    <XMI.field>222.3228</XMI.field>
    <XMI.field>39.0</XMI.field>
   </UML:GraphNode.size>
   <UML:GraphElement.semanticModel>
    <UML:SimpleSemanticModelElement xmi.id = 'I11b8a6bm10480c06e65mm7150' presentation = ''
      typeInfo = 'AttributeCompartment'/>
   </UML:GraphElement.semanticModel>
   <UML:GraphElement.contained>
    <UML:GraphNode xmi.id = 'I11b8a6bm10480c06e65mm714f' isVisible = 'true'>
     <UML:GraphElement.position>
      <XMI.field>2.0</XMI.field>
      <XMI.field>2.0</XMI.field>
     </UML:GraphElement.position>
     <UML:GraphNode.size>
      <XMI.field>218.3228</XMI.field>
      <XMI.field>35.0</XMI.field>
     </UML:GraphNode.size>
     <UML:GraphElement.semanticModel>
    <UML:SimpleSemanticModelElement xmi.id = 'I11b8a6bm10480c06e65mm714e' presentation = ''
        typeInfo = 'DelimitedSection'/>
     </UML:GraphElement.semanticModel>
     <UML:GraphElement.contained>
      <UML:GraphNode xmi.id = 'I11b8a6bm10480c06e65mm7101' isVisible = 'true'>
       <UML:GraphElement.position>
        <XMI.field>2.0</XMI.field>
        <XMI.field>2.0</XMI.field>
       </UML:GraphElement.position>
       <UML:GraphNode.size>
        <XMI.field>214.3228</XMI.field>
        <XMI.field>15.0</XMI.field>
       </UML:GraphNode.size>
```

```
  <UML:DiagramElement.property>
   <UML:Property xmi.id = 'I11b8a6bm10480c06e65mm70ff' key = 'gentleware-custom-width'
    value = '0.0'/>
   <UML:Property xmi.id = 'I11b8a6bm10480c06e65mm70fe' key = 'gentleware-custom-height'
    value = '0.0'/>
  </UML:DiagramElement.property>
  <UML:GraphElement.semanticModel>
<UML:Uml1SemanticModelBridge xmi.id = 'I11b8a6bm10480c06e65mm7100' presentation = ''>
    <UML:Uml1SemanticModelBridge.element>
     <UML:Attribute xmi.idref = 'I11b8a6bm10480c06e65mm7102'/>
    </UML:Uml1SemanticModelBridge.element>
   </UML:Uml1SemanticModelBridge>
  </UML:GraphElement.semanticModel>
  <UML:GraphElement.contained>
   <UML:GraphNode xmi.id = 'I11b8a6bm10480c06e65mm70fd' isVisible = 'true'>
    <UML:GraphElement.position>
     <XMI.field>0.0</XMI.field>
     <XMI.field>0.0</XMI.field>
    </UML:GraphElement.position>
    <UML:GraphNode.size>
     <XMI.field>3.6631</XMI.field>
     <XMI.field>15.0</XMI.field>
    </UML:GraphNode.size>
    <UML:GraphElement.semanticModel>
     <UML:SimpleSemanticModelElement xmi.id = 'I11b8a6bm10480c06e65mm70fc'
      presentation = '' typeInfo = 'Visibility'/>
    </UML:GraphElement.semanticModel>
   </UML:GraphNode>
   <UML:GraphNode xmi.id = 'I11b8a6bm10480c06e65mm70fb' isVisible = 'true'>
    <UML:GraphElement.position>
     <XMI.field>3.6631</XMI.field>
     <XMI.field>0.0</XMI.field>
    </UML:GraphElement.position>
    <UML:GraphNode.size>
     <XMI.field>51.9814</XMI.field>
     <XMI.field>15.0</XMI.field>
    </UML:GraphNode.size>
    <UML:GraphElement.semanticModel>
     <UML:SimpleSemanticModelElement xmi.id = 'I11b8a6bm10480c06e65mm70fa'
      presentation = '' typeInfo = 'Name'/>
    </UML:GraphElement.semanticModel>
   </UML:GraphNode>
   <UML:GraphNode xmi.id = 'I11b8a6bm10480c06e65mm70f9' isVisible = 'true'>
    <UML:GraphElement.position>
     <XMI.field>55.6445</XMI.field>
     <XMI.field>0.0</XMI.field>
    </UML:GraphElement.position>
    <UML:GraphNode.size>
     <XMI.field>3.0562</XMI.field>
```

```
    <XMI.field>15.0</XMI.field>
   </UML:GraphNode.size>
   <UML:GraphElement.semanticModel>
    <UML:SimpleSemanticModelElement xmi.id = 'I11b8a6bm10480c06e65mm70f8'
     presentation = '' typeInfo = 'TypeSeparator'/>
   </UML:GraphElement.semanticModel>
  </UML:GraphNode>
  <UML:GraphNode xmi.id = 'I11b8a6bm10480c06e65mm70f7' isVisible = 'true'>
   <UML:GraphElement.position>
    <XMI.field>58.7007</XMI.field>
    <XMI.field>0.0</XMI.field>
   </UML:GraphElement.position>
   <UML:GraphNode.size>
    <XMI.field>11.6177</XMI.field>
    <XMI.field>15.0</XMI.field>
   </UML:GraphNode.size>
   <UML:GraphElement.semanticModel>
    <UML:SimpleSemanticModelElement xmi.id = 'I11b8a6bm10480c06e65mm70f6'
     presentation = '' typeInfo = 'StructuralFeatureType'/>
   </UML:GraphElement.semanticModel>
   <UML:GraphElement.contained>
    <UML:GraphNode xmi.id = 'I11b8a6bm10480c06e65mm70f5' isVisible = 'true'>
     <UML:GraphElement.position>
      <XMI.field>0.0</XMI.field>
      <XMI.field>0.0</XMI.field>
     </UML:GraphElement.position>
     <UML:GraphNode.size>
      <XMI.field>11.6177</XMI.field>
      <XMI.field>15.0</XMI.field>
     </UML:GraphNode.size>
     <UML:GraphElement.semanticModel>
      <UML:Uml1SemanticModelBridge xmi.id = 'I11b8a6bm10480c06e65mm70f4'
       presentation = ''>
       <UML:Uml1SemanticModelBridge.element>
        <UML:DataType xmi.idref = 'I11b8a6bm10480c06e65mm7106'/>
       </UML:Uml1SemanticModelBridge.element>
      </UML:Uml1SemanticModelBridge>
     </UML:GraphElement.semanticModel>
     <UML:GraphElement.contained>
      <UML:GraphNode xmi.id = 'I11b8a6bm10480c06e65mm70f3' isVisible = 'true'>
       <UML:GraphElement.position>
        <XMI.field>0.0</XMI.field>
        <XMI.field>0.0</XMI.field>
       </UML:GraphElement.position>
       <UML:GraphNode.size>
        <XMI.field>11.6177</XMI.field>
        <XMI.field>15.0</XMI.field>
       </UML:GraphNode.size>
       <UML:GraphElement.semanticModel>
```

```
            <UML:SimpleSemanticModelElement xmi.id = 'I11b8a6bm10480c06e65mm70f2'
              presentation = '' typeInfo = 'Name'/>
           </UML:GraphElement.semanticModel>
          </UML:GraphNode>
         </UML:GraphElement.contained>
        </UML:GraphNode>
       </UML:GraphElement.contained>
      </UML:GraphNode>
     </UML:GraphElement.contained>
    </UML:GraphNode>
    <UML:GraphNode xmi.id = 'I11b8a6bm10480c06e65mm70f0' isVisible = 'true'>
     <UML:GraphElement.position>
      <XMI.field>2.0</XMI.field>
      <XMI.field>18.0</XMI.field>
     </UML:GraphElement.position>
     <UML:GraphNode.size>
      <XMI.field>214.3228</XMI.field>
      <XMI.field>15.0</XMI.field>
     </UML:GraphNode.size>
     <UML:DiagramElement.property>
      <UML:Property xmi.id = 'I11b8a6bm10480c06e65mm70ee' key = 'gentleware-custom-width'
       value = '0.0'/>
      <UML:Property xmi.id = 'I11b8a6bm10480c06e65mm70ed' key = 'gentleware-custom-height'
       value = '0.0'/>
     </UML:DiagramElement.property>
     <UML:GraphElement.semanticModel>
      <UML:Uml1SemanticModelBridge xmi.id = 'I11b8a6bm10480c06e65mm70ef' presentation = ''>
       <UML:Uml1SemanticModelBridge.element>
        <UML:Attribute xmi.idref = 'I11b8a6bm10480c06e65mm70f1'/>
       </UML:Uml1SemanticModelBridge.element>
      </UML:Uml1SemanticModelBridge>
     </UML:GraphElement.semanticModel>
     <UML:GraphElement.contained>
      <UML:GraphNode xmi.id = 'I11b8a6bm10480c06e65mm70ec' isVisible = 'true'>
       <UML:GraphElement.position>
        <XMI.field>0.0</XMI.field>
        <XMI.field>0.0</XMI.field>
       </UML:GraphElement.position>
       <UML:GraphNode.size>
        <XMI.field>3.6631</XMI.field>
        <XMI.field>15.0</XMI.field>
       </UML:GraphNode.size>
       <UML:GraphElement.semanticModel>
        <UML:SimpleSemanticModelElement xmi.id = 'I11b8a6bm10480c06e65mm70eb'
         presentation = '' typeInfo = 'Visibility'/>
       </UML:GraphElement.semanticModel>
      </UML:GraphNode>
      <UML:GraphNode xmi.id = 'I11b8a6bm10480c06e65mm70ea' isVisible = 'true'>
       <UML:GraphElement.position>
```

```
    <XMI.field>3.6631</XMI.field>
    <XMI.field>0.0</XMI.field>
   </UML:GraphElement.position>
   <UML:GraphNode.size>
    <XMI.field>51.9814</XMI.field>
    <XMI.field>15.0</XMI.field>
   </UML:GraphNode.size>
   <UML:GraphElement.semanticModel>
    <UML:SimpleSemanticModelElement xmi.id = 'I11b8a6bm10480c06e65mm70e9'
     presentation = '' typeInfo = 'Name'/>
   </UML:GraphElement.semanticModel>
  </UML:GraphNode>
  <UML:GraphNode xmi.id = 'I11b8a6bm10480c06e65mm70e8' isVisible = 'true'>
   <UML:GraphElement.position>
    <XMI.field>55.6445</XMI.field>
    <XMI.field>0.0</XMI.field>
   </UML:GraphElement.position>
   <UML:GraphNode.size>
    <XMI.field>3.0562</XMI.field>
    <XMI.field>15.0</XMI.field>
   </UML:GraphNode.size>
   <UML:GraphElement.semanticModel>
    <UML:SimpleSemanticModelElement xmi.id = 'I11b8a6bm10480c06e65mm70e7'
     presentation = '' typeInfo = 'TypeSeparator'/>
   </UML:GraphElement.semanticModel>
  </UML:GraphNode>
  <UML:GraphNode xmi.id = 'I11b8a6bm10480c06e65mm70e6' isVisible = 'true'>
   <UML:GraphElement.position>
    <XMI.field>58.7007</XMI.field>
    <XMI.field>0.0</XMI.field>
   </UML:GraphElement.position>
   <UML:GraphNode.size>
    <XMI.field>11.6177</XMI.field>
    <XMI.field>15.0</XMI.field>
   </UML:GraphNode.size>
   <UML:GraphElement.semanticModel>
    <UML:SimpleSemanticModelElement xmi.id = 'I11b8a6bm10480c06e65mm70e5'
     presentation = '' typeInfo = 'StructuralFeatureType'/>
   </UML:GraphElement.semanticModel>
   <UML:GraphElement.contained>
    <UML:GraphNode xmi.id = 'I11b8a6bm10480c06e65mm70e4' isVisible = 'true'>
     <UML:GraphElement.position>
      <XMI.field>0.0</XMI.field>
      <XMI.field>0.0</XMI.field>
     </UML:GraphElement.position>
     <UML:GraphNode.size>
      <XMI.field>11.6177</XMI.field>
      <XMI.field>15.0</XMI.field>
     </UML:GraphNode.size>
```

```
                <UML:GraphElement.semanticModel>
                 <UML:Uml1SemanticModelBridge xmi.id = 'I11b8a6bm10480c06e65mm70e3'
                    presentation = ''>
                  <UML:Uml1SemanticModelBridge.element>
                   <UML:DataType xmi.idref = 'I11b8a6bm10480c06e65mm7106'/>
                  </UML:Uml1SemanticModelBridge.element>
                 </UML:Uml1SemanticModelBridge>
                </UML:GraphElement.semanticModel>
                <UML:GraphElement.contained>
                 <UML:GraphNode xmi.id = 'I11b8a6bm10480c06e65mm70e2' isVisible = 'true'>
                  <UML:GraphElement.position>
                   <XMI.field>0.0</XMI.field>
                   <XMI.field>0.0</XMI.field>
                  </UML:GraphElement.position>
                  <UML:GraphNode.size>
                   <XMI.field>11.6177</XMI.field>
                   <XMI.field>15.0</XMI.field>
                  </UML:GraphNode.size>
                  <UML:GraphElement.semanticModel>
                   <UML:SimpleSemanticModelElement xmi.id = 'I11b8a6bm10480c06e65mm70e1'
                     presentation = '' typeInfo = 'Name'/>
                  </UML:GraphElement.semanticModel>
                 </UML:GraphNode>
                </UML:GraphElement.contained>
               </UML:GraphNode>
              </UML:GraphElement.contained>
             </UML:GraphNode>
            </UML:GraphElement.contained>
           </UML:GraphNode>
          </UML:GraphElement.contained>
         </UML:GraphNode>
        </UML:GraphElement.contained>
       </UML:GraphNode>
      </UML:GraphElement.contained>
     </UML:GraphNode>
    </UML:GraphElement.contained>
   </UML:GraphNode>
  </UML:GraphElement.contained>
 </UML:GraphNode>
 <UML:GraphNode xmi.id = 'I11b8a6bm10480c06e65mm714d' isVisible = 'true'>
  <UML:GraphElement.position>
   <XMI.field>1.0</XMI.field>
   <XMI.field>60.0</XMI.field>
  </UML:GraphElement.position>
  <UML:GraphNode.size>
   <XMI.field>222.3228</XMI.field>
   <XMI.field>1.0</XMI.field>
  </UML:GraphNode.size>
  <UML:GraphElement.semanticModel>
   <UML:SimpleSemanticModelElement xmi.id = 'I11b8a6bm10480c06e65mm714c'
    presentation = '' typeInfo = 'CompartmentSeparator'/>
  </UML:GraphElement.semanticModel>
 </UML:GraphNode>
 <UML:GraphNode xmi.id = 'I11b8a6bm10480c06e65mm714b' isVisible = 'true'>
  <UML:GraphElement.position>
```

```
  <XMI.field>1.0</XMI.field>
  <XMI.field>61.0</XMI.field>
</UML:GraphElement.position>
<UML:GraphNode.size>
  <XMI.field>222.3228</XMI.field>
  <XMI.field>24.0</XMI.field>
</UML:GraphNode.size>
<UML:GraphElement.semanticModel>
  <UML:SimpleSemanticModelElement xmi.id = 'I11b8a6bm10480c06e65mm714a'
    presentation = '' typeInfo = 'OperationCompartment'/>
</UML:GraphElement.semanticModel>
<UML:GraphElement.contained>
  <UML:GraphNode xmi.id = 'I11b8a6bm10480c06e65mm7149' isVisible = 'true'>
    <UML:GraphElement.position>
      <XMI.field>2.0</XMI.field>
      <XMI.field>2.0</XMI.field>
    </UML:GraphElement.position>
    <UML:GraphNode.size>
      <XMI.field>218.3228</XMI.field>
      <XMI.field>20.0</XMI.field>
    </UML:GraphNode.size>
    <UML:GraphElement.semanticModel>
      <UML:SimpleSemanticModelElement xmi.id = 'I11b8a6bm10480c06e65mm7148'
    presentation = '' typeInfo = 'DelimitedSection'/>
    </UML:GraphElement.semanticModel>
    <UML:GraphElement.contained>
      <UML:GraphNode xmi.id = 'I11b8a6bm10480c06e65mm70dc' isVisible = 'true'>
        <UML:GraphElement.position>
          <XMI.field>2.0</XMI.field>
          <XMI.field>2.0</XMI.field>
        </UML:GraphElement.position>
        <UML:GraphNode.size>
          <XMI.field>214.3228</XMI.field>
          <XMI.field>15.0</XMI.field>
        </UML:GraphNode.size>
        <UML:DiagramElement.property>
          <UML:Property xmi.id = 'I11b8a6bm10480c06e65mm70da' key = 'gentleware-custom-width'
            value = '0.0'/>
         <UML:Property xmi.id = 'I11b8a6bm10480c06e65mm70d9' key = 'gentleware-custom-height'
            value = '0.0'/>
        </UML:DiagramElement.property>
        <UML:GraphElement.semanticModel>
     <UML:Uml1SemanticModelBridge xmi.id = 'I11b8a6bm10480c06e65mm70db' presentation = ''>
          <UML:Uml1SemanticModelBridge.element>
            <UML:Operation xmi.idref = 'I11b8a6bm10480c06e65mm70e0'/>
          </UML:Uml1SemanticModelBridge.element>
        </UML:Uml1SemanticModelBridge>
      </UML:GraphElement.semanticModel>
      <UML:GraphElement.contained>
```

```
<UML:GraphNode xmi.id = 'I11b8a6bm10480c06e65mm70d8' isVisible = 'true'>
 <UML:GraphElement.position>
  <XMI.field>0.0</XMI.field>
  <XMI.field>0.0</XMI.field>
 </UML:GraphElement.position>
 <UML:GraphNode.size>
  <XMI.field>6.4238</XMI.field>
  <XMI.field>15.0</XMI.field>
 </UML:GraphNode.size>
 <UML:GraphElement.semanticModel>
  <UML:SimpleSemanticModelElement xmi.id = 'I11b8a6bm10480c06e65mm70d7'
   presentation = '' typeInfo = 'Visibility'/>
 </UML:GraphElement.semanticModel>
</UML:GraphNode>
<UML:GraphNode xmi.id = 'I11b8a6bm10480c06e65mm70d6' isVisible = 'true'>
 <UML:GraphElement.position>
  <XMI.field>6.4238</XMI.field>
  <XMI.field>0.0</XMI.field>
 </UML:GraphElement.position>
 <UML:GraphNode.size>
  <XMI.field>58.1045</XMI.field>
  <XMI.field>15.0</XMI.field>
 </UML:GraphNode.size>
 <UML:GraphElement.semanticModel>
  <UML:SimpleSemanticModelElement xmi.id = 'I11b8a6bm10480c06e65mm70d5'
   presentation = '' typeInfo = 'Name'/>
 </UML:GraphElement.semanticModel>
</UML:GraphNode>
<UML:GraphNode xmi.id = 'I11b8a6bm10480c06e65mm70b0' isVisible = 'true'>
 <UML:GraphElement.position>
  <XMI.field>64.5283</XMI.field>
  <XMI.field>0.0</XMI.field>
 </UML:GraphElement.position>
 <UML:GraphNode.size>
  <XMI.field>3.6631</XMI.field>
  <XMI.field>15.0</XMI.field>
 </UML:GraphNode.size>
 <UML:GraphElement.semanticModel>
  <UML:SimpleSemanticModelElement xmi.id = 'I11b8a6bm10480c06e65mm70af'
   presentation = '' typeInfo = 'ParameterStart'/>
 </UML:GraphElement.semanticModel>
</UML:GraphNode>
<UML:GraphNode xmi.id = 'I11b8a6bm10480c06e65mm70ae' isVisible = 'true'>
 <UML:GraphElement.position>
  <XMI.field>68.1914</XMI.field>
  <XMI.field>0.0</XMI.field>
 </UML:GraphElement.position>
 <UML:GraphNode.size>
  <XMI.field>58.0884</XMI.field>
```

```
        <XMI.field>15.0</XMI.field>
      </UML:GraphNode.size>
      <UML:GraphElement.semanticModel>
<UML:Uml1SemanticModelBridge xmi.id = 'I11b8a6bm10480c06e65mm70ad' presentation = ''>
        <UML:Uml1SemanticModelBridge.element>
          <UML:Parameter xmi.idref = 'I11b8a6bm10480c06e65mm70c8'/>
        </UML:Uml1SemanticModelBridge.element>
      </UML:Uml1SemanticModelBridge>
      </UML:GraphElement.semanticModel>
      <UML:GraphElement.contained>
       <UML:GraphNode xmi.id = 'I11b8a6bm10480c06e65mm70ac' isVisible = 'true'>
        <UML:GraphElement.position>
          <XMI.field>0.0</XMI.field>
          <XMI.field>0.0</XMI.field>
        </UML:GraphElement.position>
        <UML:GraphNode.size>
          <XMI.field>43.4146</XMI.field>
          <XMI.field>15.0</XMI.field>
        </UML:GraphNode.size>
        <UML:GraphElement.semanticModel>
          <UML:SimpleSemanticModelElement xmi.id = 'I11b8a6bm10480c06e65mm70ab'
            presentation = '' typeInfo = 'Name'/>
        </UML:GraphElement.semanticModel>
       </UML:GraphNode>
       <UML:GraphNode xmi.id = 'I11b8a6bm10480c06e65mm70aa' isVisible = 'true'>
        <UML:GraphElement.position>
          <XMI.field>43.4146</XMI.field>
          <XMI.field>0.0</XMI.field>
        </UML:GraphElement.position>
        <UML:GraphNode.size>
          <XMI.field>3.0562</XMI.field>
          <XMI.field>15.0</XMI.field>
        </UML:GraphNode.size>
        <UML:GraphElement.semanticModel>
          <UML:SimpleSemanticModelElement xmi.id = 'I11b8a6bm10480c06e65mm70a9'
            presentation = '' typeInfo = 'TypeSeparator'/>
        </UML:GraphElement.semanticModel>
       </UML:GraphNode>
       <UML:GraphNode xmi.id = 'I11b8a6bm10480c06e65mm70a8' isVisible = 'true'>
        <UML:GraphElement.position>
          <XMI.field>46.4707</XMI.field>
          <XMI.field>0.0</XMI.field>
        </UML:GraphElement.position>
        <UML:GraphNode.size>
          <XMI.field>11.6177</XMI.field>
          <XMI.field>15.0</XMI.field>
        </UML:GraphNode.size>
        <UML:GraphElement.semanticModel>
          <UML:Uml1SemanticModelBridge xmi.id = 'I11b8a6bm10480c06e65mm70a7'
```

```
          presentation = ‘’>
        <UML:Uml1SemanticModelBridge.element>
         <UML:DataType xmi.idref = ‘I11b8a6bm10480c06e65mm7106’/>
        </UML:Uml1SemanticModelBridge.element>
       </UML:Uml1SemanticModelBridge>
      </UML:GraphElement.semanticModel>
      <UML:GraphElement.contained>
       <UML:GraphNode xmi.id = ‘I11b8a6bm10480c06e65mm70a6’ isVisible = ‘true’>
        <UML:GraphElement.position>
         <XMI.field>0.0</XMI.field>
         <XMI.field>0.0</XMI.field>
        </UML:GraphElement.position>
        <UML:GraphNode.size>
         <XMI.field>11.6177</XMI.field>
         <XMI.field>15.0</XMI.field>
        </UML:GraphNode.size>
        <UML:GraphElement.semanticModel>
         <UML:SimpleSemanticModelElement xmi.id = ‘I11b8a6bm10480c06e65mm70a5’
          presentation = ‘’ typeInfo = ‘Name’/>
        </UML:GraphElement.semanticModel>
       </UML:GraphNode>
      </UML:GraphElement.contained>
     </UML:GraphNode>
    </UML:GraphElement.contained>
   </UML:GraphNode>
   <UML:GraphNode xmi.id = ‘I11b8a6bm10480c06e65mm70a4’ isVisible = ‘true’>
    <UML:GraphElement.position>
     <XMI.field>126.2798</XMI.field>
     <XMI.field>0.0</XMI.field>
    </UML:GraphElement.position>
    <UML:GraphNode.size>
     <XMI.field>3.0562</XMI.field>
     <XMI.field>15.0</XMI.field>
    </UML:GraphNode.size>
    <UML:GraphElement.semanticModel>
     <UML:SimpleSemanticModelElement xmi.id = ‘I11b8a6bm10480c06e65mm70a3’
      presentation = ‘’ typeInfo = ‘ParameterSeparator’/>
    </UML:GraphElement.semanticModel>
   </UML:GraphNode>
   <UML:GraphNode xmi.id = ‘I11b8a6bm10480c06e65mm70a2’ isVisible = ‘true’>
    <UML:GraphElement.position>
     <XMI.field>129.3359</XMI.field>
     <XMI.field>0.0</XMI.field>
    </UML:GraphElement.position>
    <UML:GraphNode.size>
     <XMI.field>58.0884</XMI.field>
     <XMI.field>15.0</XMI.field>
    </UML:GraphNode.size>
    <UML:GraphElement.semanticModel>
```

```
<UML:Uml1SemanticModelBridge xmi.id = 'I11b8a6bm10480c06e65mm70a1' presentation = ''>
  <UML:Uml1SemanticModelBridge.element>
    <UML:Parameter xmi.idref = 'I11b8a6bm10480c06e65mm70b1'/>
  </UML:Uml1SemanticModelBridge.element>
 </UML:Uml1SemanticModelBridge>
</UML:GraphElement.semanticModel>
<UML:GraphElement.contained>
 <UML:GraphNode xmi.id = 'I11b8a6bm10480c06e65mm70a0' isVisible = 'true'>
  <UML:GraphElement.position>
   <XMI.field>0.0</XMI.field>
   <XMI.field>0.0</XMI.field>
  </UML:GraphElement.position>
  <UML:GraphNode.size>
   <XMI.field>43.4146</XMI.field>
   <XMI.field>15.0</XMI.field>
  </UML:GraphNode.size>
  <UML:GraphElement.semanticModel>
   <UML:SimpleSemanticModelElement xmi.id = 'I11b8a6bm10480c06e65mm709f'
    presentation = '' typeInfo = 'Name'/>
  </UML:GraphElement.semanticModel>
 </UML:GraphNode>
 <UML:GraphNode xmi.id = 'I11b8a6bm10480c06e65mm709e' isVisible = 'true'>
  <UML:GraphElement.position>
   <XMI.field>43.4146</XMI.field>
   <XMI.field>0.0</XMI.field>
  </UML:GraphElement.position>
  <UML:GraphNode.size>
   <XMI.field>3.0562</XMI.field>
   <XMI.field>15.0</XMI.field>
  </UML:GraphNode.size>
  <UML:GraphElement.semanticModel>
   <UML:SimpleSemanticModelElement xmi.id = 'I11b8a6bm10480c06e65mm709d'
    presentation = '' typeInfo = 'TypeSeparator'/>
  </UML:GraphElement.semanticModel>
 </UML:GraphNode>
 <UML:GraphNode xmi.id = 'I11b8a6bm10480c06e65mm709c' isVisible = 'true'>
  <UML:GraphElement.position>
   <XMI.field>46.4707</XMI.field>
   <XMI.field>0.0</XMI.field>
  </UML:GraphElement.position>
  <UML:GraphNode.size>
   <XMI.field>11.6177</XMI.field>
   <XMI.field>15.0</XMI.field>
  </UML:GraphNode.size>
  <UML:GraphElement.semanticModel>
   <UML:Uml1SemanticModelBridge xmi.id = 'I11b8a6bm10480c06e65mm709b'
    presentation = ''>
    <UML:Uml1SemanticModelBridge.element>
     <UML:DataType xmi.idref = 'I11b8a6bm10480c06e65mm7106'/>
```

```
        </UML:Uml1SemanticModelBridge.element>
      </UML:Uml1SemanticModelBridge>
    </UML:GraphElement.semanticModel>
    <UML:GraphElement.contained>
     <UML:GraphNode xmi.id = 'I11b8a6bm10480c06e65mm709a' isVisible = 'true'>
      <UML:GraphElement.position>
       <XMI.field>0.0</XMI.field>
       <XMI.field>0.0</XMI.field>
      </UML:GraphElement.position>
      <UML:GraphNode.size>
       <XMI.field>11.6177</XMI.field>
       <XMI.field>15.0</XMI.field>
      </UML:GraphNode.size>
      <UML:GraphElement.semanticModel>
       <UML:SimpleSemanticModelElement xmi.id = 'I11b8a6bm10480c06e65mm7099'
         presentation = '' typeInfo = 'Name'/>
      </UML:GraphElement.semanticModel>
     </UML:GraphNode>
    </UML:GraphElement.contained>
   </UML:GraphNode>
  </UML:GraphElement.contained>
</UML:GraphNode>
<UML:GraphNode xmi.id = 'I11b8a6bm10480c06e65mm7098' isVisible = 'true'>
  <UML:GraphElement.position>
   <XMI.field>187.4243</XMI.field>
   <XMI.field>0.0</XMI.field>
  </UML:GraphElement.position>
  <UML:GraphNode.size>
   <XMI.field>3.6631</XMI.field>
   <XMI.field>15.0</XMI.field>
  </UML:GraphNode.size>
  <UML:GraphElement.semanticModel>
   <UML:SimpleSemanticModelElement xmi.id = 'I11b8a6bm10480c06e65mm7097'
    presentation = '' typeInfo = 'ParameterEnd'/>
  </UML:GraphElement.semanticModel>
</UML:GraphNode>
<UML:GraphNode xmi.id = 'I11b8a6bm10480c06e65mm7096' isVisible = 'true'>
  <UML:GraphElement.position>
   <XMI.field>191.0874</XMI.field>
   <XMI.field>0.0</XMI.field>
  </UML:GraphElement.position>
  <UML:GraphNode.size>
   <XMI.field>3.0562</XMI.field>
   <XMI.field>15.0</XMI.field>
  </UML:GraphNode.size>
  <UML:GraphElement.semanticModel>
   <UML:SimpleSemanticModelElement xmi.id = 'I11b8a6bm10480c06e65mm7095'
    presentation= '' typeInfo = 'TypeSeparator'/>
  </UML:GraphElement.semanticModel>
```

```
    </UML:GraphNode>
    <UML:GraphNode xmi.id = 'I11b8a6bm10480c06e65mm7094' isVisible = 'true'>
     <UML:GraphElement.position>
      <XMI.field>194.1436</XMI.field>
      <XMI.field>0.0</XMI.field>
     </UML:GraphElement.position>
     <UML:GraphNode.size>
      <XMI.field>20.1792</XMI.field>
      <XMI.field>15.0</XMI.field>
     </UML:GraphNode.size>
     <UML:GraphElement.semanticModel>
<UML:Uml1SemanticModelBridge xmi.id = 'I11b8a6bm10480c06e65mm7093' presentation = ''>
       <UML:Uml1SemanticModelBridge.element>
        <UML:Parameter xmi.idref = 'I11b8a6bm10480c06e65mm70dd'/>
       </UML:Uml1SemanticModelBridge.element>
      </UML:Uml1SemanticModelBridge>
     </UML:GraphElement.semanticModel>
     <UML:GraphElement.contained>
      <UML:GraphNode xmi.id = 'I11b8a6bm10480c06e65mm7092' isVisible = 'true'>
       <UML:GraphElement.position>
        <XMI.field>0.0</XMI.field>
        <XMI.field>0.0</XMI.field>
       </UML:GraphElement.position>
       <UML:GraphNode.size>
        <XMI.field>20.1792</XMI.field>
        <XMI.field>15.0</XMI.field>
       </UML:GraphNode.size>
       <UML:GraphElement.semanticModel>
        <UML:Uml1SemanticModelBridge xmi.id = 'I11b8a6bm10480c06e65mm7091'
          presentation = ''>
         <UML:Uml1SemanticModelBridge.element>
          <UML:DataType xmi.idref = 'I11b8a6bm10480c06e65mm7103'/>
         </UML:Uml1SemanticModelBridge.element>
        </UML:Uml1SemanticModelBridge>
       </UML:GraphElement.semanticModel>
       <UML:GraphElement.contained>
        <UML:GraphNode xmi.id = 'I11b8a6bm10480c06e65mm7090' isVisible = 'true'>
         <UML:GraphElement.position>
          <XMI.field>0.0</XMI.field>
          <XMI.field>0.0</XMI.field>
         </UML:GraphElement.position>
         <UML:GraphNode.size>
          <XMI.field>20.1792</XMI.field>
          <XMI.field>15.0</XMI.field>
         </UML:GraphNode.size>
         <UML:GraphElement.semanticModel>
          <UML:SimpleSemanticModelElement xmi.id = 'I11b8a6bm10480c06e65mm708f'
            presentation = '' typeInfo = 'Name'/>
         </UML:GraphElement.semanticModel>
```

```
                </UML:GraphNode>
              </UML:GraphElement.contained>
            </UML:GraphNode>
          </UML:GraphElement.contained>
        </UML:GraphNode>
      </UML:GraphElement.contained>
    </UML:GraphNode>
  </UML:GraphElement.contained>
  <UML:GraphElement.anchorage>
    <UML:GraphConnector xmi.id = 'I11b8a6bm10480c06e65mm7122'>
      <UML:GraphConnector.position>
        <XMI.field>224.3228</XMI.field>
        <XMI.field>40.0</XMI.field>
      </UML:GraphConnector.position>
      <UML:GraphConnector.graphEdge>
        <UML:GraphEdge xmi.idref = 'I11b8a6bm10480c06e65mm7120'/>
      </UML:GraphConnector.graphEdge>
    </UML:GraphConnector>
  </UML:GraphElement.anchorage>
</UML:GraphNode>
<UML:GraphNode xmi.id = 'I11b8a6bm10480c06e65mm713b' isVisible = 'true'>
  <UML:GraphElement.position>
    <XMI.field>370.0</XMI.field>
    <XMI.field>22.0</XMI.field>
  </UML:GraphElement.position>
  <UML:GraphNode.size>
    <XMI.field>100.0</XMI.field>
    <XMI.field>71.0</XMI.field>
  </UML:GraphNode.size>
  <UML:GraphElement.semanticModel>
    <UML:Uml1SemanticModelBridge xmi.id = 'I11b8a6bm10480c06e65mm713a' presentation = ''>
      <UML:Uml1SemanticModelBridge.element>
        <UML:Class xmi.idref = 'I11b8a6bm10480c06e65mm713c'/>
      </UML:Uml1SemanticModelBridge.element>
    </UML:Uml1SemanticModelBridge>
  </UML:GraphElement.semanticModel>
  <UML:GraphElement.contained>
    <UML:GraphNode xmi.id = 'I11b8a6bm10480c06e65mm7139' isVisible = 'true'>
      <UML:GraphElement.position>
        <XMI.field>1.0</XMI.field>
        <XMI.field>1.0</XMI.field>
      </UML:GraphElement.position>
      <UML:GraphNode.size>
        <XMI.field>98.0</XMI.field>
        <XMI.field>19.0</XMI.field>
```

```
  </UML:GraphNode.size>
  <UML:GraphElement.semanticModel>
   <UML:SimpleSemanticModelElement xmi.id = 'I11b8a6bm10480c06e65mm7138'
    presentation = '' typeInfo = 'NameCompartment'/>
  </UML:GraphElement.semanticModel>
  <UML:GraphElement.contained>
   <UML:GraphNode xmi.id = 'I11b8a6bm10480c06e65mm7137' isVisible = 'true'>
    <UML:GraphElement.position>
     <XMI.field>28.2058</XMI.field>
     <XMI.field>2.0</XMI.field>
    </UML:GraphElement.position>
    <UML:GraphNode.size>
     <XMI.field>41.5884</XMI.field>
     <XMI.field>15.0</XMI.field>
    </UML:GraphNode.size>
    <UML:GraphElement.semanticModel>
     <UML:SimpleSemanticModelElement xmi.id = 'I11b8a6bm10480c06e65mm7136'
      presentation = '' typeInfo = 'Name'/>
    </UML:GraphElement.semanticModel>
   </UML:GraphNode>
  </UML:GraphElement.contained>
 </UML:GraphNode>
 <UML:GraphNode xmi.id = 'I11b8a6bm10480c06e65mm7135' isVisible = 'true'>
  <UML:GraphElement.position>
   <XMI.field>1.0</XMI.field>
   <XMI.field>20.0</XMI.field>
  </UML:GraphElement.position>
  <UML:GraphNode.size>
   <XMI.field>98.0</XMI.field>
   <XMI.field>1.0</XMI.field>
  </UML:GraphNode.size>
  <UML:GraphElement.semanticModel>
   <UML:SimpleSemanticModelElement xmi.id = 'I11b8a6bm10480c06e65mm7134'
    presentation = '' typeInfo = 'CompartmentSeparator'/>
  </UML:GraphElement.semanticModel>
 </UML:GraphNode>
 <UML:GraphNode xmi.id = 'I11b8a6bm10480c06e65mm7133' isVisible = 'true'>
  <UML:GraphElement.position>
   <XMI.field>1.0</XMI.field>
   <XMI.field>21.0</XMI.field>
  </UML:GraphElement.position>
  <UML:GraphNode.size>
   <XMI.field>98.0</XMI.field>
   <XMI.field>24.0</XMI.field>
  </UML:GraphNode.size>
  <UML:GraphElement.semanticModel>
   <UML:SimpleSemanticModelElement xmi.id = 'I11b8a6bm10480c06e65mm7132' presentation = ''
    typeInfo = 'AttributeCompartment'/>
  </UML:GraphElement.semanticModel>
```

```
 <UML:GraphElement.contained>
  <UML:GraphNode xmi.id = 'I11b8a6bm10480c06e65mm7131' isVisible = 'true'>
   <UML:GraphElement.position>
    <XMI.field>2.0</XMI.field>
    <XMI.field>2.0</XMI.field>
   </UML:GraphElement.position>
   <UML:GraphNode.size>
    <XMI.field>94.0</XMI.field>
    <XMI.field>20.0</XMI.field>
   </UML:GraphNode.size>
   <UML:GraphElement.semanticModel>
  <UML:SimpleSemanticModelElement xmi.id = 'I11b8a6bm10480c06e65mm7130' presentation = ''
      typeInfo = 'DelimitedSection'/>
   </UML:GraphElement.semanticModel>
  </UML:GraphNode>
 </UML:GraphElement.contained>
</UML:GraphNode>
<UML:GraphNode xmi.id = 'I11b8a6bm10480c06e65mm712f' isVisible = 'true'>
 <UML:GraphElement.position>
  <XMI.field>1.0</XMI.field>
  <XMI.field>45.0</XMI.field>
 </UML:GraphElement.position>
 <UML:GraphNode.size>
  <XMI.field>98.0</XMI.field>
  <XMI.field>1.0</XMI.field>
 </UML:GraphNode.size>
 <UML:GraphElement.semanticModel>
  <UML:SimpleSemanticModelElement xmi.id = 'I11b8a6bm10480c06e65mm712e' presentation = ''
   typeInfo = 'CompartmentSeparator'/>
 </UML:GraphElement.semanticModel>
</UML:GraphNode>
<UML:GraphNode xmi.id = 'I11b8a6bm10480c06e65mm712d' isVisible = 'true'>
 <UML:GraphElement.position>
  <XMI.field>1.0</XMI.field>
  <XMI.field>46.0</XMI.field>
 </UML:GraphElement.position>
 <UML:GraphNode.size>
  <XMI.field>98.0</XMI.field>
  <XMI.field>24.0</XMI.field>
 </UML:GraphNode.size>
 <UML:GraphElement.semanticModel>
  <UML:SimpleSemanticModelElement xmi.id = 'I11b8a6bm10480c06e65mm712c' presentation = ''
   typeInfo = 'OperationCompartment'/>
 </UML:GraphElement.semanticModel>
 <UML:GraphElement.contained>
  <UML:GraphNode xmi.id = 'I11b8a6bm10480c06e65mm712b' isVisible = 'true'>
   <UML:GraphElement.position>
    <XMI.field>2.0</XMI.field>
    <XMI.field>2.0</XMI.field>
```

```
        </UML:GraphElement.position>
        <UML:GraphNode.size>
          <XMI.field>94.0</XMI.field>
          <XMI.field>20.0</XMI.field>
        </UML:GraphNode.size>
        <UML:GraphElement.semanticModel>
       <UML:SimpleSemanticModelElement xmi.id = 'I11b8a6bm10480c06e65mm712a' presentation = ''
            typeInfo = 'DelimitedSection'/>
        </UML:GraphElement.semanticModel>
       </UML:GraphNode>
     </UML:GraphElement.contained>
   </UML:GraphNode>
  </UML:GraphElement.contained>
  <UML:GraphElement.anchorage>
    <UML:GraphConnector xmi.id = 'I11b8a6bm10480c06e65mm7121'>
     <UML:GraphConnector.position>
       <XMI.field>0.0</XMI.field>
       <XMI.field>30.0</XMI.field>
     </UML:GraphConnector.position>
     <UML:GraphConnector.graphEdge>
       <UML:GraphEdge xmi.idref = 'I11b8a6bm10480c06e65mm7120'/>
     </UML:GraphConnector.graphEdge>
    </UML:GraphConnector>
  </UML:GraphElement.anchorage>
</UML:GraphNode>
<UML:GraphEdge xmi.id = 'I11b8a6bm10480c06e65mm7120' isVisible = 'true'>
  <UML:GraphElement.position>
    <XMI.field>0.0</XMI.field>
    <XMI.field>0.0</XMI.field>
  </UML:GraphElement.position>
  <UML:GraphEdge.waypoints>
    <XMI.field>
      <XMI.field>244.3228</XMI.field>
      <XMI.field>52.0</XMI.field>
    </XMI.field>
    <XMI.field>
      <XMI.field>0.0</XMI.field>
      <XMI.field>0.0</XMI.field>
    </XMI.field>
    <XMI.field>
      <XMI.field>0.0</XMI.field>
      <XMI.field>0.0</XMI.field>
    </XMI.field>
    <XMI.field>
      <XMI.field>370.0</XMI.field>
      <XMI.field>52.0</XMI.field>
    </XMI.field>
    <XMI.field>
      <XMI.field>0.0</XMI.field>
```

```
    <XMI.field>0.0</XMI.field>
   </XMI.field>
   <XMI.field>
    <XMI.field>0.0</XMI.field>
    <XMI.field>0.0</XMI.field>
   </XMI.field>
  </UML:GraphEdge.waypoints>
  <UML:GraphElement.semanticModel>
   <UML:Uml1SemanticModelBridge xmi.id = 'I11b8a6bm10480c06e65mm711f' presentation = ''>
    <UML:Uml1SemanticModelBridge.element>
     <UML:Association xmi.idref = 'I11b8a6bm10480c06e65mm7123'/>
    </UML:Uml1SemanticModelBridge.element>
   </UML:Uml1SemanticModelBridge>
  </UML:GraphElement.semanticModel>
  <UML:GraphElement.contained>
   <UML:GraphNode xmi.id = 'I11b8a6bm10480c06e65mm711e' isVisible = 'true'>
    <UML:GraphElement.position>
     <XMI.field>244.3228</XMI.field>
     <XMI.field>52.0</XMI.field>
    </UML:GraphElement.position>
    <UML:GraphNode.size>
     <XMI.field>0.0</XMI.field>
     <XMI.field>0.0</XMI.field>
    </UML:GraphNode.size>
    <UML:DiagramElement.property>
     <UML:Property xmi.id = 'I11b8a6bm10480c06e65mm711c' key = 'gentleware-custom-width'
      value = '0.0'/>
     <UML:Property xmi.id = 'I11b8a6bm10480c06e65mm711b' key = 'gentleware-custom-height'
      value = '0.0'/>
    </UML:DiagramElement.property>
    <UML:GraphElement.semanticModel>
     <UML:Uml1SemanticModelBridge xmi.id = 'I11b8a6bm10480c06e65mm711d' presentation = ''>
      <UML:Uml1SemanticModelBridge.element>
       <UML:AssociationEnd xmi.idref = 'I11b8a6bm10480c06e65mm7129'/>
      </UML:Uml1SemanticModelBridge.element>
     </UML:Uml1SemanticModelBridge>
    </UML:GraphElement.semanticModel>
    <UML:GraphElement.contained>
     <UML:GraphNode xmi.id = 'I11b8a6bm10480c06e65mm711a' isVisible = 'false'>
      <UML:GraphElement.position>
       <XMI.field>12.9904</XMI.field>
       <XMI.field>4.798</XMI.field>
      </UML:GraphElement.position>
      <UML:GraphNode.size>
       <XMI.field>37.2969</XMI.field>
       <XMI.field>15.0</XMI.field>
      </UML:GraphNode.size>
      <UML:GraphElement.semanticModel>
     <UML:SimpleSemanticModelElement xmi.id = 'I11b8a6bm10480c06e65mm7119' presentation = ''
```

```
        typeInfo = 'Name'/>
      </UML:GraphElement.semanticModel>
    </UML:GraphNode>
    <UML:GraphNode xmi.id = 'I11b8a6bm10480c06e65mm7118' isVisible = 'false'>
      <UML:GraphElement.position>
        <XMI.field>4.5666</XMI.field>
        <XMI.field>4.798</XMI.field>
      </UML:GraphElement.position>
      <UML:GraphNode.size>
        <XMI.field>6.4238</XMI.field>
        <XMI.field>15.0</XMI.field>
      </UML:GraphNode.size>
      <UML:GraphElement.semanticModel>
    <UML:SimpleSemanticModelElement xmi.id = 'I11b8a6bm10480c06e65mm7117' presentation = ''
        typeInfo = 'Visibility'/>
      </UML:GraphElement.semanticModel>
    </UML:GraphNode>
    <UML:GraphNode xmi.id = 'I11b8a6bm10480c06e65mm7116' isVisible = 'false'>
      <UML:GraphElement.position>
        <XMI.field>12.9904</XMI.field>
        <XMI.field>-22.5</XMI.field>
      </UML:GraphElement.position>
      <UML:GraphNode.size>
        <XMI.field>6.1177</XMI.field>
        <XMI.field>15.0</XMI.field>
      </UML:GraphNode.size>
      <UML:GraphElement.semanticModel>
    <UML:SimpleSemanticModelElement xmi.id = 'I11b8a6bm10480c06e65mm7115' presentation = ''
        typeInfo = 'Multiplicity'/>
      </UML:GraphElement.semanticModel>
    </UML:GraphNode>
  </UML:GraphElement.contained>
</UML:GraphNode>
<UML:GraphNode xmi.id = 'I11b8a6bm10480c06e65mm7114' isVisible = 'true'>
  <UML:GraphElement.position>
    <XMI.field>370.0</XMI.field>
    <XMI.field>52.0</XMI.field>
  </UML:GraphElement.position>
  <UML:GraphNode.size>
    <XMI.field>0.0</XMI.field>
    <XMI.field>0.0</XMI.field>
  </UML:GraphNode.size>
  <UML:DiagramElement.property>
    <UML:Property xmi.id = 'I11b8a6bm10480c06e65mm7112' key = 'gentleware-custom-width'
      value = '0.0'/>
    <UML:Property xmi.id = 'I11b8a6bm10480c06e65mm7111' key = 'gentleware-custom-height'
      value = '0.0'/>
  </UML:DiagramElement.property>
  <UML:GraphElement.semanticModel>
```

```
 <UML:Uml1SemanticModelBridge xmi.id = 'I11b8a6bm10480c06e65mm7113' presentation = ''>
  <UML:Uml1SemanticModelBridge.element>
   <UML:AssociationEnd xmi.idref = 'I11b8a6bm10480c06e65mm7126'/>
  </UML:Uml1SemanticModelBridge.element>
 </UML:Uml1SemanticModelBridge>
</UML:GraphElement.semanticModel>
<UML:GraphElement.contained>
 <UML:GraphNode xmi.id = 'I11b8a6bm10480c06e65mm7110' isVisible = 'false'>
  <UML:GraphElement.position>
   <XMI.field>-50.2873</XMI.field>
   <XMI.field>-19.798</XMI.field>
  </UML:GraphElement.position>
  <UML:GraphNode.size>
   <XMI.field>37.2969</XMI.field>
   <XMI.field>15.0</XMI.field>
  </UML:GraphNode.size>
  <UML:GraphElement.semanticModel>
 <UML:SimpleSemanticModelElement xmi.id = 'I11b8a6bm10480c06e65mm710f' presentation = ''
     typeInfo = 'Name'/>
  </UML:GraphElement.semanticModel>
 </UML:GraphNode>
 <UML:GraphNode xmi.id = 'I11b8a6bm10480c06e65mm710e' isVisible = 'false'>
  <UML:GraphElement.position>
   <XMI.field>-58.7111</XMI.field>
   <XMI.field>-19.798</XMI.field>
  </UML:GraphElement.position>
  <UML:GraphNode.size>
   <XMI.field>6.4238</XMI.field>
   <XMI.field>15.0</XMI.field>
  </UML:GraphNode.size>
  <UML:GraphElement.semanticModel>
<UML:SimpleSemanticModelElement xmi.id = 'I11b8a6bm10480c06e65mm710d' presentation = ''
     typeInfo = 'Visibility'/>
  </UML:GraphElement.semanticModel>
 </UML:GraphNode>
 <UML:GraphNode xmi.id = 'I11b8a6bm10480c06e65mm710c' isVisible = 'false'>
  <UML:GraphElement.position>
   <XMI.field>-19.1081</XMI.field>
   <XMI.field>7.5</XMI.field>
  </UML:GraphElement.position>
  <UML:GraphNode.size>
   <XMI.field>6.1177</XMI.field>
   <XMI.field>15.0</XMI.field>
  </UML:GraphNode.size>
  <UML:GraphElement.semanticModel>
<UML:SimpleSemanticModelElement xmi.id = 'I11b8a6bm10480c06e65mm710b' presentation = ''
     typeInfo = 'Multiplicity'/>
  </UML:GraphElement.semanticModel>
 </UML:GraphNode>
```

```
          </UML:GraphElement.contained>
        </UML:GraphNode>
        <UML:GraphNode xmi.id = 'I11b8a6bm10480c06e65mm710a' isVisible = 'false'>
          <UML:GraphElement.position>
            <XMI.field>279.3211</XMI.field>
            <XMI.field>62.0</XMI.field>
          </UML:GraphElement.position>
          <UML:GraphNode.size>
            <XMI.field>95.6807</XMI.field>
            <XMI.field>15.0</XMI.field>
          </UML:GraphNode.size>
          <UML:GraphElement.semanticModel>
           <UML:SimpleSemanticModelElement xmi.id = 'I11b8a6bm10480c06e65mm7109' presentation = ''
             typeInfo = 'DirectedName'/>
          </UML:GraphElement.semanticModel>
          <UML:GraphElement.contained>
            <UML:GraphNode xmi.id = 'I11b8a6bm10480c06e65mm7108' isVisible = 'true'>
              <UML:GraphElement.position>
                <XMI.field>0.0</XMI.field>
                <XMI.field>0.0</XMI.field>
              </UML:GraphElement.position>
              <UML:GraphNode.size>
                <XMI.field>95.6807</XMI.field>
                <XMI.field>15.0</XMI.field>
              </UML:GraphNode.size>
              <UML:GraphElement.semanticModel>
             <UML:SimpleSemanticModelElement xmi.id = 'I11b8a6bm10480c06e65mm7107' presentation = ''
                 typeInfo = 'Name'/>
              </UML:GraphElement.semanticModel>
            </UML:GraphNode>
          </UML:GraphElement.contained>
        </UML:GraphNode>
      </UML:GraphElement.contained>
      <UML:GraphEdge.anchor>
        <UML:GraphConnector xmi.idref = 'I11b8a6bm10480c06e65mm7122'/>
        <UML:GraphConnector xmi.idref = 'I11b8a6bm10480c06e65mm7121'/>
      </UML:GraphEdge.anchor>
    </UML:GraphEdge>
  </UML:GraphElement.contained>
 </UML:GraphNode>
</UML:GraphElement.contained>
<UML:Diagram.owner>
 <UML:Uml1SemanticModelBridge xmi.id = 'I11b8a6bm10480c06e65mm715b' presentation = ''>
  <UML:Uml1SemanticModelBridge.element>
   <UML:Model xmi.idref = 'I11b8a6bm10480c06e65mm715e'/>
  </UML:Uml1SemanticModelBridge.element>
 </UML:Uml1SemanticModelBridge>
```

```
      </UML:Diagram.owner>
    </UML:Diagram>
  </XMI.content>
</XMI>
```

B.2 Full Example

This example is a representation of the Diagram Interchange Metamodel itself. However, due to the rather large size the resulting XMI file is not reproduced here but included in the accompanying ZIP file - OMG document ptc/05-06-07.

Annex C: Nesting Guide

(informative)

The nesting guide specifies the hierarchies of diagram elements to represent model elements of the UML 1.4. This guide specifies for example that e.g., a ‘class’ is represented by a GraphNode. This GraphNode contains GraphNodes for the compartments, etc. It also specifies which semantic model elements are associated to the diagram elements or which typeInfos are used for diagram elements that do not refer directly to a semantic model element.

C.1 Notation

This guide shows only the container-contained relations between the diagram elements. The important information includes the link to the SemanticModelBridge too. The nesting relations spans a tree. The node of the tree are diagram elements that can have a link to a SemanticModelBridge. The diagram element is shown with its name (e.g., GraphNode). If it has a SemanticModelBridge, it is shown in curly braces ‘{}’. If the SemanticModelBridge is a SimpleSemanticModelElement, the typeInfo is written in quotation marks “”, if it is a CoreSemanticModelBridge it is written in left and right angle quotes <>. So a class can be represented as ‘class = GraphNode{<class>}’ and an attribute compartment can be represented as ‘GraphNode{“AttributeCompartement”}’.

This means the GraphNode representing a class is associated via the SemanticModelBridge to the semantic model element ‘class.’ And the GraphNode representing an attribute compartment is associated to a SimpleSemanticModelElement with the typeInfo ‘AttributeCompartment.’

If the SemanticModelBridge has a non-empty presentation, it is written in the curly braces before the typeInfo or model element type, e.g., an Actor is written as ‘GraphNode{“classifier”, ,actor>}’ if it is represented as a classifier and not as a sticky man.

The nesting of elements is shown in rounded braces (). So a short representation of a class is ‘GraphNode{,class>}(NameCompartment, AttributeCompartment, . . .)’. The elements nested in the brackets can be placeholders like NameCompartment which are specified in this guide too.

If an element is optional, it is followed by a ‘?’ or between brackets []. If it can be repeated multiple times, it is followed by an asterisk ‘*’. If you can choose one of many alternatives, they are separated with a vertical line ‘|’. Lists of elements are separated with a comma ‘,’.

An expression can have parameters. Parameters are passed in curly braces {} (without <> or “”). For example, the expression ‘Name{SimpleState}’ with the definition for Name:

Name: Name{ModelElement}

DI-Element: GraphNode{<ModelElement >}

is evaluated to GraphNode{<SimpleState>}.

The following specifications define for each semantic model element the associated hierarchy of nested diagram elements. The names of the specifications are only used as crosslinks in this document.

C.2 Autonomous Elements

Some elements are used in more than one diagram type.

C.2.1 Class Diagram

Name: **DiClass**

DI-Element: GraphNode{<Class>}

Nested elements: NameCompartment, [CompartmentSeparator, AttributeCompartment], [CompartmentSeparator, OperationCompartment], [CompartmentSeparator, ToolCompartment]*

alternate representation:

DI-Element: GraphNode{"StereotypeImage", <Class>}

Nested elements: DiStereotypeImage, [NameCompartment]

Name: **DiPackage**

DI-Element: GraphNode{<Package>}

Nested elements: NameCompartment, BodyCompartment

Name: **DiAssociationClass**

DI-Element: GraphNode{<AssociationClass>}

Nested elements: [Stereotypes], [DirectedName], [Properties], GraphEdge{“AssociationClassEdge”}, AssociationClassNode, DiAssociationEnd, DiAssociationEnd

Name: **DiAssociationClass**

DI-Element: GraphNode{<AssociationClass>}

Nested elements: [Stereotypes], [DirectedName], [Properties], GraphEdge{“AssociationClassEdge”}, AssociationClassNode, DiAssociationEnd, DiAssociationEnd

Name: **AssociationClassNode**

DI-Element: GraphNode{“ AssociationClassNode ”}

Nested elements: NameCompartment, [CompartmentSeparator, AttributeCompartment], [CompartmentSeparator, OperationCompartment], [CompartmentSeparator, ToolCompartment]*

Name: **DiGeneralization**

DI-Element: GraphEdge{<Generalization >}

Nested elements: [GraphNode {“Discriminator”}], [Name], [Stereotype]

Name: **DiAssociationEnd**

DI-Element: GraphNode{<AssociationEnd >}

Nested elements: [RoleAdornment], [Multiplicity], [Properties]

Use-case Diagram

Name: **DiActor**

DI-Element: GraphNode{<Actor >}

Nested elements: ModelElementBasics

alternate representation

DI-Element: GraphNode{"Classifer", <Actor>}

Nested elements: NameCompartment, [CompartmentSeparator, AttributeCompartment], [CompartmentSeparator, OperationCompartment], [CompartmentSeparator, ToolCompartment]*

Name: **DiUseCase**

DI-Element: GraphNode{<UseCase >}

Nested elements: NameCompartment, [CompartmentSeparator, AttributeCompartment], [CompartmentSeparator, OperationCompartment], [CompartmentSeparator, ExtensionPointCompartment], [CompartmentSeparator, ToolCompartment]*

Name: **DiInclude**

DI-Element: GraphEdge{<Include >}

Nested elements: [StereotypeCompartment], [Name]

Name: **DiExtend**

DI-Element: GraphEdge{<Extend >}

Nested elements: [StereotypeCompartment], [Name]

Activity diagram

Name: **DiObjectFlowState**

DI-Element: GraphNode{<ObjectFlowState>}

Nested elements: [NameCompartment], InstanceType | InstanceTypeInState

Name: **DiActionState**

DI-Element: GraphNode{<ActionState>}

Nested elements: [StereotypeCompartment], CallAction

State diagram

Name: **DiCompositeState**

DI-Element: GraphNode{<CompositeState>}

Nested elements: [StereotypeCompartment], NameCompartment, CompartmentSeparator, InternalTransitionCompartment, (SubStateCompartment | RegionCompartment)

Name: **DiSimpleState**

DI-Element: GraphNode{<SimpleState>}

Nested elements: [StereotypeCompartment], NameCompartment, CompartmentSeparator, InternalTransitionCompartment

Name: **DiPseudostate**

DI-Element: GraphNode{<Pseudostate>}

Nested elements: [StereotypeCompartment], [Name]

Name: **DiFinalState**

DI-Element: GraphNode{<FinalState>}

Nested elements: [StereotypeCompartment], [Name]

Name: **DiSynchState**

DI-Element: GraphNode{<SynchState>}

Nested elements: [StereotypeCompartment], [Name]

Name: **CallAction**

DI-Element: GraphNode{<CallAction>}

Nested elements: [GraphNode{“Label”}], [GraphNode{“EffectStart”}], [GraphNode{“Expression”}]

Name: **ReturnAction**

DI-Element: GraphNode{<ReturnAction>}

Nested elements: [GraphNode{“Label”}], [GraphNode{“EffectStart”}], [GraphNode{“Expression”}]

Name: **DestroyAction**

DI-Element: GraphNode{<DestroyAction>}

Nested elements: [GraphNode{“Label”}], [GraphNode{“EffectStart”}], [GraphNode{“Expression”}]

Name: **DiActionState**

DI-Element: GraphNode{<ActionState>}

Nested elements: [EntryAction], [Stereotypes]

Name: **DiTransition**

DI-Element: GraphEdge{<Transition>}

Nested elements: [Stereotypes], [Name], TransitionDescription

Name: **TransitionDescription**

DI-Element: GraphNode{“TransitionDescription”}

Nested elements: [Name{CallEvent}], [GraphNode{“GuardStart “}], [Guard], [GraphNode{“ GuardEnd “}], [GraphNode{“ EffectStart “}], [CallAction]

Name: **Guard**

DI-Element: GraphNode{<Guard>}

Nested elements: GraphNode{“Expression”}

Sequence diagrams

Name: **DiObject**

DI-Element: GraphNode{[“Actor”], <Object >}

Nested elements: [HeaderCompartment | NameAndTypeCompartment], [GraphNode{“Activation”}], [GraphNode{“Destruction”}]

Name: **DiLink**

DI-Element: GraphEdge{<Link >}

Nested elements: [StereotypeCompartment], [Name], [DiStimulus]

Name: **DiStimulus**

DI-Element: GraphNode{<Stimulus>}

Nested elements: [SequenceExpression], [GraphNode {“TypeSeparator”}], [CallAction | ReturnAction | DestroyAction]

Name: **DiSequenceExpression**

DI-Element: GraphNode{“SequenceExpression”}

Nested elements: Name

Deployment diagram

Name: **DiNode**

DI-Element: GraphNode{<Node>}

Nested elements: [NameCompartment], [BodyCompartment]

Name: **DiNodeInstance**

DI-Element: GraphNode{<NodeInstance>}

Nested elements: [NameCompartment], [BodyCompartment]

Name: **DiComponent**

DI-Element: GraphNode{<Component>}

Nested elements: [NameCompartment], [BodyCompartment]

Name: **DiComponentInstance**

DI-Element: GraphNode{<ComponentInstance>}

Nested elements: [NameCompartment], [BodyCompartment]

C.3 Compartments

Name: **StereotypeCompartment**

DI-Element: GraphNode{“StereotypeCompartment”}

Nested elements: GraphNode{“StereotypeStart”}, ((DiStereotype |
GraphNode{“KeywordMetaclass”}),
GraphNode{“StereotypeSeparator”})*,
GraphNode{“StereotypeEnd”}

Name: **ExtensionPointCompartment**

DI-Element: GraphNode{“ExtensionPointCompartment”}

Nested elements: GraphNode{“Label”}, DiExtensionPoint*

Name: **InternalTransitionCompartment**

DI-Element: GraphNode{“InternalTransitionCompartment”}

Nested elements: CallAction*, Tansition*

Name: **RegionCompartment**

DI-Element: GraphNode{“RegionCompartment”}

Nested elements: [DiCompositeState | DiSimpleState]*

Name: **SubStateCompartment**

DI-Element: GraphNode{“SubStateCompartment”}

Nested elements: [DiCompositeState | DiSimpleState]*

Name: **HeaderCompartment**

DI-Element: GraphNode{“HeaderCompartment”}

Nested elements: NameAndTypeCompartment

Name: **NameCompartment**

DI-Element: GraphNode{“NameCompartment”}

Nested elements: [StereotypeCompartment], [Visibility], [NamespaceCompartment], Name, [PropertyCompartment]

Name: **NameAndTypeCompartment**

DI-Element: GraphNode{“NameCompartment”}

Nested elements: [StereotypeCompartment], [Visibility], [NamespaceCompartment], NameAndType,[PropertyCompartment]

Name: **NamespaceCompartment**

DI-Element: GraphNode{“NamespaceCompartment”}

Nested elements: GraphNode{"NamespaceStart"}, (Name{Namespace}, GraphNode{"NamespaceSeparator"})*, GraphNode{"NamespaceEnd"}

Name: **AttributeCompartment**

DI-Element: GraphNode{“AttributeCompartment”}

Nested elements: GraphNode{"DelimitedSection"}

Name: **OperationCompartment**

DI-Element: GraphNode{“OperationCompartment”}

Nested elements: GraphNode{"DelimitedSection"}

Name: **BodyCompartment**

DI-Element: GraphNode{“BodyCompartment”}

Nested elements: DiagramElement*

Name: **DelimitedSection**

DI-Element: GraphNode{“DelimitedSection”}

Nested elements: (DiAttribute | DiOperation | DiExtensionPoint)*

C.4 Sub-elements, nested in autonomous elements

Name: **DiStereotype**

DI-Element: GraphNode{<Stereotype>}

Nested elements: DiName | DiStereotypeImage

Name: **DiStereotypeImage**

DI-Element: GraphNode{"Image", <Stereotype>}

Nested elements: Image

Name: **DiAttribute**

DI-Element: GraphNode{<Attribute>}

Nested elements: [StereotypeCompartment], [Visibility], Name,
[GraphNode{“TypeSeparator”}, StructuralFeatureType],
[GraphNode{“MultiplicityStart”},
GraphNode{“Multiplicity”},
[GraphNode{“Ordering”}],
GraphNode{“MultiplicityEnd”}],
[GraphNode{“InitialValueSeparator”},
GraphNode{“InitialValue”}],
[PropertyCompartment]

Name: **DiOperation**

DI-Element: GraphNode{<Operation>}

Nested elements: [StereotypeCompartment], [Visibility], Name, DelimitedParameterList,
[GraphNode{“TypeSeparator”}, ParameterList], [PropertyCompartment]

Name: **DelimitedParameterList**

DI-Elements: GraphNode{“ParameterStart”}, (DiParameter, GraphNode{“ParameterSeparator”})*, GraphNode{“ParameterEnd”}

Name: **DiParameter**

DI-Element: GraphNode{<Parameter>}

Nested elements: GraphNode{“ParameterDirectionKind”}, Name, GraphNode{“TypeSeparator”}, DiDataType, GraphNode{“DefaultValueSeparator”}, GraphNode{“DefaultValue”}

Name: **ParameterList**

DI-Elements: (DiParameter, [GraphNode{“ParameterSeparator”}])*

Name: **DiExtensionPoint**

DI-Element: GraphNode{<ExtensionPoint>}

Nested elements: Name, GraphNode{“NameLocationSeparator”}, GraphNode{“Location”}

Name: **DiComment**

DI-Element: GraphNode{<Comment >} | GraphNode{<TaggedValue>}| GraphNode{<Constraint>}

C.5 Subelements

Name: **RoleAdornment**

DI-Element: GraphNode{“RoleAdornment”}

Nested elements: [Visibility], Name

Name: **DirectedName**

DI-Element: GraphNode{“DirectedName”}

Nested elements: [GraphNode{"Direction"}], Name

Name: **StructuralFeatureType**

DI-Element: GraphNode{"StructuralFeatureType”}

Nested elements: DiDataType

Name: **DiDataType**

DI-Element: GraphNode{<DataType>}

Nested elements: Name

Name: **InStates**

DI-Element: GraphNode{“InStates”}

Nested elements: GraphNode{"InStatesStart"}, (Name{SimpleState},
GraphNode{"InStatesSeparator"})*,
GraphNode{"InStatesEnd"}

Name: **DiCollaboration**

DI-Element: GraphNode{<Collaboration>}

Nested elements: NameCompartment

Name: **DiName**

DI-Element: GraphNode{“Name”}

Nested elements: TextElement

Name: **NameAndType**

DI-Element: GraphNode{“NameAndType”}

Nested elements: Name, GraphNode{“TypeSeparator”}, Name{Class}

Name: **Name{ModelElement}**

DI-Element: GraphNode{<ModelElement>}

Nested elements: Name

Name: **Name**

DI-Element: GraphNode{“Name”}

Annex D: Diagram Interchange XML Schema

(informative)

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xmi="http://www.omg.org/XMI" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:org.omg.uml2.diagram_interchange=
   "http://schema.omg.org/uml2/diagram_interchange.xml" targetNamespace=
   "http://schema.omg.org/uml2/diagram_interchange.xml">
 <xsd:import schemaLocation="XMI.xsd"
    namespace="http://www.omg.org/XMI"/>
 <xsd:complexType name="BezierPoint">
  <xsd:complexContent>
   <xsd:extension
      base="org.omg.uml2.diagram_interchange:Point">
     <xsd:choice minOccurs="0" maxOccurs="unbounded">
      <xsd:element name="controls"
         type="org.omg.uml2.diagram_interchange:Point"/>
     </xsd:choice>
     <xsd:attribute name="controls" type="xsd:string"/>
   </xsd:extension>
  </xsd:complexContent>
 </xsd:complexType>
 <xsd:element name="BezierPoint" type="org.omg.uml2.diagram_interchange:BezierPoint"/>
 <xsd:complexType name="CoreSemanticModelBridge">
  <xsd:complexContent>
   <xsd:extension base=
      "org.omg.uml2.diagram_interchange:SemanticModelBridge">
     <xsd:attribute name="element" type="xsd:string"/>
   </xsd:extension>
  </xsd:complexContent>
 </xsd:complexType>
 <xsd:element name="CoreSemanticModelBridge" type=
    "org.omg.uml2.diagram_interchange:CoreSemanticModelBridge"/>
 <xsd:complexType name="Diagram">
  <xsd:complexContent>
   <xsd:extension
      base="org.omg.uml2.diagram_interchange:GraphNode">
     <xsd:choice minOccurs="0" maxOccurs="unbounded">
      <xsd:element name="viewport"
         type="org.omg.uml2.diagram_interchange:Point"/>
      <xsd:element name="diagramLink"
         type="org.omg.uml2.diagram_interchange:DiagramLink"/>
      <xsd:element name="owner" type=
```

```
        “org.omg.uml2.diagram_interchange:SemanticModelBridge”/>
    </xsd:choice>
    <xsd:attribute name=”name” type=”xsd:string”/>
    <xsd:attribute name=”zoom” type=”xsd:double”/>
    <xsd:attribute name=”viewport” type=”xsd:string”/>
    <xsd:attribute name=”diagramLink” type=”xsd:string”/>
   </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
<xsd:element name=”Diagram”
   type=”org.omg.uml2.diagram_interchange:Diagram”/>
<xsd:complexType name=”DiagramElement”>
  <xsd:choice minOccurs=”0” maxOccurs=”unbounded”>
   <xsd:element name=”property”
      type=”org.omg.uml2.diagram_interchange:Property”/>
   <xsd:element name=”reference”
      type=”org.omg.uml2.diagram_interchange:Reference”/>
   <xsd:element ref=”xmi:Extension”/>
  </xsd:choice>
  <xsd:attribute ref=”xmi:id”/>
  <xsd:attributeGroup ref=”xmi:ObjectAttribs”/>
  <xsd:attribute name=”isVisible” type=”xsd:boolean”/>
  <xsd:attribute name=”reference” type=”xsd:string”/>
</xsd:complexType>
<xsd:element name=”DiagramElement”
   type=”org.omg.uml2.diagram_interchange:DiagramElement”/>
<xsd:complexType name=”DiagramLink”>
  <xsd:choice minOccurs=”0” maxOccurs=”unbounded”>
   <xsd:element name=”viewport”
      type=”org.omg.uml2.diagram_interchange:Point”/>
   <xsd:element name=”diagram”
      type=”org.omg.uml2.diagram_interchange:Diagram”/>
   <xsd:element ref=”xmi:Extension”/>
  </xsd:choice>
  <xsd:attribute ref=”xmi:id”/>
  <xsd:attributeGroup ref=”xmi:ObjectAttribs”/>
  <xsd:attribute name=”zoom” type=”xsd:double”/>
  <xsd:attribute name=”viewport” type=”xsd:string”/>
  <xsd:attribute name=”diagram” type=”xsd:string”/>
</xsd:complexType>
<xsd:element name=”DiagramLink”
   type=”org.omg.uml2.diagram_interchange:DiagramLink”/>
<xsd:complexType name=”Dimension”>
  <xsd:choice minOccurs=”0” maxOccurs=”unbounded”>
   <xsd:element ref=”xmi:Extension”/>
  </xsd:choice>
  <xsd:attribute ref=”xmi:id”/>
  <xsd:attributeGroup ref=”xmi:ObjectAttribs”/>
  <xsd:attribute name=”width” type=”xsd:double”/>
```

```
  <xsd:attribute name="height" type="xsd:double"/>
</xsd:complexType>
<xsd:element name="Dimension"
   type="org.omg.uml2.diagram_interchange:Dimension"/>
<xsd:complexType name="Ellipse">
  <xsd:complexContent>
    <xsd:extension
       base="org.omg.uml2.diagram_interchange:GraphicPrimitive">
      <xsd:choice minOccurs="0" maxOccurs="unbounded">
        <xsd:element name="center"
           type="org.omg.uml2.diagram_interchange:Point"/>
      </xsd:choice>
      <xsd:attribute name="radiusX" type="xsd:double"/>
      <xsd:attribute name="radiusY" type="xsd:double"/>
      <xsd:attribute name="rotation" type="xsd:double"/>
      <xsd:attribute name="startAngle" type="xsd:double"/>
      <xsd:attribute name="endAngle" type="xsd:double"/>
      <xsd:attribute name="center" type="xsd:string"/>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
<xsd:element name="Ellipse"
   type="org.omg.uml2.diagram_interchange:Ellipse"/>
<xsd:complexType name="GraphConnector">
  <xsd:choice minOccurs="0" maxOccurs="unbounded">
    <xsd:element name="position"
       type="org.omg.uml2.diagram_interchange:Point"/>
    <xsd:element name="graphEdge"
       type="org.omg.uml2.diagram_interchange:GraphEdge"/>
    <xsd:element ref="xmi:Extension"/>
  </xsd:choice>
  <xsd:attribute ref="xmi:id"/>
  <xsd:attributeGroup ref="xmi:ObjectAttribs"/>
  <xsd:attribute name="position" type="xsd:string"/>
  <xsd:attribute name="graphEdge" type="xsd:string"/>
</xsd:complexType>
<xsd:element name="GraphConnector"
   type="org.omg.uml2.diagram_interchange:GraphConnector"/>
<xsd:complexType name="GraphEdge">
  <xsd:complexContent>
    <xsd:extension
       base="org.omg.uml2.diagram_interchange:GraphElement">
      <xsd:choice minOccurs="0" maxOccurs="unbounded">
        <xsd:element name="waypoints"
           type="org.omg.uml2.diagram_interchange:Point"/>
        <xsd:element name="anchor"
           type="org.omg.uml2.diagram_interchange:GraphConnector"/>
      </xsd:choice>
      <xsd:attribute name="waypoints" type="xsd:string"/>
```

```
    <xsd:attribute name="anchor" type="xsd:string"/>
   </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
<xsd:element name="GraphEdge"
   type="org.omg.uml2.diagram_interchange:GraphEdge"/>
<xsd:complexType name="GraphElement">
  <xsd:complexContent>
   <xsd:extension
      base="org.omg.uml2.diagram_interchange:DiagramElement">
    <xsd:choice minOccurs="0" maxOccurs="unbounded">
      <xsd:element name="position"
         type="org.omg.uml2.diagram_interchange:Point"/>
      <xsd:element name="anchorage"
         type="org.omg.uml2.diagram_interchange:GraphConnector"/>
      <xsd:element name="contained"
         type="org.omg.uml2.diagram_interchange:DiagramElement"/>
      <xsd:element name="semanticModel" type=
    "org.omg.uml2.diagram_interchange:SimpleSemanticModelElement"/>
      <xsd:element name="link"
         type="org.omg.uml2.diagram_interchange:DiagramLink"/>
    </xsd:choice>
    <xsd:attribute name="position" type="xsd:string"/>
   </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
<xsd:element name="GraphElement"
   type="org.omg.uml2.diagram_interchange:GraphElement"/>
<xsd:complexType name="GraphNode">
  <xsd:complexContent>
   <xsd:extension
      base="org.omg.uml2.diagram_interchange:GraphElement">
    <xsd:choice minOccurs="0" maxOccurs="unbounded">
      <xsd:element name="size"
         type="org.omg.uml2.diagram_interchange:Dimension"/>
    </xsd:choice>
    <xsd:attribute name="size" type="xsd:string"/>
   </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
<xsd:element name="GraphNode"
   type="org.omg.uml2.diagram_interchange:GraphNode"/>
<xsd:complexType name="GraphicPrimitive">
  <xsd:complexContent>
   <xsd:extension
      base="org.omg.uml2.diagram_interchange:LeafElement"/>
  </xsd:complexContent>
</xsd:complexType>
<xsd:element name="GraphicPrimitive"
```

```
    type="org.omg.uml2.diagram_interchange:GraphicPrimitive"/>
<xsd:complexType name="Image">
  <xsd:complexContent>
    <xsd:extension
      base="org.omg.uml2.diagram_interchange:LeafElement">
      <xsd:attribute name="url" type="xsd:string"/>
      <xsd:attribute name="mimeType" type="xsd:string"/>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
<xsd:element name="Image" type="org.omg.uml2.diagram_interchange:Image"/>
<xsd:complexType name="LeafElement">
  <xsd:complexContent>
    <xsd:extension
      base="org.omg.uml2.diagram_interchange:DiagramElement"/>
  </xsd:complexContent>
</xsd:complexType>
<xsd:element name="LeafElement"
    type="org.omg.uml2.diagram_interchange:LeafElement"/>
<xsd:complexType name="Point">
  <xsd:choice minOccurs="0" maxOccurs="unbounded">
    <xsd:element ref="xmi:Extension"/>
  </xsd:choice>
  <xsd:attribute ref="xmi:id"/>
  <xsd:attributeGroup ref="xmi:ObjectAttribs"/>
  <xsd:attribute name="x" type="xsd:double"/>
  <xsd:attribute name="y" type="xsd:double"/>
</xsd:complexType>
<xsd:element name="Point"
    type="org.omg.uml2.diagram_interchange:Point"/>
<xsd:complexType name="Polyline">
  <xsd:complexContent>
    <xsd:extension
      base="org.omg.uml2.diagram_interchange:GraphicPrimitive">
      <xsd:choice minOccurs="0" maxOccurs="unbounded">
        <xsd:element name="waypoints"
          type="org.omg.uml2.diagram_interchange:Point"/>
      </xsd:choice>
      <xsd:attribute name="closed" type="xsd:boolean"/>
      <xsd:attribute name="waypoints" type="xsd:string"/>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
<xsd:element name="Polyline"
    type="org.omg.uml2.diagram_interchange:Polyline"/>
<xsd:complexType name="Property">
  <xsd:choice minOccurs="0" maxOccurs="unbounded">
    <xsd:element ref="xmi:Extension"/>
  </xsd:choice>
```

```
  <xsd:attribute ref=”xmi:id”/>
  <xsd:attributeGroup ref=”xmi:ObjectAttribs”/>
  <xsd:attribute name=”key” type=”xsd:string”/>
  <xsd:attribute name=”value” type=”xsd:string”/>
</xsd:complexType>
<xsd:element name=”Property”
   type=”org.omg.uml2.diagram_interchange:Property”/>
<xsd:complexType name=”Reference”>
  <xsd:complexContent>
    <xsd:extension
      base=”org.omg.uml2.diagram_interchange:DiagramElement”>
     <xsd:choice minOccurs=”0” maxOccurs=”unbounded”>
       <xsd:element name=”referenced”
         type=”org.omg.uml2.diagram_interchange:DiagramElement”/>
     </xsd:choice>
     <xsd:attribute name=”isIndividualRepresentation”
       type=”xsd:boolean”/>
     <xsd:attribute name=”referenced” type=”xsd:string”/>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
<xsd:element name=”Reference”
   type=”org.omg.uml2.diagram_interchange:Reference”/>
<xsd:complexType name=”SemanticModelBridge”>
  <xsd:choice minOccurs=”0” maxOccurs=”unbounded”>
    <xsd:element ref=”xmi:Extension”/>
  </xsd:choice>
  <xsd:attribute ref=”xmi:id”/>
  <xsd:attributeGroup ref=”xmi:ObjectAttribs”/>
  <xsd:attribute name=”presentation” type=”xsd:string”/>
</xsd:complexType>
<xsd:element name=”SemanticModelBridge”
   type=”org.omg.uml2.diagram_interchange:SemanticModelBridge”/>
<xsd:complexType name=”SimpleSemanticModelElement”>
  <xsd:complexContent>
    <xsd:extension
      base=”org.omg.uml2.diagram_interchange:SemanticModelBridge”>
     <xsd:attribute name=”typeInfo” type=”xsd:string”/>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
<xsd:element name=”SimpleSemanticModelElement” type=
   “org.omg.uml2.diagram_interchange:SimpleSemanticModelElement”/>
<xsd:complexType name=”TextElement”>
  <xsd:complexContent>
    <xsd:extension
      base=”org.omg.uml2.diagram_interchange:LeafElement”>
     <xsd:attribute name=”text” type=”xsd:string”/>
    </xsd:extension>
```

```
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:element name=”TextElement”
     type=”org.omg.uml2.diagram_interchange:TextElement”/>
</xsd:schema>
```

Index