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.

PATENT

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.

NOTICE

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

WHILE THE INFORMATION IN THIS PUBLICATION IS BELIEVED TO BE ACCURATE, THE OBJECT MANAGEMENT GROUP AND THE COMPANIES LISTED ABOVE MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL INCLUDING, BUT NOT LIMITED TO ANY WARRANTY OF TITLE OR OWNERSHIP, IMPLIED WARRANTY OF MERCHANTABILITY OR WARRANTY OF FITNESS FOR 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 indirect, incidental, special, consequential, reliance or cover damages, including loss of profits, revenue, data or use, incurred by any user or any third party. 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. 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.

RESTRICTED RIGHTS LEGEND. Use, duplication, or disclosure by government is subject to restrictions as set forth in subdivision (c) (1) (ii) of the Right in Technical Data and Computer Software Clause at DFARS 252.227.7013 OMG® and Object Management are registered trademarks of the Object Management Group, Inc. OMG OBJECT MANAGEMENT
GROUP, CORBA, CORBA ACADEMY, CORBA ACADEMY & DESIGN, THE INFORMATION BROKERAGE, OBJECT REQUEST BROKER, OMG IDL, CORBAFACILITIES, CORBASERVICES, CORBANET, CORBAMED, CORBADOMAINS, GIOP, IIOP, OMA, CORBA THE GEEK, UNIFIED MODELING LANGUAGE, UML, and UML CUBE LOGO are registered trademarks or trademarks of the Object Management Group, Inc.

X/Open is a trademark of X/Open Company Ltd.

ISSUE REPORTING

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 & Specifications, Report a Bug/Issue.
Contents

Preface ................................................................. v

1. XMI Overview ...................................................... 1-1
   1.1 Introduction ................................................. 1-1
   1.2 XMI and the MOF ............................................. 1-2
       1.2.1 An Overview of the MOF ......................... 1-2
       1.2.2 The Relationship between XMI and MOF .... 1-6
       1.2.3 The Relationship between XMI, MOF and UML 1-6
   1.3 XMI and XML ................................................ 1-7
       1.3.1 The Roots of XML ................................. 1-7
       1.3.2 Benefits of using XML ......................... 1-7
       1.3.3 XML and the Computer Industry .............. 1-8
       1.3.4 How XML Works ................................. 1-8
       1.3.5 XML and the OMG ............................... 1-11
       1.3.6 New XML Technologies ....................... 1-11

2. Usage Scenarios .................................................. 2-1
   2.1 Purpose ...................................................... 2-1
   2.2 Combining Tools in a Heterogeneous Environment .... 2-1
   2.3 Co-operating with Common Metamodel Definitions ... 2-2
   2.4 Work Environment ......................................... 2-3
       2.4.1 Working in a Distributed and Intermittently Connected Environment ................. 2-3
   2.5 Promoting Design Patterns and Reuse ................. 2-3

3. XMI DTD Design Principles ...................................... 3-1
   3.1 Purpose ...................................................... 3-1
Contents

3.2 Use of XML DTDs ........................................ 3-2
  3.2.1 XML Validation of XMI Documents ............ 3-3
  3.2.2 Requirements for XMI DTDs ................. 3-3

3.3 Basic Principles ...................................... 3-3
  3.3.1 Required XML Declarations .................. 3-3
  3.3.2 Metamodel Class Representation ........... 3-4
  3.3.3 Metamodel Extension Mechanism .......... 3-4

3.4 XMI DTD and Document Structure .................. 3-5

3.5 Necessary XMI DTD Declarations ................... 3-6
  3.5.1 Necessary XMI Attributes .................... 3-6
  3.5.2 Common XMI Elements ......................... 3-8
  3.5.3 XMI ............................................... 3-9
  3.5.4 XMI.header .................................... 3-10
  3.5.5 XMI.content .................................. 3-10
  3.5.6 XMI.extensions ............................... 3-10
  3.5.7 XMI.extension .................................. 3-10
  3.5.8 XMI.documentation ............................ 3-11
  3.5.9 XMI.model ..................................... 3-11
  3.5.10 XMI.metamodel ................................. 3-12
  3.5.11 XMI.metametamodel ......................... 3-12
  3.5.12 XMI.import .................................... 3-12
  3.5.13 XMI.difference ............................... 3-13
  3.5.14 XMI.delete .................................... 3-13
  3.5.15 XMI.add ........................................ 3-13
  3.5.16 XMI.replace .................................. 3-14
  3.5.17 XMI.reference .................................. 3-14

3.6 Metamodel Class Specification ..................... 3-14
  3.6.1 Namespace Qualified XML Element Names .... 3-14
  3.6.2 Metamodel Multiplicities .................... 3-16
  3.6.3 Class Specification ........................... 3-16
  3.6.4 Inheritance Specification .................... 3-16
  3.6.5 Attribute Specification ....................... 3-17
  3.6.6 Association Specification ..................... 3-19
  3.6.7 Containment Specification ..................... 3-19

3.7 Transmitting Incomplete Metadata .................. 3-20
  3.7.1 Interchange of Model Fragments ............ 3-20
  3.7.2 XMI Encoding .................................... 3-20
  3.7.3 Example .......................................... 3-20

3.8 Linking ................................................. 3-21
  3.8.1 Design Principles .............................. 3-21
## Contents

3.8.2 Linking ............................................. 3-21
3.8.3 Example from UML ................................. 3-22
3.8.4 XML.reference ................................... 3-24

3.9 Transmitting Metadata Differences ................. 3-24
   3.9.1 Definitions ................................... 3-24
   3.9.2 Differences ................................... 3-25
   3.9.3 XMI Encoding .................................. 3-25
   3.9.4 Example ....................................... 3-26

3.10 Document Exchange with Multiple Tools .......... 3-27
   3.10.1 Definitions ................................... 3-27
   3.10.2 Procedures ................................... 3-28
   3.10.3 Example ....................................... 3-29

3.11 General Datatype Mechanism ....................... 3-30

4. XML DTD Production ................................. 4-1
   4.1 Purpose .......................................... 4-1
   4.2 DTD Syntax Approaches ......................... 4-2
      4.2.1 DTD without entities ..................... 4-2
      4.2.2 DTD with class-level entities .......... 4-3
      4.2.3 DTD with multi-level entities .......... 4-3
   4.3 Rule Set 1: Simple DTD ......................... 4-4
      4.3.1 EBNF ...................................... 4-4
   4.4 Fixed DTD Elements ............................. 4-13

5. XML Generation Principles .......................... 5-1
   5.1 Purpose .......................................... 5-1
   5.2 Introduction .................................... 5-1
   5.3 Two Model Sources ............................... 5-2
      5.3.1 Production by Object Containment ....... 5-2
      5.3.2 MOF’s Role in XML Production .......... 5-8
      5.3.3 Production by Package Extent .......... 5-8
   5.4 Distinctions between Approaches ................ 5-12
      5.4.1 Distinctions between Approaches in Certain Situations .......... 5-12
      5.4.2 External Links ............................. 5-13
      5.4.3 Links not Represented by References ..... 5-13
      5.4.4 Classifier-level Attributes ............. 5-13
      5.4.5 Standard Elements ......................... 5-14

6. XML Document Production ............................ 6-1
   6.1 Introduction ..................................... 6-1
Contents

6.2 ENBF Rules Representation ........................................... 6-1
Appendix A - UML DTD ................................................... A-1
Appendix B - MOF DTD .................................................... B-1
Appendix C - Example Model Encodings ............................... C-1
Appendix D - References .................................................. D-1
Appendix E - Compatibility and Conformance ....................... E-1
Index .............................................................................. 1
Preface

About the XML Metadata Interchange (XMI) Specification

Note – Text revised by the XMI 1.2 RTF is identified with change bars. In some cases, the RTF removed entire sections of text from this document to resolve issues; the places where this has happened are marked with change bars and the statement `<text deleted by RTF>` in the left margin. For details about the issues and associated revisions, see the RTF Final Report (ptc/01-08-33) or the convenience document for this version of the specification (ptc/01-08-27).

The main purpose of XMI is to enable easy interchange of metadata between modeling tools (based on the OMG-UML) and metadata repositories (OMG-MOF based) in distributed heterogeneous environments. XMI integrates three key industry standards:

1. XML - eXtensible Markup Language, a W3C standard
2. UML - Unified Modeling Language, an OMG modeling standard
3. MOF - Meta Object Facility, an OMG metamodeling and metadata repository standard

The integration of these three standards into XMI marries the best of OMG and W3C metadata and modeling technologies, allowing developers of distributed systems to share object models and other metadata over the Internet.

XMI, together with MOF and UML form the core of the OMG metadata repository architecture as illustrated in Figure 1.
SMIF (XMI) and OMG Repository Architecture and the XM

Figure 1  The OMG Repository Architecture and the SMIF

The UML standard defines a rich, object oriented modeling language that is supported by a range of graphical design tools. The MOF standard defines an extensible framework for defining models for metadata, and providing tools with programmatic interfaces to store and access metadata in a repository. XMI allows metadata to be interchanged as streams or files with a standard format based on XML. The complete architecture offers a wide range of implementation choices to developers of tools, repositories, and object frameworks. XMI in particular lowers the barrier to entry for the use of OMG metadata standards.

Key aspects of the architecture include:

• A four layered metamodeling architecture for general purpose manipulation of metadata in distributed object repositories. See the MOF and UML specifications for more details.

• The use of MOF to define and manipulate metamodels programmatically using fine grained CORBA interfaces. This approach leverages the strength of CORBA distributed object infrastructure.

• The use of UML notation for representing models and metamodels.

• The use of standard information models (UML) to describe the semantics of object analysis and design models.

• The use of Stream-based Model Interchange Format (SMIF) - the current XMI proposal - for stream based interchange of metadata.
The OMG ADTF and other task forces have already begun extending this architecture to include data warehouse metadata (Common Warehouse Metadata Interchange RFP) and other metadata by defining MOF compliant metamodels.

This specification mainly consists of:

- A set of XML Document Type Definition (DTD) production rules for transforming MOF based metamodels into XML DTDs.
- A set of XML Document production rules for encoding and decoding MOF based metadata.
- Design principles for XMI based DTDs and XML Streams.
- Concrete DTDs for UML and MOF.

This specification defines these standards and provides proof of concept that covers key aspects of the XMI. The specification represents the integration of work currently underway by the co-submitters and supporters in the areas of object repositories, object modeling tools, web authoring technology, and business object management in distributed object environments. The co-submitters intend to commercialize the XMI technology within the guidelines of the OMG.

The XMI specification enhances metadata management and metadata interoperability in distributed object environments in general and in distributed development environments in particular. While this specification addresses stream based metadata interoperability in the object analysis and design domain, XMI (in part because it is MOF based) is equally applicable to metadata in many other domains. Examples include metamodels that cover the application development life cycle as well as additional domains such as data warehouse management, distributed objects, and business object management.

The adoption of the UML and MOF specifications in 1997 was a key step forward for the OMG and the industry in terms of achieving consensus on modeling technology and repositories after years of failed attempts to unify both areas. The adoption of XMI is expected to reduce the plethora of proprietary metadata interchange formats and minimally successful attempts of the Meta Data Coalition (Meta Data Interchange Specification) and Case Data Interchange Format (EIA CDIF) because of widespread adoption of W3C (XML) and OMG (UML, MOF) standards. XMI is also expected to ease the integration of CORBA, XML, Java, and COM based development environments, which are evolving towards similar extensible repository architectures based on standard information models, repository interfaces, and interchange formats.

About the Object Management Group

The Object Management Group, Inc. (OMG) is an international organization supported by over 800 members, including information system vendors, software developers and users. Founded in 1989, the OMG promotes the theory and practice of object-oriented technology in software development. The organization’s charter includes the establishment of industry guidelines and object management specifications to provide a common framework for application development. Primary goals are the reusability, portability, and interoperability of object-based software in distributed, heterogeneous environments.
environments. Conformance to these specifications will make it possible to develop a heterogeneous applications environment across all major hardware platforms and operating systems.

OMG’s objectives are to foster the growth of object technology and influence its direction by establishing the Object Management Architecture (OMA). The OMA provides the conceptual infrastructure upon which all OMG specifications are based.

**Intended Audience and Use**

The information described in this manual is aimed at managers and software designers who want to produce applications that comply with the family of OMG standards. The benefit of compliance is, in general, to be able to produce interoperable applications that run in heterogeneous, distributed environments.

**Context of OMG Modeling**

The OMG is dedicated to producing a framework and specifications for commercially available object-oriented environments. The Object Management Architecture (as defined in the *Object Management Architecture Guide*) is the umbrella architecture for OMG specifications. The defining model for the architecture is the Reference Model, which classifies the components, interfaces, and protocols that compose an object system. The Reference Model consists of the following components:

- **Object Request Broker**, which enables objects to transparently make and receive requests and responses in a distributed environment. It is the foundation for building applications from distributed objects and for interoperability between applications in hetero- and homogeneous environments. The architecture and specifications of the Object Request Broker are described in *CORBA: Common Object Request Broker Architecture and Specification*.

- **Object Services**, a collection of services (interfaces and objects) that support basic functions for using and implementing objects. Services are necessary to construct any distributed application and are always independent of application domains. For example, the Life Cycle Service defines conventions for creating, deleting, copying, and moving objects; it does not dictate how the objects are implemented in an application. Specifications for Object Services are contained in *CORBA:services: Common Object Services Specification*.

- **Common Facilities**, a collection of services that many applications may share, but which are not as fundamental as the Object Services. For instance, a system management or electronic mail facility could be classified as a common facility.

- **Application Objects**, which are objects specific to particular commercial products or end user systems. Application Objects correspond to the traditional notion of applications, so they are not standardized by the OMG. Instead, Application Objects constitute the uppermost layer of the Reference Model.
• **OMG Modeling**, a collection of modeling specifications that advance the state of the industry by enabling OO visual modeling tool interoperability. OMG Modeling provides a set of CORBA interfaces that can be used to define and manipulate a set of interoperable metamodels.

OMG formal documents are available from our web site in PostScript and PDF format. To obtain print-on-demand books in the documentation set or other OMG publications, contact the Object Management Group, Inc., at:

OMG Headquarters
250 First Avenue, Suite 201
Needham, MA 02494
USA
Tel: +1-781-444-0404
Fax: +1-781-444-0320
pubs@omg.org
http://www.omg.org

**Structure of this Manual**

In addition to this Preface, the *XMI Specification* contains the following chapters:

Chapter 1 - XMI Overview

Includes a descriptive summary of XMI and MOF, XMI and XML, and XML and the OMG.

Chapter 2 - Usage Scenarios

Describes how the XMI is expected to be used by customers and tool vendors.

Chapter 3 - XMI DTD Design Principles

Provides a discussion of Document Type Definition (DTD) usage, generation and standard parts.

Chapter 4 - XML DTD Production

Specifies the production rules for DTDs, as part of the encoding of MOF based metamodels into the proposed format.

Chapter 5 - XML Generation Principles

Discusses the manner in which a model is represented as an XML document.

Chapter 6 - XML Document Production

Specifies the production rules for encoding any model, with a MOF-defined metamodel, in the proposed format.

Appendix A - UML DTD

Appendix B - MOF DTD

Appendix C - Example Model Encodings
Appendix D - References
Appendix E - Compatibility and Conformance

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.

Helvetica bold - OMG Interface Definition Language (OMG IDL) and syntax elements.

Courier bold - Programming language elements.

Helvetica - Exceptions

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

Acknowledgments

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

- Ardent
- Aviatis
- Boeing
- Cayenne Software
- Cooperative Research Centre for Distributed Systems Technology (DSTC)
- Daimler-Benz
- Fujitsu
- Genesis Development
- ICONIX
- Inline Software
- Integrated Systems
- International Business Machines Corporation
- MCI Systemhouse
- NCR
- Nihon Unisys
- NTT
- Oracle Corporation
- Platinum Technology, Inc.
• Rational Software Corporation
• Recerca Informatica
• Select Software
• Softeam
• Sprint Communications Company
• Sybase, Inc.
• Telefonica I+D
• Unisys Corporation
• Universitat Politecnica de Catalunya
• Verilog
• Xerox
XMI Overview

Contents

This chapter includes the following topics.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Introduction”</td>
<td>1-1</td>
</tr>
<tr>
<td>“XMI and the MOF”</td>
<td>1-2</td>
</tr>
<tr>
<td>“XMI and XML”</td>
<td>1-7</td>
</tr>
</tbody>
</table>

1.1 Introduction

This specification is based on the W3C’s Extensible Markup Language (XML). The XML-based Metadata Interchange (XMI) specification has two major components:

1. The XML DTD Production Rules for producing XML Document Type Definitions (DTDs) for XMI encoded metadata. XMI DTDs serve as syntax specifications for XMI documents, and allow generic XML tools to be used to compose and validate XMI documents.

2. The XML Document Production Rules for encoding metadata into an XML compatible format. The production rules can be applied in reverse to decode XMI documents and reconstruct the metadata.

The XMI specification supports the interchange of any kind of metadata that can be expressed using the MOF specification, including both model and metamodel information. The specification supports the encoding of metadata consisting of both complete models and model fragments, as well as tool-specific extension metadata. XMI has optional support for interchange of metadata in differential form, and for metadata interchange with tools that have incomplete understanding of the metadata.
XML is gaining widespread acceptance as the de facto standard for representing structured information in the context of the world-wide web and beyond. Basing the proposed OMG SMIF on XML means that XMI can be used for metadata interchange with and between non-CORBA based metadata repositories and tools.

The XML language is defined by the W3C’s “Extensible Markup Language (XML) Recommendation 1.0” document [REC-xml-19980210]. This definition includes a specification of XML in Extended Backus-Naur Form (EBNF) notation. XML is LL(1) parsable.

Note – Equivalent models are those that have the same data structures and relationships. Equivalent XML documents are documents equivalent models when the Stream Consumption rules are applied to the two documents.

1.2 XMI and the MOF

XMI is an interchange format for metadata that is defined in terms of the Meta Object Facility (MOF) standard. This section provides an overview of the MOF and gives a rationale for basing XMI on the MOF rather than some other modeling technology.

1.2.1 An Overview of the MOF

The MOF is the OMG’s adopted technology for defining metadata and representing it as CORBA objects. In this specification, metadata is a general term for data that in some sense describes information. The information so described may be information represented in a computer system (e.g., in the form of files, databases, running program instances and so on). Alternatively, the information may be embodied in some system, with the metadata being a description of some aspect of the system such as a part of its design.

The MOF supports any kind of metadata that can be described using Object Modeling techniques. This metadata may describe any aspect of a system and the information it contains, and may describe it to any level of detail and rigor depending on the metadata requirements.

The designers envisaged that the MOF-based metadata will be used in a wide range of CORBA related applications. For example:

- Metadata repositories and tools will support the process of analysis, design, and development of CORBA-based software.
- Metadata repositories will support infrastructure services such as COS Trading, COS Events, and ultimately the CORBA Interface Repository itself.
- Metadata repositories will support data warehousing, data mining, and database interoperability.
- Metadata will be used to describe free-text data sources such as on-line document collections and the world-wide web.
The term **model** is generally usually used to denote a description of something, typically something in the real world. The concept of a model is highly fluid, and depends on one’s point of view. To someone who is concerned with building or understanding an entire system, a model would include all of the metadata for the system. On the other hand, most people are only concerned with certain components (e.g., programs A and B) or certain kinds of detail (e.g., wiring diagrams) of the system.

In the MOF context, the term **model** has a broader meaning. Here, a model is any collection of metadata that is related in the following ways:

- The metadata describes information that is itself related in some way.
- The metadata all conforms to rules governing its structure and consistency; that is, it has a common **abstract syntax**.
- The metadata has meaning in a common (often implied) semantic framework.

**Note** – An MOF model is not necessarily a model in the usual sense of the word. It does not necessarily describe something in the real world, and it does not necessarily describe things in a way that is interesting to modelers.

Metadata is itself a kind of information, and can accordingly be described by other metadata. In MOF terminology, metadata that describes metadata is called **meta-metadata**, and a model that consists of a meta-metadata is called a **metamodel**.

One kind of metamodel plays a central role in the MOF. An **MOF metamodel** defines the abstract syntax of the metadata in the MOF representation of a model. Since there are many possible kinds of metadata in a typical system, the MOF framework needs to support many different MOF metamodels. The MOF integrates these metamodels by defining a common abstract syntax for defining metamodels. This abstract syntax is called the **MOF Model** and is model for metamodels (i.e., a meta-metamodel). The MOF metadata framework is typically depicted as a four layer architecture as shown in Table 1-1.

<table>
<thead>
<tr>
<th>Meta-level</th>
<th>MOF terms</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>M3</td>
<td>meta-metamodel</td>
<td>The “MOF Model”</td>
</tr>
<tr>
<td>M2</td>
<td>meta-metadata</td>
<td>UML Metamodel,</td>
</tr>
<tr>
<td></td>
<td>metamodel</td>
<td>CWMI Metamodel(s), etc.</td>
</tr>
<tr>
<td>M1</td>
<td>metadata</td>
<td>UML Models,</td>
</tr>
<tr>
<td></td>
<td>model</td>
<td>Warehouse Schemas, etc.</td>
</tr>
<tr>
<td>M0</td>
<td>data</td>
<td>Modeled systems,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Warehouse databases, etc.</td>
</tr>
</tbody>
</table>

A couple points on the OMG / MOF metadata terminology:
• To make things easier to understand, we often describe things in terms of their level in the meta-stack (e.g., the MOF Model is an M3-level model in a 4 level stack).

• The “meta-” prefix should be viewed in a relative rather than absolute sense. Similarly, the numbering of meta-levels is not absolute.

• While there are typically 4 layers in an MOF based metadata stack, the number of layers can be more or less than this.

The MOF specification has three core parts (i.e., the specification of MOF Model, the MOF IDL Mapping, and the MOF’s interfaces).

1.2.1.1 The MOF Model

The “MOF Model” is the MOF’s built-in meta-metamodel. One can think of it as the “abstract language” for defining MOF metamodels. This is analogous to the way that the UML metamodel is an abstract language for defining UML models. While the MOF and UML are designed for two different kinds of modeling (i.e., metadata versus object modeling), the MOF Model and the core of the UML metamodel are closely aligned in their modeling concepts. (The alignment of the two models is close enough to allow UML notation to be used to express MOF-based metamodels!)

The three main metadata modeling constructs provided by the MOF are the Class, Association, and Package. These are similar to their counterparts in UML, with some simplifications:

• Classes can have Attributes and Operations at both “object” and “class” level. Attributes have the obvious usage (i.e., representation of metadata). Operations are provided to support metamodel specific functions on the metadata. Both Attributes and Operation Parameters may be defined as “ordered,” or as having structural constraints on their cardinality and uniqueness. Classes may multiply inherit from other Classes.

• Associations support binary links between Class “instances.” Each Association has two AssociationEnds that may specify “ordering” or “aggregation” semantics, and structural constraints on cardinality or uniqueness. When a Class is the type of an AssociationEnd, the Class may contain a Reference that allows navigability of the Association’s links from a Class “instance.”

• Packages are collections of related Classes and Associations. Packages can be composed by importing other Packages or by inheriting from them. Packages can also be nested, though this provides a form of information hiding rather than reuse.

The other significant MOF Model constructs are DataTypes and Constraints. DataTypes allow the use of non-object types for Parameters or Attributes. In the OMG MOF specification, these must be data types or interface types expressible in CORBA IDL.

Constraints are used to associate semantic restrictions with other elements in an MOF metamodel. This defines the well-formedness rules for the metadata described by a metamodel. Any language may be used to express Constraints, though there are obvious advantages in using a formal language like OCL.
1.2.1.2 The MOF IDL Mapping

The MOF’s “IDL Mapping” is a standard set of templates that map an MOF metamodel onto a corresponding set of CORBA IDL interfaces. If the input to the mapping is the metamodel for a given kind of metadata, then the resulting IDL interfaces are for CORBA objects that can represent that metadata. The mapped IDL are typically used in a repository for storing the metadata.

The IDL mapping is too large to describe here, and indeed it is largely irrelevant to the problem of model interchange. Instead, we will simply note the main correspondences between elements in an MOF metamodel (M2-level entities) and the CORBA objects that represent metadata (M1-level entities).

- A Class in the metamodel maps onto an IDL interface for metadata objects and a metadata class proxy. These interfaces support the Operations, Attributes, and References defined in the metamodel, and in the case of class proxy, provide a factory operation for metadata objects.
- An Association maps onto an interface for a metadata association proxy that supports association queries and updates.
- A Package maps onto an interface for a metadata package proxy. A package proxy acts as a holder for the proxies for the Classes and Associations contained by the Package, and therefore serves to define a logical extent for metadata associations, classifier level attributes, and the like.

The IDL that is produced by the mapping is defined in precise detail so that different vendor implementations of the MOF can generate compatible repository interfaces from a given MOF metamodel. Similarly, the semantic specification of the mapped interfaces allows metadata objects to be interoperable.

In addition to the metamodel specific interfaces for the metadata (defined by the IDL mapping), MOF metadata objects share a common set of Reflective base interfaces. These interfaces allow a ‘generic’ client program to access and update metadata without either being compiled against the metamodel’s generated IDL or having to use the CORBA DII.

1.2.1.3 The MOF Interfaces

The final component of the MOF specification is the set of IDL interfaces for the CORBA objects that represent an MOF metamodel. These are not of interest to the meta-modeler who will typically use vendor supplied graphical editors, compilers, and generator tools to access an MOF Model repository. However, they are of interest to MOF-based tool vendors, and to programmers who need to access metadata using the Reflective interfaces.

In fact, there is not a lot to say about these interfaces, except to explain how they were derived. In the MOF specification, the MOF Model is defined using the MOF Model as its own modeling language; that is, it is the “fixed point” of the metadata stack Conceptually, the MOF Model is M3 level metadata conforming to an M4 level metamodel that is isomorphic to the MOF Model. The IDL mapping is then applied to
this metamodel (or strictly speaking meta-metamodel) to produce the MOF Model’s IDL interfaces. Likewise, the MOF Model IDL’s operational semantics are largely defined by the mapping and the OCL constraints in the MOF Model specification.

### 1.2.2 The Relationship between XMI and MOF

The purpose of SMIF is to allow the interchange of models in a serialized form. Since the MOF is the OMG’s adopted technology for representing metadata, it is natural that the XMI specification should focus on the interchange of MOF metadata (i.e., metadata that conforms to a MOF metamodel). In fact, XMI is really a pair of parallel mappings between MOF metamodels and XML DTDs, and between MOF metadata and XML documents.

From the viewpoint users of MOF-based metadata repositories, XMI represents a new way of transferring metadata from one repository to another. Since XMI is a transfer format rather than a CORBA interface, there is no need for ORB to ORB connectivity to effect the transfer: indeed any mechanism capable of transferring ASCII text will do. Thus XMI enables a new form of metadata interchange that significantly enhances the usefulness of the MOF.

In the wider context, XMI can be viewed as a common metadata interchange format that is independent of middleware technology. Any metadata repository or tool that can encode and decode XMI streams can exchange metadata with other repositories or tools with the same capability. There is no need for to implement the MOF defined CORBA interfaces, or even to “speak” CORBA at all.

XMI provides a possible route for interchange of metadata with repositories whose metamodels are not MOF based. This interchange can be realized by ad hoc mappings between an XMI document and the repository’s native metamodel. Alternatively it can be based on mapping at the meta-metamodel level. For example, interoperability with CDIF-based repositories can be based on a mapping between the MOF Model and the CDIF meta-metamodel.

### 1.2.3 The Relationship between XMI, MOF and UML

There are two points to make under this heading. First, as mentioned above, there is a close relationship (alignment) between the meta-modeling concepts of MOF and the modeling concepts of UML. This allows the UML graphical notation to be used to express MOF meta-models. The increasing popularity of UML modeling should make an SMIF based on the MOF more accessible than an SMIF based on other meta-modeling concepts.

The second point is that the adopted OMG UML specification defines the UML metamodel as a MOF meta-model. This means that the XMI specification will lead directly to a model interchange format for UML.
1.3 XML and XML

1.3.1 The Roots of XML

The Web is the visual interface to the Internet's vast collection of resources. Today, HTML (HyperText Markup Language) is the predominant language for expressing web pages. An HTML document consists of the textual content of the document embedded in matched display tags which specify the visual presentation of the content. A well designed HTML document is visually interesting to a human viewer when displayed in a web browser. However, the automatic extraction of information from HTML documents is difficult since HTML tags are designed to express presentation rather than semantic information. This makes HTML a less than ideal medium for general electronic interchange in the Internet.

HTML is a specific tailoring of the more powerful SGML (Standard Generalized Markup Language), a sophisticated tag language that separates view from content and data from metadata. Due to SGML’s complexity, and the complexity of the tools required, it has not achieved widespread uptake.

XML, the Extensible Markup Language, is a new format designed to bring structured information to the Web. It is in effect a Web based language for electronic data interchange. XML is an open technology standard of the World Wide Web Consortium (W3C), the standards group responsible for maintaining and advancing HTML and other Web related standards.

XML is a subset of SGML that maintains the important architectural aspects of contextual separation while removing nonessential features. The XML document format embeds the content within tags that express the structure. XML also provides the ability to express rules for the structure (i.e., grammar) of a document. These two features allow automatic separation of data and metadata, and allow generic tools to validate an XML document against its grammar.

Unlike HTML, an XML document does not include presentation information. Instead, an XML document may be rendered for visual presentation by applying layout style information with technologies such as XSL (Extensible Style Language). Web sites and browsers are rapidly adding XML and XSL to their functionality.

1.3.2 Benefits of using XML

There are many advantages in basing an OMG metadata interchange format on XML. These include the following:

- XML is already an open, platform independent, and vendor independent standard.
- XML supports the international character set standards of extended ISO Unicode.
- XML is metamodel neutral and can represent metamodels compliant with OMG’s meta-metamodel, the MOF.
• The XML standard itself is programming language-neutral and API-neutral. A range of XML APIs are available, giving the programmer a choice of access methods to create, view, and integrate XML information. Leading XML APIs include DOM, SAX, and Web-DAV.

• The cost of entry for XML information providers is low. XML documents can currently be created by hand using any text editor. In the future, XML-based WYSIWYG editors with support for XSL rendering will allow creation of XML documents. XML’s tag structure and textual syntax make it as easy to read as HTML, and is clearly superior for conveying structured information.

• The cost of entry for automatic XML document producers and consumers is low. A growing set of tools is available for XML development. This includes a complete, free, commercially unrestricted XML parser written in Java available from one of the submitting companies (IBM). A variety of other XML support tools including implementations of the XML APIs are available on the Internet.

The XML approach to structured data interchange has been validated through the wide experience with XML itself and with other members of the XML family: SGML, used in high-end document processing, and HTML, the predominant language of the web.

1.3.3 XML and the Computer Industry

XML is widely believed to be the next step in the evolution of the Web. This is demonstrated by announcements by Netscape and Microsoft that upcoming versions of the leading web browsers Netscape Navigator and Internet Explorer will incorporate XML support. This kind of high profile uptake will enhance the ability of XMI documents based on XML to be integrated into the information Web of the Internet.

While XML is still in its infancy, there are many well documented applications of XML. Example application domains include web commerce, publishing, repositories, modeling, databases and data warehouses, services, financial, health care, semiconductors, inventory access, and more. Companies involved in standardizing XML include: Adobe, ArborText, DSTC, HP, IBM, Microsoft, Netscape, Oracle, Platinum, Select, Sun, and Xerox.

Widespread public interest in XML has led to a substantial number of books being written. Amazon.com lists 28 books on XML as published in the last year, including two books in the “XML for Dummies” series. The cover article of Byte Magazine’s March 1998 issue was on XML, with a multi-page article by Bill Gates.

1.3.4 How XML Works

This section provides a simple overview of XML technology. More advanced XML features are described in sections of the specification that use them.
1.3.4.1 XML Structure Elements

XML documents are tree-based structures of matched tag pairs containing nested tags and data. In combination with its advanced linking capabilities, XML can encode a wide variety of information structures. The rules that specify how the tags are structured are called a Document Type Declaration or DTD.

In the simple case, an XML tag consists of a tag name enclosed by less-than (‘<’) and greater-than (‘>’) characters. Tags in an XML document always come in pairs consisting of an opening tag and a closing tag. The closing tag in a pair has the name of the opening tag preceded by a slash symbol. Formally, a balanced tag pair is called an element, and the material between the opening and closing tags is called the element's content. The following example shows a simple element:

```xml
<Dog>a description of my dog</Dog>
```

The content of an element may include other elements, which may contain other elements in turn. However, at all levels of nesting, the closing tag for each element must be closed before its surrounding element may be closed. This requirement to balance the tags is what provides XML with its tree data structure and is a key architectural feature missing from HTML.

1.3.4.2 XML Example

This is a simple example document describing a Car. (New lines and indentation have no semantic significance in XML. They are included here simply to highlight the structure of the example document.)

```xml
<Car>
    <Make> Ford </Make>
    <Model> Mustang </Model>
    <Year> 1998 </Year>
    <Color> red </Color>
    <Price> 25000 </Price>
</Car>
```

The Car element contains five nested elements that describe it in more detail: Make, Model, Year, Color, and Price. The content of each of the nested elements encodes a value in some agreed format.

1.3.4.3 XML Attributes

In addition to contents, an XML element may contain attributes. Element attributes are expressed in the opening tag of the element as a list of name value pairs following the tag name. For example:

```xml
<Class xmi.label="c1"/>
```
XML defines a special attribute, the ID, which can be used to attach a unique identifier to an element in the context of a document. These IDs can be used to cross-link the elements to express meaning that cannot be expressed in the confines of XML’s strict tree structure.

1.3.4.4 Document Type Definitions

A Document Type Definition or DTD is XML’s way of defining the syntax of an XML document. An XML DTD defines the different kinds of elements that can appear in a valid document, and the patterns of element nesting that are allowed.

A DTD for the Car example above could contain the following declaration:

```
ELEMENT Car (Make, Model, Year, Color, Price)
```

This indicates that for a Car must contain each of the Make, Model, Year, Color, and Price elements. The declaration for an element can have a more complex grammar, including multiplicities (zero to one ‘?’ , one ‘ ’, zero or more ‘*’ , and one or more ‘+’) and logical-or ‘|’.

DTDs also define the attributes that can be included in an element using an ATTLIST. For example, the following DTD component specifies that every Class element has an optional xmi.label XML attribute and that the xmi.label consists of a character data string: (The #IMPLIED directive indicates that the attribute is optional.)

```
ATTLIST Class xmi.label CDATA #IMPLIED
```

While a DTD can be embedded in the document whose syntax it defines, DTDs are typically stored in external files and referenced by the XML document using a Universal Resource Identifier (URI) such as

```
“http://www.xmi.org/car.dtd”
```

or

```
“file:car.dtd”
```

1.3.4.5 XML Document Correctness

There are three levels of correctness associated with XML document; well-formedness, validity, and semantic correctness:

1. A “well-formed” XML document is one where the elements are properly structured as a tree with the opening and closing tags correctly nested. Well-formed documents are essential for information exchange.

2. A “valid” XML document is one that is well-formed and that conforms to the structure defined by a DTD. A valid document will only contain elements and attributes defined in the DTD. Similarly, the element contents and attribute values
will conform to the DTD. While the DTD need not be specified in an XML document, and a consumer need not use the DTD when decoding the document, the DTD is essential for checking validity.

3. The highest level of document correctness (“semantic correctness”) is beyond the scope of XML and DTDs as they are currently defined. Only an XML document consumer with deep domain knowledge can check that the information in an XML document makes sense. In the Car example, this might include a check that a particular Color was available for a given combination of Make, Model, and Year.

### 1.3.5 XML and the OMG

There is strong synergy between the OMG technologies and XML in a number of areas. OMG defines CORBA as the medium for interchange of data between objects that have (inter-)network connectivity. XML represents a potential alternative interchange medium for cases where ORB to ORB connectivity is not possible. Furthermore, XML presents a possible medium for interchange of data between CORBA based systems and other systems.

The OMG’s MOF specification defines a common framework for representing metadata. At the moment, the MOF is restricted in providing metadata for CORBA based systems since the only defined way to interchange MOF metadata is to use the CORBA interfaces produced by the MOF’s IDL mapping. XML (in the form of XMI) provides a way to lift this restriction.

OMG can use the MOF and XMI to expand the significance of the current OMG activities, which are producing Domain Service specifications. If a Domain Service specification includes a normative MOF-based metamodel, XMI can then be used to generate XML DTDs for these metamodels. These DTDs would allow interchange of metadata between and beyond CORBA-based systems, increasing relevance for the Domain Service specifications. There is considerable scope for duplicating this pattern for data interchange.

The XMI submitters believe that this approach would enhance the OMG’s position as providing leadership in the data and metadata interchange standards of the future.

### 1.3.6 New XML Technologies

The XML family of standards is currently undergoing rapid development. This section gives capsule summaries of important new XML technologies, which are in the process of being standardized by the W3C and other organizations. While the XMI specification is designed to be upwards compatible with these technologies, it is rather difficult to use them in this specification. In the future when the technology has stabilized and been standardized it may well be feasible to revise XMI to make use of them. XMI has been designed to be upwards compatible with these upcoming XML technologies and provide facilities for their use where possible.
Namespaces - The namespace draft by the W3C is a work in progress with the goal of providing support for multiple DTDs in the same document. Each DTD is given a local namespace within a document (no global registration necessary) which prevents any conflicts by differing definitions of similarly named constructs.

Links - There are two linking technology drafts in progress at the W3C that provide advanced linking facilities, which are integrated with web technology. XLink is for cross document links and XPointer is for links within a document. They are used together and are discussed in more detail in Section 3.8, “Linking,” on page 3-21.

There are three proposals for enhancing the base capabilities of XML at the W3C:

- RDF (Resource Description Framework) is a working draft specification for infrastructure to support web information based on the entity-relationship model.
- RDF-Schema is a working draft to provide types for XML. XML-Data is a note to the W3C for public comment on providing schemas and types for XML. The latter is particularly significant to XMI, and future incorporation would be of great benefit. XML-Data has been superseded by DCD (Document Content Definition) - a proposal to provide data type support and a new syntax for DTDs.
- XSL - Extensible Style Language is a working draft of the W3C that specifies user-definable declarative transforms of XML documents with the goal of providing formatting style information. XSL is used in conjunction with XML to create the visual layout of the underlying XML data and metadata.

There are three major APIs to XML:

- DOM, the Document Object Model, is a language-neutral interface to XML documents for creation and reading data and metadata information. DOM also works with style processing and scripts.
- SAX is an event-driven API for XML parsing.
- Web-DAV is an API for Web based Distributed Authoring and Versioning and is currently a working draft of the IETF (Internet Engineering Task Force) standards body. It uses the HTTP protocol to provide on-line, distributed XML access and modification.
Usage Scenarios

Contents

This chapter includes the following topics.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Purpose”</td>
<td>2-1</td>
</tr>
<tr>
<td>“Combining Tools in a Heterogeneous Environment”</td>
<td>2-1</td>
</tr>
<tr>
<td>“Co-operating with Common Metamodel Definitions”</td>
<td>2-2</td>
</tr>
<tr>
<td>“Work Environment”</td>
<td>2-3</td>
</tr>
<tr>
<td>“Promoting Design Patterns and Reuse”</td>
<td>2-3</td>
</tr>
</tbody>
</table>

2.1 Purpose

This chapter describes some of the problems that IT users and vendors face today and illustrates how XMI helps to address these problems.

2.2 Combining Tools in a Heterogeneous Environment

Implementing an effective and efficient IT solution for an enterprise requires a detailed understanding of processes, rules, and data used by the business and how each map to supporting applications. Without this information, it is difficult to assess the effectiveness of the application components in use, to identify opportunities for improvement, and to evaluate candidate solutions. A further complication is that the applications in use will probably originate from a variety of sources and consequently be a mix of custom solutions and packaged applications implemented in a variety of technologies.
The reality is that no single tool exists for both modeling the enterprise and documenting the applications that implement the business solution. A combination of tools from different vendors is necessary but difficult to achieve because the tools often cannot easily interchange the information they use with each other. This leads to translation or manual re-entry of information, both of which are sources of loss and error.

XMI eases the problem of tool interoperability by providing a flexible and easily parsed information interchange format. In principle, a tool needs only to be able to save and load the data it uses in XMI format in order to inter-operate with other XMI capable tools. There is no need to implement a separate export and import utility for every combination of tools that exchange data.

The makeup of an XMI stream is important too. It contains both the definitions of the information being transferred as well as the information itself. Including the semantics of the information in the stream enables a tool reading the stream to better interpret the information content. A second advantage of including the definitions in the stream is that the scope of information that can be transferred is not fixed; it can be extended with new definitions as more tools are integrated to exchange information.

2.3 Co-operating with Common Metamodel Definitions

The extent of the information that can be exchanged between two tools is limited by how much of the information can be understood by both tools. If they both share the same metamodel (the definition of the structure and meaning of the information being used), all of the information transferred can be understood and used. However, gaining consensus on a totally shared metamodel is a difficult task even within a single company. It is more likely that a subset of the metamodel can be shared with each tool adding its own extensions. The need to agree the structure and syntax for encoding as a stream adds further complexity.

XMI builds on the OMG Meta Object Facility that already provides a standard way to define metamodels within the OMG. UML is one example of a metamodel that can be defined in the MOF and which has already been adopted as a standard by the OMG. The model definitions required for the transfer of UML models using XMI are included with this specification as a set of concrete XML DTDs. Any tool vendor can use these definitions to save and load UML models in XMI format without the need for an implementation of the MOF. This is a practical step to encourage as many tool vendors as possible to adopt the standard by keeping their initial investment low.

However, manually writing the XML DTDs for a metamodel is tedious, error prone, and subject to variations in how model concepts are implemented in XML. Using XMI, the XML DTDs for a metamodel are obtained by defining the metamodel in MOF and then applying the XMI generation rules. The generation approach ensures that a given metamodel will always map to the same set of XML DTDs regardless of which vendor implemented the MOF and the XMI stream protocol.

The fact that the MOF meta-metamodel, (the description of the MOF itself), can be defined in the MOF itself means that XMI can also be used to transfer metamodel definitions from one MOF to another. Being able to share metamodel definitions is an
important step to promoting the use of common metamodels by different tool vendors. The combination of the MOF and XMI provides an effective way for vendors to co-operate on the definition and use of common models.

As mentioned earlier, having a shared model is not enough on its own. Each vendor must be able to extend the information content of the model to include items of information that have not been included in the shared model. XMI allows a vendor to attach additional information to shared definitions in a way that allows the information to be preserved and passed through a tool that does not understand the information. Loss-less transfer of information through tools is necessary to prevent errors that may be introduced by the filtering effect of a tool passing on only that information it can understand itself. Using this extension mechanism, XMI stream can be passed from tool to tool without suffering information loss.

2.4 Work Environment

2.4.1 Working in a Distributed and Intermittently Connected Environment

Another aspect of sharing metadata is encountered when trying to provide effective consultancy services. This requires the ability to exploit and share best practices between the consultants of the group. However, consultants on site typically have restricted connectivity to the network and limited bandwidth for exchanging models and design information with their colleagues.

The use of XMI for a metadata interchange facilitates the exchange of model and design data over the Internet and by phone. Appearing as a set of hyper-linked Internet documents, the data to be transferred can be transported easily through firewalls and downloaded using a modem. The documents in a related set are accessed on-demand and cached locally to eliminate the retransmission of frequently used sub-documents.

The remote consultant would be equipped with a notebook installed with a set of tools that can import and export metadata in XMI format. Connecting to the home site via the Internet or dialup networking, the consultant can download metadata resources published as links from pages on a standard WEB server. The same mechanism can be used to upload modification that the consultant wants to publish for his colleagues.

Typically, the type definitions that define the semantics of a transfer do not change frequently and can be stored in a separate document from the actual data to be transferred. The type definitions are versioned to allow consistency checking. On the first use of the type definitions, the document containing the type definitions would be downloaded and cached on the consultant’s machine. Subsequent transfers are faster because only the metadata content is transferred while the cached type definitions are reused.

2.5 Promoting Design Patterns and Reuse

Consultants will often need to integrate their work with the development tools being used at a customer site. This often results in the consultants actually using the same
tool set as the customer. Of course, the tools used will differ from customer to customer.

The problem in this scenario is that it is difficult to develop and exploit best practices across the consulting group without being able to exchange model and design data between different tool sets.

XMI addresses this problem by defining a standard format for interchange of model and design data between different tool sets. It does not require the tool vendors to invest in the same technology stack. It only requires them to agree on the Meta models for the data to be shared, plus a standard mechanism for extending that Meta model with their own types of metadata.

The XMI format allows Meta models to be standardized and revised over time, the set of Meta models being extensible. For example, this specification covers just the UML Meta model, but other Meta models can be agreed and added without affecting the current set of Meta models.

Vendor extensions to a standard meta model are designed to enable other vendors tools to process and use the standardized information while being able to easily retain and pass through vendor specific extensions.
XMI DTD Design Principles

Contents

This chapter includes the following topics.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Purpose”</td>
<td>3-1</td>
</tr>
<tr>
<td>“Use of XML DTDs”</td>
<td>3-2</td>
</tr>
<tr>
<td>“Basic Principles”</td>
<td>3-3</td>
</tr>
<tr>
<td>“XMI DTD and Document Structure”</td>
<td>3-5</td>
</tr>
<tr>
<td>“Necessary XMI DTD Declarations”</td>
<td>3-6</td>
</tr>
<tr>
<td>“Metamodel Class Specification”</td>
<td>3-14</td>
</tr>
<tr>
<td>“Transmitting Incomplete Metadata”</td>
<td>3-20</td>
</tr>
<tr>
<td>“Linking”</td>
<td>3-21</td>
</tr>
<tr>
<td>“Transmitting Metadata Differences”</td>
<td>3-24</td>
</tr>
<tr>
<td>“Document Exchange with Multiple Tools”</td>
<td>3-27</td>
</tr>
<tr>
<td>“General Datatype Mechanism”</td>
<td>3-30</td>
</tr>
</tbody>
</table>

3.1 Purpose

This chapter contains a description of the XML Document Type Definitions (DTDs) that may be used with the XMI specification to allow some metamodel information to be verified through XML validation. The use of DTDs in XMI is described first, followed by a brief description of some basic principles, which includes a short description of each XML attribute and XML element defined by XMI. Those descriptions are followed by more complete descriptions that provide examples
illustrating the motivation for the XMI DTD design in the areas of metamodel class specification, transmitting incomplete metadata, linking, transmitting metadata differences, and exchanging documents between tools.

It is possible to define how to automatically generate a DTD from the MOF metamodel to represent any MOF-compliant metamodel.

### 3.2 Use of XML DTDs

An XML DTD provides a means by which an XML processor can validate the syntax and some of the semantics of an XML document. This specification provides rules by which a DTD can be generated for any valid XMI-transmissible MOF-based metamodel. However, the use of DTDs is optional; an XML document need not reference a DTD, even if one exists. The resulting document can be processed more quickly, at the cost of some loss of confidence in the quality of the document.

It can be advantageous to perform XML validation on the XML document containing MOF metamodel data. If XML validation is performed, any XML processor can perform some verification, relieving import/export programs of the burden of performing these checks. It is expected that the software program that performs verification will not be able to rely solely on XML validation for all of the verification, however, since XML validation does not perform all of the verification that could be done.

Each XML document that contains metamodel data conforming to this specification contains: XML elements that are required by this specification, XML elements that contain data that conform to a metamodel, and, optionally, XML elements that contain metadata that represent extensions of the metamodel. Metamodels are explicitly identified in XML elements required by this specification. Some metamodel information can also be encoded in an XML DTD. Performing XML validation provides useful checking of the XML elements that contain metadata about the information transferred, the transfer information itself, and any extensions to the metamodel.

It is possible to use an internal DTD to provide all of the declarations of XML elements described in this chapter. However, it is advantageous to use an external DTD, because the DTD need not be transmitted along with each XML document that contains the metadata. An internal DTD may be used in addition to an external DTD, for example to specify extensions to the metamodel.

The XML Namespace specification has been adopted by the W3C, allowing XMI to use multiple metamodels at the same time. The local namespace name acts as a prefix to all the elements declared in a DTD and avoids any name collisions so that it will not be necessary to use fully qualified names for XMI elements.
3.2.1 XML Validation of XMI Documents

XML validation can determine whether the XML elements required by this specification are present in the XML document containing metamodel data, whether XML attributes that are required in these XML elements have values for them, and whether some of the values are correct.

XML validation can also perform some verification that the metamodel data conforms to a metamodel. Although some checking can be done, it is impossible to rely solely on XML validation to verify that the information transferred satisfies all of a metamodel’s semantic constraints. Complete verification cannot be done through XML validation because it is not currently possible to specify all of the semantic constraints for a metamodel in an XML DTD, and the rules for automatic generation of a DTD preclude the use of semantic constraints that could be encoded in a DTD manually, but cannot be automatically encoded.

Finally, XML validation can be used to validate extensions to the metamodel, because extensions must be represented as elements declared in either the external DTD or the internal DTD.

3.2.2 Requirements for XMI DTDs

Each DTD used by XMI must satisfy the following requirements:

- All XML elements defined by the XMI specification must be declared in the DTD.
- Each metamodel construct (class, attribute, and association) must have a corresponding element declaration, and may have an XML attribute declaration, as described below. The element declaration may be defined in terms of entity declarations, also, as described below.
- Any XML elements that represent extensions to the metamodel must be declared in the external DTD or internal DTD.

It is permissible for users of XMI to generate a DTD, which relaxes the multiplicities described in Section 3.6, “Metamodel Class Specification,” on page 3-14” to enable incomplete models to be transmitted according to this specification. See Section 3.7, “Transmitting Incomplete Metadata,” on page 3-20” for further details.

3.3 Basic Principles

This section discusses the basic organization of an XML DTD for XMI. Detailed information about each of these topics is included later in this chapter.

3.3.1 Required XML Declarations

This specification requires that a number of XML element declarations be included in DTDs that enable XML validation of metadata that conforms to this specification. These declarations must be included in the DTD because there is no mechanism currently available in XML to validate a document against more than one external
Some of these XML elements contain metadata about the metadata to be transferred, for example, the identity of the metamodel associated with the metadata, the time the metadata was generated, the tool that generated the metadata, whether the metadata has been verified, etc.

All XML elements defined by this specification have the prefix “XML.” They have this prefix to avoid name conflicts with XML elements that would be a part of a metamodel. After XML Schemas become a W3C recommendation rather than a working draft, it may be possible to place all of the required XML elements in a single Schema and use the XML namespace mechanism to avoid name conflicts.

In addition to required XML element declarations, there are some attributes that must be defined according to this specification. Every XML element that corresponds to a metamodel class must have attributes that enable the XML element to act as a proxy for a local or remote XML element. These attributes are used to associate an XML element with another XML element.

Most of the XML attributes defined by this specification have the prefix “xmi;” however, the XML attributes of XMI elements defined by this specification do not, in general, have that prefix.

### 3.3.2 Metamodel Class Representation

Every metamodel class is represented in the DTD by an XML element whose name is the class name. The element definition lists the attributes of the class; references to association ends relating to the class; and the classes that this class contains, either explicitly or through composition associations. In XMI 1.1, the content models of XML elements corresponding to metamodel classes no longer impose an order on the attributes and references.

Every attribute of a metamodel class is represented in the DTD by an XML element whose name is the attribute name. In addition, attributes that have primitive or enumeration data types are represented in the DTD by an XML attribute declaration, as described below. The attributes are included in the content model of the XML element corresponding to the metamodel class, in any order, as described below.

Each association (both with and without containment) between metamodel classes is represented by two XML elements that represent the roles of the association ends. The multiplicities of the association ends are not included in the DTD. The content model of the XML element that represents the container class has an XML element with the name of the role at the association end. The XML element representing the role has a content model that allows XML elements representing the associated class and any of its subclasses to be included.

### 3.3.3 Metamodel Extension Mechanism

Every XMI DTD contains a mechanism for extending a metamodel class. Any number of XML.extension elements are included in the content model of any class. These extension elements have a content model of ANY, allowing considerable freedom in the nature of the extensions. In addition, the top level XMI element may contain zero
or more `XML.extensions` elements, which provides for the inclusion of any new information. One use of the extension mechanism might be to associate display information for a particular tool with the metamodel class represented by the XML element. Another use might be to transmit data that represents extensions to a metamodel.

Tools that rely on XMI are expected to store the extension information and export it again to enable round trip engineering, even though it is unlikely they will be able to process it further. Also, any XML elements that are put in either the `XML.extension` or `XML.Extensions` elements must be declared in either the internal DTD or external DTD.

### 3.4 XMI DTD and Document Structure

Every XMI DTD consists of the following declarations:

- An XML version processing instruction. Example: `<? XML version="1.0" ?>`
- An optional encoding declaration that specifies the character set, which follows the ISO-10646 (also called extended Unicode) standard. Example: `<? XML version="1.0" ENCODING="UCS-2" ?>`
- Any other valid XML processing instructions.
- The required XMI declarations specified in Section 3.5, “Necessary XMI DTD Declarations,” on page 3-6.
- Declarations for a specific metamodel.
- Declarations for differences.
- Declarations for extensions.

Every XMI document consists of the following declarations:

- An XML version processing instruction.
- An optional encoding declaration that specifies the character set.
- Any other valid XML processing instructions.

XMI imposes no ordering requirements beyond those defined by XML. XML Namespaces may also be declared in the XMI element as described below.

The top element of the XMI information structure is the XMI element. An XML document containing only XMI information will have XMI as the root element of the document. It is possible for future XML exchange formats to be developed which extend XMI and embed XMI elements within their XML elements.
3.5 **Necessary XMI DTD Declarations**

This section declares the elements and element attributes whose definitions must appear in valid XMI DTDs.

3.5.1 **Necessary XMI Attributes**

3.5.1.1 **Element Identification Attributes**

Three XML attributes are defined by this specification to identify XML elements so that XML elements can be associated with each other. The purpose of these attributes is to allow XML elements to reference other XML elements using XML IDREFs, XLinks, and XPointers.

These attributes are declared in an XML entity called `XML.element.att`. Placing these attributes in an XML entity prevents errors in the declarations of these attributes in DTDs. Its declaration is as follows:

```
<!ENTITY % XMI.element.att 'xmi.id       ID #IMPLIED
xmi.label CDATA #IMPLIED
xmi.uuid  CDATA #IMPLIED ' >
```

**xmi.id**

XML semantics require the values of this attribute to be unique within an XML document; however, the value is not required to be globally unique. This attribute may be used as the value of the `xmi.idref` attribute defined in the next section. It may also be included as part of the value of the `href` attribute in XLinks. An example of the use of this attribute and the other attributes in this section can be found in Section 3.8.3, “Example from UML,” on page 3-23.”

**xmi.label**

This attribute may be used to provide a string label identifying a particular XML element. Users may put any value in this attribute.

**xmi.uuid**

The purpose of this attribute is to provide a globally unique identifier for an XML element. The values of this attribute should be globally unique strings prefixed by the type of identifier. If the `refMofId()` operation defined by MOF returns a value, it is recommended that you use the value as the value of the `xmi.uuid` XML attribute, although you are not required to do so. For example, to include a DCE UUID as defined by The Open Group, the UUID would be preceded by “DCE.” The values of this attribute may be used in the `href` attribute in simple XLinks. XMI does not specify which UUID convention is chosen.
The form of the UUID (Universally Unique Identifier) is taken from a standard defined by the Open Group (formerly the Open Software Foundation). This standard is widely used, including by Microsoft for COM (GUIDs) and by many companies for DCE, which is based on CORBA. The method for generating these 128-bit IDs is published in the standard and the effectiveness and uniqueness of the IDs is not in practice disputed.

When a UUID is placed in an XMI file, the form is “id namespace:uuid.” The id namespace of UUIDs is typically DCE. An example is “DCE:2fac1234-31f8-11b4-a222-08002b34c003.”

3.5.1.2 Linking Attributes

XMI requires the use of several XML attributes to enable XML elements to refer to other XML elements using the values of the attributes defined in the previous section. The purpose of these attributes is to allow XML elements to act as simple XLinks or to hold a reference to an XML element in the same document using the XML IDREF mechanism. See Section 3.8, “Linking,” on page 3-21 for more information on linking.

The attributes described in this section must be included in a DTD as an XML entity. The entity must be declared as follows:

```xml
<!ENTITY % XMI.link.att
  'href CDATA #IMPLIED
  xmi.idref IDREF #IMPLIED' >
```

The link attributes act as a union of three linking mechanisms, any one of which may be used at one time. The mechanisms are the XLink `href` for advanced linking across or within a document, or the `xmi.idref` for linking within a document.

**Simple XLink Attributes**

The `href` attribute declared in the above entity enables an XML element to act in a fashion compatible with the simple XLink according to the XLink and XPointer W3C working drafts. The declaration and use of `href` is defined in the XLink and XPointer specifications. XMI enables the use of simple XLinks. XMI does not preclude the use of extended XLinks. Since the form of extended links is undergoing further development in the XLink specification, no recommendations for their use in XMI are given at this time. Since XLink defines many additional XML attributes, some of which may be useful in rare circumstances, it is permissible to use those additional attributes provided that they are prefixed by the “xlink:” namespace. This decouples XMI from a dependency on rarely used attributes in the W3C XLink working draft.

To use simple XLinks, set the `href` to the URL of the desired location. The `href` attribute can be used to reference XML elements whose `xmi.id`, `xmi.label`, or `xmi.uuid` attributes are set to particular values. The `xmi.id` attribute value can be specified using a special URI form for XPointers defined in the XLink and XPointer working drafts.
xmi.idref
This attribute allows an XML element to refer to another XML element within the same document using the XML IDREF mechanism. In XMI documents, the value of this attribute should be the value of one of the xmi.id attributes.

xmi.uuidref
This attribute is no longer used in XMI 1.2. In XMI 1.0, this attribute provides a mechanism for referring to another XML element within the same document by using a UUID specified in the xmi.uuid attribute of another XML element. The value of this attribute should be the value of a UUID, although XML does not enforce this restriction. [DCE]

3.5.2 Common XMI Elements
Every XMI-compliant DTD must include the declarations of the following XML elements:

- XMI
- XMI.header
- XMI.content
- XMI.extensions
- XMI.extension
- XMI.documentation
- XMI.owner
- XMI.contact
- XMI.longDescription
- XMI.shortDescription
- XMI.exporter
- XMI.exporterVersion
- XMI.exporterID
- XMI.notice
- XMI.model
- XMI.metamodel
- XMI.metametamodel
- XMI.import
- XMI.difference
- XMI.delete
- XMI.add
The following declarations are required if used by the particular metamodel:

- XMI.replace
- XMI.reference

The following declarations are required if used by the particular metamodel:

- XMI.field
- XMI.seqItem
- XMI.sequence
- XMI.enum

### 3.5.3 XMI

The top level XML element for each XMI document is the XMI element. Its declaration is:

```xml
<!ELEMENT XMI (XMI.header?,
               XMI.content?,
               XMI.difference*,
               XMI.extensions*) >
<!ATTLIST XMI
  xmi.version CDATA #FIXED "1.2"
  timestamp   CDATA #IMPLIED
  verified (true | false) #IMPLIED
>```

The `xmi.version` attribute is required to be set to “1.2.” This indicates that the metadata conforms to this version of the XMI specification. Revised versions of this standard will have another number associated with them, but there is no guarantee that any particular numbering scheme will be used. The `timestamp` indicates the date and time that the metadata was written. The `verified` attribute indicates whether the metadata has been verified. If it is set to “true,” verification of the model was performed by the document creator at the full semantic level of the metamodel. In that case, XML validation should find errors only in encoding or transmission.

The format for timestamps is not defined in this specification.

In addition to the fixed XMI element’s attributes, the namespaces used within XMI may be declared, either in an internal or external DTD. Each namespace “n” used in the XMI element is declared as follows:

```xml
<!ATTLIST XMI xmlns:n #CDATA IMPLIED>
```

The generated DTDs following the production rules will provide this attribute for the generated metamodel. When combining multiple metamodels and also using DTDs, the DTDs should be concatenated (with the fixed declarations included only once) and the ATTLIST declarations for all of the DTDs used.
3.5.4 XMI.header

The XMI.header element contains XML elements that identify the model, metamodel, and metametamodel for the metadata, as well as an optional XML element, which contains various information about the metadata being transferred. This XML element is now optional in XMI 1.2. The declaration is:

```xml
<!ELEMENT XMI.header (XMI.documentation?,
                       XMI.model*,
                       XMI.metamodel*,
                       XMI.metametamodel*,
                       XMI.import*) >
```

3.5.5 XMI.content

The XMI.content XML element contains the actual metadata being transferred. It may represent model information or metamodel information. Its declaration is:

```xml
<!ELEMENT XMI.content ANY >
```

3.5.6 XMI.extensions

The XMI.extensions element contains XML elements that contain metadata, which is an extension of the metamodel. This information might include presentation information associated with the metadata, for example. Its declaration is:

```xml
<!ELEMENT XMI.extensions ANY >
<!ATTLIST XMI.extensions
  xmi.extender CDATA #REQUIRED
>
```

The xmi.extender attribute should indicate which tool made the extension. It is provided so that tools may ignore the extensions made by other tools before the content of the XMI.extensions element is processed.

3.5.7 XMI.extension

The XMI.extension element contains XML elements that also contain metadata, which is an extension of the metamodel. This element can be directly included in XML elements in the content section of an XMI document to associate the extension metadata with a particular XML element. Its declaration is:

```xml
<!ELEMENT XMI.extension ANY >
<!ATTLIST XMI.extension
  %XML.element.att;
  %XML.link.att;
  xmi.extender CDATA #REQUIRED
  xmi.extenderID CDATA #IMPLIED
>
```
The `xmi.extender` attribute should indicate which tool made the extension. It is provided so that tools may ignore the extensions made by other tools before the content of the XMI.extensions element is processed. The `xmi.extenderID` is an optional internal ID from the extending tool. The other attributes allow individual extensions to be identified and to act as proxies for local or remote extensions.

### 3.5.8 XMI.documentation

This XML element contains information about the metadata being transmitted, for instance the owner of the metadata, a contact person for the metadata, long and short descriptions of the metadata, the exporter tool that created the metadata, the version of the tool, and copyright or other legal notices regarding the metadata. In addition, other information can be included as text within this element, since its content model is mixed. The declaration is:

```xml
<!ELEMENT XMI.documentation (#PCDATA | XMI.owner | XMI.contact | XMI.longDescription | XMI.shortDescription | XMI.exporter | XMI.exporterVersion | XMI.notice)* >
<!ELEMENT XMI.owner ANY >
<!ELEMENT XMI.contact ANY >
<!ELEMENT XMI.longDescription ANY >
<!ELEMENT XMI.shortDescription ANY >
<!ELEMENT XMI.exporter ANY >
<!ELEMENT XMI.exporterVersion ANY >
<!ELEMENT XMI.exporterID ANY >
<!ELEMENT XMI.notice ANY >
```

### 3.5.9 XMI.model

This XML element identifies the model to which the instance data being transferred conforms. There may be multiple models, if the model to which the instance data being transferred conforms to more than one model. This element is expected to become a simple XLink when it becomes a recommendation of the W3C. Its declaration is:

```xml
<!ELEMENT XMI.model ANY>  
<!ATTLIST XMI.model %XMI.link.att;  
 xmi.name CDATA #REQUIRED  
 xmi.version CDATA #REQUIRED
>`
```

The `xmi.name` and `xmi.version` attributes are the name and version of the model described in the enclosed `XMI.content`, respectively. The `href` attribute may contain a physical URI that contains model data. Since the content is ANY, additional documentation is possible.
3.5.10 XMI.metamodel

This XML element identifies the metamodel to which the model data that is transferred conforms. There may be multiple metamodels, if the model data that is transferred conforms to more than one metamodel. Including this element enables tools to perform more verification of the metadata to the metamodel than is possible to perform by XML validation. This element is expected to become a simple XLink when it becomes a recommendation of the W3C. Its declaration is:

```xml
<!ELEMENT XMI.metamodel ANY>
<!ATTLIST XMI.metamodel
    %XMI.link.att;
    xmi.name    CDATA #REQUIRED
    xmi.version CDATA #REQUIRED
>
```

The `xmi.name` and `xmi.version` attributes are the name and version of the metamodel, respectively. The `href` attribute may contain a physical URI that contains metamodel data. Since the content is ANY, additional documentation is possible.

3.5.11 XMI.metametamodel

This XML element identifies the metametamodel to which the metadata that is transferred conforms. This element will often refer to the MOF version that was used. Including this element enables tools to perform more verification of the metadata to the metamodel than is possible to perform by XML validation. This element is expected to become a simple XLink when it becomes a recommendation of the W3C. Its declaration is:

```xml
<!ELEMENT XMI.metametamodel ANY>
<!ATTLIST XMI.metametamodel
    %XMI.link.att;
    xmi.name    CDATA #REQUIRED
    xmi.version CDATA #REQUIRED
>
```

The `xmi.name` and `xmi.version` attributes are the name and version of the metametamodel, respectively. The `href` attribute may contain a physical URI that contains metamodel data. Since the content is ANY, additional documentation is possible.

3.5.12 XMI.import

This XML element identifies additional documents that are needed to process the current document; it points to other documents that define metadata that defines the metadata in the document in which it appears. Its declaration is:

```xml
<!ELEMENT XMI.import ANY>
<!ATTLIST XMI.import
    %XMI.link.att;
```
The `xmi.name` and `xmi.version` attributes are the name and version of the imported model, respectively. The `href` attribute may contain a physical URI that contains model data. Since the content is ANY, additional documentation is possible.

### 3.5.13 XMI.difference

This XML element holds XML elements representing differences to base data. Users may use it within the content part of an XMI file or in a separate XMI.difference section. The attributes in this element allow references to be made to other elements using XLinks, XPointers, or IDREFs. Its declaration is:

```xml
<!ELEMENT XMI.difference (XMI.difference | XMI.add | XMI.delete | XMI.replace)* >
<!ATTLIST XMI.difference
  %XMI.element.att;
  %XMI.link.att;
>
```

### 3.5.14 XMI.delete

This XML element represents a deletion to base metadata. It must be within an `XMI.difference` XML element. The attributes in this element allow references to be made to other elements using XLinks, XPointers, or XML IDREFs. Its declaration is:

```xml
<!ELEMENT XMI.delete EMPTY >
<!ATTLIST XMI.delete
  %XMI.element.att;
  %XMI.link.att;
>
```

### 3.5.15 XMI.add

This XML element represents an addition to base metadata. It must be within an `XMI.difference` XML element. The attributes in this element allow references to be made to other elements using XLinks, XPointers, or XML IDREFs. Its declaration is:

```xml
<!ELEMENT XMI.add ANY >
<!ATTLIST XMI.add
  %XMI.element.att;
  %XMI.link.att;
  xmi.position CDATA "-1"
>
```

The `xmi.position` attribute indicates where to place the addition relative to other XML elements.
3.5.16 XMI.replace

This XML element represents a replacement of base metadata with other metadata. It must be within an XMI.difference XML element. The attributes in this element allow references to be made to other elements using XLinks, XPointers, or XML IDREFs. Its declaration is:

```xml
<!ELEMENT XMI.replace ANY >
<!ATTLIST XMI.replace
  %XMI.element.att;
  %XMI.link.att;
  xmi.position CDATA "-1"
>
```

The xmi.position attribute indicates where to place the contents of the replacement element relative to other XML elements.

3.5.17 XMI.reference

This XML element allows references to other XML elements within an attribute of type string or an XMI.any element, which represents a data type that is not defined in the metamodel. It should be used within an XMI.any element or in attributes to specify a remote value. Its declaration is:

```xml
<!ELEMENT XMI.reference ANY >
<!ATTLIST XMI.reference %XMI.link.att; >
```

For information on how to use the link attributes, see Section 3.8, “Linking,” on page 3-21.

3.6 Metamodel Class Specification

This section describes in detail how to represent information about metamodel classes in an XMI compliant DTD. It uses the rules for generating a DTD as described in the “XML DTD Production” chapter to describe the manner in which attributes, associations, and containment relationships are represented in an XML DTD, and how inheritance between metamodel classes is handled. It uses a short example to explain the encoding.

3.6.1 Namespace Qualified XML Element Names

In XMI 1.0, the use of fully qualified names was mandatory, since the W3C Namespace recommendation was not finalized by the time XMI 1.0 was written. Now that Namespaces are an official recommendation, it is no longer necessary to use fully qualified names; however, there was no provision in the Namespace recommendation
for using XML Namespaces in conjunction with XML validation. The use of Namespaces with XMI 1.2 documents is restricted as described below to enable validation with DTDs.

When the official DTD for a metamodel is produced, the DTD generator may choose a namespace name that all documents to be validated with the DTD must use. That namespace name followed by “:” becomes the prefix for each tag name declared in an XMI DTD that corresponds to the metamodel.

The XML element name for each metamodel class, package, and association is its name prefixed by the namespace. The name for tags corresponding to metamodel attributes and references is the XML element name of the class followed by “.” followed by the name of the attribute or reference. The name of XML attributes corresponding to metamodel references and metamodel attributes is the name of the reference or attribute, since each tag in XML has its own namespace. An example of namespaces is given in Appendix C.

The responsibility of ensuring uniqueness of names for DTD generation belongs to the metamodel owner. In the event of duplicate names, the preferred resolution is to place the duplicates in a different document and assign a second namespace. Alternatively, an additional namespace is assigned to the document package containing the duplicates. With nested packages, it is possible for class names to conflict. If this happens, you need to assign namespaces to the packages and use the namespace prefixes to create unique names for the XML elements corresponding to classes.

Each namespace is assigned a logical and a physical URI. The logical URI is placed in the namespace declaration of the XMI element in XML documents that contain instances of the metamodel and the physical URI is placed in the XMI.metamodel tag. The XML namespace specification assigns logical names to namespaces that are expected to remain fixed throughout the life of all uses of the namespace since it provides a permanent global name for the resource. An example is “org.omg/standards/UML.” There is no requirement or expectation by the XML Namespace specification that the logical URI be resolved or dereferenced during processing of XML documents. The physical URI is the mechanism for resolving where the actual document may be found. The physical URI could be local, as in “UML13.xml” or remote as in “ftp://server.omg.org/resources/xmi/UML13.xml.” The namespace name links the logical URI declared in the XMI element with the physical URI in the XMI.metamodel element.

The following is an example of a UML model in an XMI document using namespaces.

```xml
<XMI xmi.version="1.2" xmlns:UML="org.omg/standards/UML">
  <XMI.header>
    <XMI.metamodel name="UML" version="1.3" href="UML.xml"/>
    <XMI.model name="example" version="1" href="example.xml"/>
  </XMI.header>
  <XMI.content>
    <UML:Class name="C1">
      <UML:Classifier.feature>
        <UML:Attribute name="a1" visibility="private"/>
      </UML:Classifier.feature>
    </UML:Class>
  </XMI.content>
</XMI>```
The model has a single class named C1 that contains a single attribute named a1 with visibility private. The XMI element declares the version of XMI and the namespace for UML with the logical URI. The XMI.metamodel has the same name “UML” and an href to the physical location where the UML.xml file is located. The model name is “example.” The XMI.content contains the model, using the new XML attributes.

3.6.2 Metamodel Multiplicities

In XMI 1.0 a mapping was defined between the multiplicities in a metamodel and XML multiplicities. To enforce the multiplicities, it was necessary to define an order to XML elements corresponding to attributes, and association ends in a metamodel, due to a limitation of XML 1.0. The order that was specified is not necessary for exchanging data, and makes all XMI document writers conform to the order for the documents they produce to validate.

The XMI RTF has concluded that it is better to give document producers flexibility in the order that XML elements appear than to enforce the multiplicity. Therefore, in XMI 1.2, the multiplicities from the metamodel are ignored when generating a DTD.

3.6.3 Class Specification

The representation of a metamodel class named “C” is shown below for the simplest case where “C” does not have any attributes, associations, or containment relationships:

```xml
<!ELEMENT C (XML.extension)*>
<!ATTLIST C
  %XML.element.att;
  %XML.link.att;
>
```

3.6.4 Inheritance Specification

XML does not currently have a built-in mechanism to represent inheritance. In its place, XMI specifies that inheritance will be copy-down inheritance. Inheritance is represented by using the local and inherited attributes, references, and compositions in the declaration of a class. The content model of the XML element declaration for a class contains XML elements representing both the local and inherited attributes, references, and compositions. If the inherited attributes and references have corresponding XML attribute declarations, they are included along with the XML attribute declarations for the local attributes and references for a class.
For example, consider a class “C1” that has a superclass “C0.” Class “C0” has attribute “a0,” reference “r0,” and composition “comp0.” Class “C1” has attribute “a1,” reference “r1,” and composition “comp1.” The MOF attributes and references have both XML element and XML attribute declarations according to the DTD production rules. The declaration of the XML element corresponding to class C1 includes the inherited features as well as the local features (the XML element declarations for the features are not shown for simplicity):

```
<!ELEMENT % C1 (C0.a0 | C1.a1 | XMI.extension | C0.r0 | C1.r1 | C0.comp0 | C1.comp1)* >
<!ATTLIST C1
  a0 CDATA #IMPLIED
  a1 CDATA #IMPLIED
  r0 CDATA #IMPLIED
  r1 CDATA #IMPLIED
  %XML.element.att;
  %XML.link.att;
> 
```

Multiple inheritance is treated in such a way that the inherited attributes, references, and compositions that occur more than once in the inheritance hierarchy are only included once in their subclasses.

### 3.6.5 Attribute Specification

The representation of attributes of metamodel class “c” uses XML elements and XML attributes. If the metamodel attribute types are primitives or enumerations, XML elements are declared for them as well as XML attributes. The reasons for this encoding choice are several, including: the values to be exchanged may be very large values and unsuitable for XML attributes, and may have poor control of whitespace processing with options that apply only to element contents.

The declaration of each attribute named “a” with a non-enumerated type is as follows:

```
<!ELEMENT c.a (type specification) >
```

The type specification for an element may come from the metamodel or be defined outside the metamodel. In the former case the type specification is the name of the type; in the latter case it is considered to be a string type. If the data is a string type, then its type is mixed, and the specification must take the form:

```
<!ELEMENT c.a (#PCDATA| XMI.reference)* >
```

For attributes whose types are string types, an XML attribute must also be declared in the attribute list of the XML element corresponding to metamodel class “c;” the declaration of the XML attribute is as follows:

```
a CDATA #IMPLIED
```
An element is also declared to be of XML type string if the class contains a Tag XMIDataType with Value “string.”

When “a” is an attribute with enumerated values or Boolean values, a modified declaration is used to allow an XML processor to validate that the value of the attribute is one of the legal values of the enumeration. Attributes of this type are declared as follows:

```xml
<!ELEMENT a EMPTY >
<!ATTLIST c.a xmi.value (enum1 | enum2 | ...) #REQUIRED >
```

where `enum1`, `enum2`, … are replaced with an entry for each member of the enumeration set. An attribute whose type is boolean or an enumeration must also have an XML attribute declared in the XML element corresponding to metamodel class “c,” as follows:

```xml
a (enum1 | enum2 | ...) #IMPLIED
```

For example, if a class is named “c” with attributes “a1” and “a2,” where “a1” is a string type and “a2” is Boolean, the attributes are represented as follows:

```xml
<!ELEMENT a1 (#PCDATA | XMI.reference) *>
<!ELEMENT a2 EMPTY >
<!ATTLIST a2 xmi.value (true | false) #REQUIRED >
```

An element is also declared to be of XML type enumeration if the class contains a Tag XMIEnumSet where the Values are delimited by spaces.

In some MOF models, enumerations have a prefix substring that should be removed before placing the enumeration literals in the DTD. The Tag org.omg.xmi.enumerationUnprefix indicates the substring that should be stripped from the beginning of the enumeration literal when the literal begins with that prefix.

Default values for property and enumeration attributes may be specified in DTDs using the Tag org.omg.xmi.defaultValue attached to the attribute. The default value should be the XML string representation to be placed in the DTD. Default values for attributes should be specified in DTDs with care since XML allows the processor reading the document the option of not processing a DTD as an optional optimization. When tools skip processing the DTD, they do not obtain the default value of XML attributes. Instead, they would have to know the default value from understanding the metamodel. The form for specifying defaults, where “d” is the default, is:

```xml
a CDATA #IMPLIED d
```

For property model attributes as XML attributes:

```xml
a CDATA #IMPLIED d
```

For enumerated model attributes as XML attributes:
a (enum1 | enum2 | ...) #IMPLIED d

For enumerated model attributes as XML elements:

```xml
<!ELEMENT a EMPTY >
<!ATTLIST c.a xmi.value (enum1 | enum2 | …) #REQUIRED d >
```

Note – When reading documents with XML elements specifying model attribute values, be sure to use the value in the XML element rather than the default value from the unused XML attribute.

The multiplicities of metamodel attributes are not used in XMI 1.1 DTDs, so that the ordering of XML elements in XMI documents is not fixed.

### 3.6.6 Association Specification

Each association role is represented in an XML entity, an XML element, and an XML attribute. The multiplicity of the role is not used to create the declarations in XMI 1.2. The representation of an association role named “r” for a metamodel class “c” is:

```xml
<!ELEMENT r (content)* >
```

An XML attribute would be declared in the attribute list for the XML element corresponding to class c; the attribute declaration appears as follows:

```xml
r IDREFS #IMPLIED
```

The `content` is defined so that XML elements representing the classifier attached to the referenced `associationEnd` and any of its concrete subclasses may be included in XML element “r”. For example, if class `c1` is the classifier attached to the association end `r`, and it has three subclasses, `c2`, `c3`, and `c4`, and `c3` is abstract, the XML element `r` would be declared as follows:

```xml
<!ELEMENT r (c1 | c2 | c3 | c4)* >
```

The XML attribute `r` would be declared as follows:

```xml
r IDREFS #IMPLIED
```

The multiplicity in the XML element declaration is always “*” in XMI 1.2, rather than a mapping from the metamodel multiplicity to XML, as in XMI 1.0.

### 3.6.7 Containment Specification

Each association end that represents containment is also represented by an XML entity and an XML element. The content model of the XML element representing the association end is the XML element corresponding to the class, and the XML elements
corresponding to each of the subclasses of the class. If a class “c” is at the container end of an association link representing composition, and the other association end has role “r” for a class “c1” with concrete subclass “c2,” the representation in an XML DTD is as follows:

```xml
<!ELEMENT r (c1 | c2)* >
```

Note that the multiplicity is no longer used in the declarations in XMI 1.2.

### 3.7 Transmitting Incomplete Metadata

In XMI 1.2, multiplicities specified in the metamodel are no longer used when creating DTDs, so all DTDs support the interchange of model fragments. A DTD generator does not need to decide whether the DTD will support model fragments.

#### 3.7.1 Interchange of Model Fragments

In practice, most information is related. The ability to transfer a subset of known information is essential for practical information interchange. In addition, as information models are developed, they will frequently need to be interchanged before they are complete.

The following guidelines apply for interchanging incomplete models via XMI:

- Information may be missing from a model. The transmission format should not require the addition or invention of new information.
- Model fragments may be disjoint sets. Each set may be transmitted in the same XMI file at the XMI.content level or in different XMI files.
- “Incomplete” indicates a quantity of information less than or equal to “complete.” Additional information beyond that which the metamodel prescribes may be transmitted only via the extension mechanism.
- Semantic verification is performed on the metadata that is actually present just as if it was included in complete metadata.

#### 3.7.2 XMI Encoding

If you follow the rules for producing XMI 1.2 DTDs as described in the previous sections, you do not need to do anything else to enable the interchange of model fragments.

#### 3.7.3 Example

The following is an example of a UML model:

```xml
<UML:Model name="model1" xmi.id="id1">
```
3.8 Linking

The goal is to provide a mechanism for specifying references within and across documents. Although based on the upcoming XLinks standard, it is downwards compatible and does not require XLinks as a prerequisite.

3.8.1 Design Principles

- Links are based on XLinks to navigate to the document (which may be the current document) and XPointers to navigate to the element within the document.
- Links take the same form if the target is within the current or an external document.
- Link definitions are encapsulated in the entity XMI.link.att defined in Section 3.5.1, “Necessary XMI Attributes,” on page 3-6.
- Elements act as a union, where they are either a definition or a proxy. Proxies use the XMI.link.att to define the link, and contain no nested elements.
- XML.link.att supports external links through the XLink attributes, and internal links through the xmi.id and xmi.uuid attributes.
- Links are always to elements of the same type or subclasses of that type. Restricting proxies to reference the same element type reduces complexity, enhances reliability and type safety, and promotes caching. In XMI 1.1, subclasses are also allowed, to permit more flexibility in combining models and metamodels.
- When acting as a proxy, XML attributes may be defined, but not contents. The XML attributes act as a cache, which gives an indication if the link should be followed.
- Proxies may be chained.
- When following the link from a proxy, the definition of the proxy is replaced by the referenced element.
- It is efficient practice for maximizing caching and encapsulation to use local proxies of the same element within a document to link to a single proxy that holds an external reference.
- Association role elements typically contain proxies, which link to the definitions of the classes that participate in the association.
3.8.2 Linking

3.8.2.1 XLinks

When specifying an XLink, the “href” attribute may be used to specify an optional URI and XPointer that identify an XML element in another XML document. The href attribute must contain a locator for the model construct referred to. This model construct should be of the form URI “|” NAME, where URI locates the file that contains the model construct, and NAME is the value of the ID attribute of the referenced model construct. If the URI is not given, then NAME must be the value of an ID attribute in the current file. NAME is a shorthand for XPointer id(NAME). In elementary use, href could refer to another element id in the same XML file using href=”|id.”

When navigating into an XML document using an XPointer, the href=”XLink|XPointer” form for locating an element by xmi.id is: XLink + “|” + id. For example, href=”mydoc.xml|xxxx-yyyy...” The form for locating an element by xmi.label is: XLink + “|descendent(1,type,attribute,value)” where type is the expected element type or “#element” for any type, attribute is the name of the attribute, and value is the name of the attribute. For example, href=”|descendent(1,#element,xmi.label,class1)” XLink specifies the document to search and is the empty string when using the current document.

3.8.2.2 IDrefs

The xmi.idref attribute may be used to specify the XML ID of an XML document within the current XML document. Every construct that can be referred to has a local XML ID, a string that is locally unique within a single XML file. The XPointer part of a Reference uses the ID to find the construct. The XPointer specification also has relative addressing capabilities within a document that may be used. The choice of absolute ID-based addressing or relative addressing is made by the document creator on a per-reference basis.

3.8.2.3 UUIDrefs

As indicated previously, UUIDrefs are no longer used in XMI 1.1. The following description is how UUIDrefs were used in XMI 1.0.

The xmi.uuidref attribute is used for linking using absolute object identity. The UUID specified should correspond to the value of an xmi.uuid within the same document. Although there is no built-in support for UUIDs in XML at this time, it is envisioned that this support will be added in the near future. Linking by uuid results in the same action as the XPointer “|descendant(1,#element,xmi.uuid,DCE:abcd-efgh)”.

In XML there is currently no mechanism to enforce that the actual type of the XML element referred to is the desired one. Some tools might issue a warning if the type does not match the type of model construct actually referred to. This caching of expected information could be extended with other expected information attributes.
3.8.3 Example from UML

There is an association between ModelElements and Constraints in UML. Operations are a subclass of ModelElements. This example shows an association between Operations and four Constraints with roles `constraint` and `constrainedElement`. Qualified names have been suppressed for clarity. Each of the methods of linking is shown. The Constraints are shown in both definition and proxy form.

**Document 1**

```xml
<Operation xmi.id="idO1" xmi.label="op1" xmi.uuid="DCE:1234">
  <constraint>
    <Constraint xmi.id="idC1" xmi.label="co1" xmi.uuid="DCE:abcd">
      <body>First Constraint definition</body>
      <constrainedElement>
        <Operation xmi.idref="idO1" />
      </constrainedElement>
    </Constraint>
    <Constraint xmi.idref="idC2" />
    <Constraint xmi.idref="idC3" />
    <Constraint href="doc2.xml|idC4" />
  </constraint>
</Operation>

<Constraint xmi.id="idC2" xmi.label="co2" xmi.uuid="DCE:efgh">
  <body>Second Constraint definition</body>
  <constrainedElement>
    <Operation xmi.idref="idO1" />
  </constrainedElement>
</Constraint>

<Constraint xmi.id="idC3" xmi.label="co3" xmi.uuid="DCE:ijkl">
  <body>Third Constraint definition</body>
  <constrainedElement>
    <Operation href="|descendent(1,Operation,xmi.label,op1)"/>
  </constrainedElement>
</Constraint>

Document 2

<Constraint xmi.id="idC4" xmi.label="co4" xmi.uuid="DCE:mnop">
  <body>Fourth Constraint definition</body>
  <constrainedElement>
    <Operation href="doc1.xml|idO1"/>
  </constrainedElement>
</Constraint>
```

The first constraint is a definition. The `<constrainedElement>` role contains an Operation proxy, which has a local reference to the initial Operation definition using `xmi.idref`. The second constraint is a proxy referencing a constraint definition using the `xmi.idref` of “idC2.” The third constraint is a proxy reference to the definition...
using `xmi.idref` to the constraint “idC3.” The fourth constraint is an XLink and XPointer reference proxy to the definition of the constraint using the `href` to the file `doc2.xml` with id “idC4.”

Following the definition of the operation and its three constraint proxies are the definitions of two of the constraints. The second document contains the third constraint definition.

The use and placement of references is freely determined by the document creator. It is likely that most documents will make internal and external references for a number of reasons: to minimize the amount of duplicate declarations, to compartmentalize the size of the document streams, or to refer to useful information outside the scope of transmission. For example, the `href` of an XLink could contain a query to a repository that will recall additional related information. Or there may be a set of XMI documents created, one file per package to be transferred, where there are relationships between the packages.

### 3.8.4 XMI.reference

Any type of content can be allowed for the Reference XML element. This allows the receiver of the XML document to add additional processing to the content. For example, the content could be empty, contain an SQL query into a repository, a phone number, or a human readable version of the target’s name (useful in web browsers or any other convention desired.

XMI.reference can be used for values by pointing to large resources such as bitmaps outside of XML.

### 3.9 Transmitting Metadata Differences

The goal is to provide a mechanism for specifying the differences between documents so that an entire document does not need to be transmitted each time. This design does not specify an algorithm for computing the differences, just a form for transmitting them.

Up to now we have seen how to transmit a complete or full model. This way of working may not be adequate for all environments. More precisely, we could mention environments where there are many model changes that must be transmitted very quickly to other users. For these environments the full model transmission can be very resource consuming (time, network traffic, etc.) making it very difficult or even not viable for finding solutions for cooperative work.

The most viable way to solve this problem is to transmit only the model changes that occur. In this way different instances of a model can be maintained and synchronized more easily and economically. Concurrent work of a group of users becomes possible with a simple mechanism to synchronize models. Transmitting less information allows synchronizing models more efficiently.
3.9.1 Definitions

The idea is to transmit only the changes made to the model (differences between new and old model) together with the necessary information to be able to apply the changes to the old model.

A. New - Old = Difference

Model differencing is the comparison of two models and identifying the differences between them in a reversible fashion. The difference is expressed in terms of changes made to the old document to arrive at the new document.

B. New = Old + Difference

Model merging is the ability to combine difference information plus a common reference model to construct the appropriate new model.

3.9.2 Differences

Differences must be applied in the order defined. A later difference may refer to information added by a previous difference by linking to its contents. Model integrity requires that all the differences transmitted are applied. The following are the types of differences recognized, the information transmitted, and the changes they represent:

- Delete (reference to deleted element): The delete operation refers to a particular element of the old model and specifies a deep removal of the referenced element and all of its contents.
- Add (reference to containing element, new element, optional position): The add operation refers to a particular element of the old model and specifies a deep addition. The element and its contents are added. The contents of the new element are added at the optional position specified, the default being as the last element of the contents. The optional position form is based on XPointer’s position form.
  - 1 means the first position,
  - -1 means the last position, and
  - higher numbers count across the contents in the specified direction.
- Replace (reference to replaced element, replacement element, optional position): This operation deletes the old element but not its contents. The new element and its contents are added at the position of the old element. The original contents of the old element are then added to the contents of the new element at the optional position specified, the default being at the end.

3.9.3 XMI Encoding

The following are the elements used to encode the differences:
**XMI.difference**

The **XMI.difference** element is contained by the **XMI.content** section of the XMI document. There may be 0 or more **XMI.difference** elements and each **XMI.difference** element may contain 0 or more particular differences. The difference element optionally links to the original document to which the differences are applied.

**XMI.delete**

The **XMI.delete** element is contained by **XMI.difference**. Its link attributes contain a link to the element to be deleted.

**XMI.add**

The **XMI.add** element is contained by **XMI.difference**. The contents of **XMI.add** is the element to be added. The link attributes contain a link to the element to be deleted and an optional position element. The numbering corresponds to XPointer numbering, where 1 is the first and -1 is the last element.

**XMI.replace**

The **XMI.replace** element is contained by **XMI.difference**. The contents of **XMI.replace** is the element to replace the old element with. The attributes contain a link to the element to be replaced and an optional position element for the replacing element’s contents. The numbering corresponds to XPointer numbering, where 1 is the first and -1 is the last element.

### 3.9.4 Example

This example will delete a class and its attributes, add a second class, and rename a package. Fully qualified names are shortened for clarity.

**The original document**

```xml
<XMI.content>
    <Package xmi.id="ppp" xmi.label="p1">
        <Class xmi.id="ccc" xmi.label="c1">
            <ownedElement>
                <Attribute xmi.label="a1"/>
                <Attribute xmi.label="a2"/>
            </ownedElement>
        </Class>
    </Package>
</XMI.content>
```

**The differences document**

```xml
<XMI.content>
    <XMI.difference href="original.xml">
        <XMI.delete href="original.xml|ccc"/>
    </XMI.difference>
</XMI.content>
```
Here’s how the 3 differences change the document as they’re applied.

The **XML.delete**:

```
<XML.delete>
  <Package xmi.id="ppp" xmi.label="p1">
    <Class xmi.label="c2"/>
  </Package>
</XML.delete>
```

Next, the **XML.add**:

```
<XML.add>
  <Package xmi.id="ppp" xmi.label="p1">
    <Class xmi.label="c2"/>
  </Package>
</XML.add>
```

Finally, the **XML.replace**:

```
<XML.replace>
  <Package xmi.id="ppp" xmi.label="p2">
    <Class xmi.label="c2"/>
  </Package>
</XML.replace>
```

### 3.10 Document Exchange with Multiple Tools

This section contains a recommendation for an optional methodology that can be used when multiple tools interchange documents. In this methodology, the `xmi.uuid` and extensions are used together to preserve tool-specific information. In particular, tools may have particular requirements on their IDs, which makes ID interchange difficult. Extensions are used to hold tool-specific information, including tool-specific IDs.

The basic policy is that the XML ID is assigned by the tool that initially creates a construct. The **UUID** will most likely be the same as the ID the tool would chose for its own use. Any other modifiers of the document must preserve the original **UUID**, but may add their own as part of their extensions.
3.10.1 Definitions

3.10.1.1 General

- Extension - Extensions use the `XML.extension` element. Extensions to MCs may be nested in MCs, linked to the `XML.extensions` section(s) of the document, or linked outside the document. Each `XML.extension` contains a tool-specific identifier in the `xmi.extender` attribute. `XML.extensions` are considered private to a particular tool. An MC may have zero or more `XML.extensions`. `XML.extensions` may be nested.

3.10.1.2 IDs

- `xmi.uuid` - The universally unique ID of an MC, expressed as the `xmi.uuid` attribute. Example: `<Class xmi.uuid="ABCDEFGH">
- `xmi.extenderID` - The tool-specific ID of an MC. The `xmi.extenderID` is stored in an `XML.extension` of the MC when it differs from the `xmi.uuid`.

3.10.1.3 Tool ID policies:

Every tool is either Open or Closed.

- Open tool - A tool that will accept any `xmi.uuid` as its own. Open tools do not need to add extensions to contain a tool-specific id.
- Closed tool - A tool that will not accept an `xmi.uuid` created by another tool. Closed tools store their ids in the `xmi.extenderID` attribute of an `XML.extension`. The `xmi.extender` attribute of the `XML.extension` is set to the name of the closed tool.

3.10.2 Procedures

3.10.2.1 Document Creation

The Creating Tool writes a new XMI document. Each MC is assigned an `xmi.uuid`. If the `xmi.uuid` differs from the `xmi.extenderID`, an `XML.extension` for that tool is added containing the `xmi.extenderID`.

3.10.2.2 Document Import

The importing tool reads an existing XMI document. Extensions from other tools may be stored internally but not interpreted in the event a Modification will occur at a later time. One of the following cases occurs:

1. If the importing tool is an Open tool, the `xmi.uuid`s are accepted internally and no conversion is needed.
2. If the importing tool is a closed tool, the tool looks for a contained \texttt{XML.extension}
(identified by \texttt{xmi.extender}) with an \texttt{xmi.extenderID}. If one does not exist, the importing tool creates its own internal id.

3.10.2.3 Document Modification

- The modifying tool writes the MCs and any extensions preserved from import.
- For new MCs, the MC is assigned an \texttt{xmi.uuid}.
- Closed tools add an \texttt{XML.extension} including their internal id in the \texttt{xmi.extenderID}.

3.10.3 Example

This section describes a scenario in which Tool1 creates an XMI document that is imported by Tool2, then exported to Tool1, and then a third tool imports the document. All the tools are closed tools.

1. A model is created in Tool1 with one class and written in XMI.

```xml
<Class xmi.label="c1" xmi.uuid="abcdefgh">
</Class>
```

2. The class is imported into Tool2. Tool2 assigns \texttt{xmi.extenderID} “JKLMNOPQRST.” A second class is added with name “c2” and \texttt{xmi.extenderID} “X012345678.”

3. The model is merged back to XMI:

```xml
<Class xmi.label="c1" xmi.uuid="abcdefgh">
  <XML.extension xmi.extender="Tool2" xmi.extenderID="JKLMNOPQRST"/>
</Class>
<Class xmi.label="c2" xmi.uuid="X012345678">
</Class>
```

4. The model is imported into Tool1. Tool1 assigns \texttt{xmi.extenderID} “ijklmnop” to “c2” and a new class “c3” is created with \texttt{xmi.extenderID} “qrstuvwxyz.”

5. The model is merged back to XMI:

```xml
<Class xmi.label="c1" xmi.uuid="abcdefgh">
  <XML.extension xmi.extender="Tool2" xmi.extenderID="JKLMNOPQRST"/>
</Class>
<Class xmi.label="c2" xmi.uuid="X012345678">
  <XML.extension xmi.extender="Tool1" xmi.extenderID="ijklmnop"/>
</Class>
<Class xmi.label="c3" xmi.uuid="qrstuvwxyz">
</Class>
```
3.11 General Datatype Mechanism

The ability to support general data types in XMI has significant benefits. The applicability of XMI is significantly expanded since domain metamodels are likely to have a set of domain-specific data types. This general solution allows the user to provide a domain datatype metamodel with a defined mapping to the XML data types.

The domain metamodel is supplemented by adding the domain data type metamodel. These metamodels are connected by adding a relationship between the metamodels. For example, the Java data type String could be made a subclass of the UML class DataType, which is an instance of MOF DataType.

The datatypes are mapped into XML types. Each “primitive” element of the data type metamodel is mapped to an XML data type. Currently, XML supports two data types, string and enumeration. Future versions of XML are expected to support additional types. The mapping is accomplished by attaching a Tag-Value to the primitive data type.

The Tag XMIDataType indicates that this class is a datatype with XML mapping. If the XMIDataType Value is org.omg.xmi.string the XML string datatype is used. If the Value is org.omg.xmi.enumeration the XML enumeration type is used. The set of allowed values is provided by the Value of the XMIEnumSet Tag. The DTD declarations for these types are shown in Section 3.6.5, “Attribute Specification,” on page 3-17.

In UML, the DataType classes “String” and “Integer” have XMIDataType “string.” The class Boolean has XMIDataType “enum” and XMIEnumSet “true false.” The class VisibilityKind has XMIDataType “enum” and XMIEnumSet “public private protected.”

The convention for converting UML classes into MOF datatypes is the following: UML classes with a “primitive” stereotype become the String datatype. UML classes with the “enumeration” stereotype become the Enumeration datatype. The names of
attributes of the enumeration class become the names of the enumeration literals. If the type of the attribute in an enumeration class is another enumeration class, those enumerations are added to the set of literals recursively.
XML DTD Production

Contents

This chapter includes the following topics.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Purpose”</td>
<td>4-1</td>
</tr>
<tr>
<td>“DTD Syntax Approaches”</td>
<td>4-2</td>
</tr>
<tr>
<td>“Rule Set 1: Simple DTD”</td>
<td>4-4</td>
</tr>
<tr>
<td>“Fixed DTD Elements”</td>
<td>4-13</td>
</tr>
</tbody>
</table>

4.1 Purpose

This section describes the rules for creating a DTD from an MOF-based metamodel. Rule set 1 describes the grammar for an XMI DTD without using XML entities.

The DTDs defined by the rules in this section may be used to validate the XML text created by following the rules of the XML Document Production chapter. In XMI 1.2, these rule sets are stated formally in EBNF. The pseudo-code that was used to state these rules in XMI 1.0 remains for reference and explanatory value.

Conformance

The conformance rules are stated in Appendix E.
4

Notation for EBNF

The rule sets are stated in EBNF notation. Each rule is numbered for reference. Rule names are enclosed in angle brackets, for example <DTD>. Text within quotation marks are literal values, for example "<!ELEMENT." Text enclosed in double slashes represents a placeholder to be filled in with the appropriate external value, for example //Name of Attribute//. Literals should be enclosed in single or double quotation marks when used as the values for XML attributes in XML documents. The suffix "*" is used to indicate repetition of an item 0 or more times. The suffix "?" is used to indicate repetition of an item 0 or 1 times. The suffix "+" is used to indicate repetition of item 1 or more times. The vertical bar "|" indicates a choice between two items. Parentheses "()" are used for grouping items together.

EBNF ignores white space; hence these rules do not specify white space treatment. However, since white space in XML is significant, the actual DTD generation process must insert white space at the appropriate points.

The XML element names generated using these rules are qualified names. A qualified name consists of an optional namespace name and colon "::" and a Class, Package, or Association name. Attributes or References are further prefixed by a period ("." ) delimiter. See Section 3.6.1, “Namespace Qualified XML Element Names,” on page 3-14.

4.2 DTD Syntax Approaches

This section describes various approaches for creating XMI DTDs using XML entity definitions. The following section contains the actual DTD syntax produced by the simplest of these approaches.

As indicated in the conformance chapter, the key conformance point for XMI is the actual document rather than the DTD. It is, after all, the XMI documents themselves that are interchanged among the various parties, not the DTD(s) used to produce them.

The XML entity construct, acting as a “macro” facility enables the DTD producer to create a number of DTDs that is factorial in the size of the underlying metamodel, each of which would produce the same XMI stream. Thus, rather than specify a DTD production method, this standard simply offers some possible general approaches and describes the advantages and disadvantages of each.

4.2.1 DTD without entities

An XMI DTD need not use entities at all. In this approach, DTD elements are generated for every package and class, along with associations that have no references. The package DTD elements list every possible class element that can be in the package. The class DTD elements list each attribute and reference for the class (including those of all of its superclasses). Composite and non-composite references have different formats in XMI and hence different DTD element definitions.
This approach has the advantages of simplicity and of a clear mapping between the metamodel constructs and the XML elements used to represent them. It has the disadvantages of being extremely verbose, as the same information is repeated over and over in the various DTD definitions, and of low maintainability, since each of these repeated constructs must be changed when the underlying item in the metamodel is changed.

For example, a DTD for UML 1.3 produced by this approach has almost 100 occurrences of the “name” attribute of the “ModelElement” class. While it is clear from the DTD just where the “name” attribute is used, if the attribute were to be changed to “elementname” in the UML 1.3 metamodel, it would be troublesome to change in the DTD.

4.2.2 DTD with class-level entities

The large number of repetitions of class elements, such as “ModelElement.name” that occur in the first DTD production approach is due to the fact that inheritance often plays an important part in MOF metamodel definition. Thus, an attribute of a class near the top of the inheritance tree becomes an attribute of each of its subclasses.

An approach to DTD production that removes considerable duplication is one that encapsulates the attributes and references of a particular class into an entity definition. In this case, when the DTD element for a class is defined, the entities for its various (non-empty) superclasses are included, along with the entity(s) for the class itself.

This approach has two variants. In one case a single entity defines both the attributes and references of a class. In the other, separate entities are used for the attributes, the non-composite references and the composite references. The former variant produces the more compact DTD, though the individual entity definitions are more complicated than in the latter. The latter approach is more likely to result in empty entities.

The advantages of this approach are the compact DTDs that can be produced and improved maintainability. The disadvantage is primarily one of computational complexity. In particular, since XML does not allow empty entity definitions, the DTD producer must beware not to produce them when a class has no attributes or references and to avoid invoking them in subclasses.

For example, a DTD for UML 1.3 produced by this approach would have 5 (in the first variant) or approximately 8 (in the second) entity invocations in the DTD element for the Class class, instead of 39 individual element definitions.

4.2.3 DTD with multi-level entities

An approach that can produce an even more compact DTD than the class-level approach is one that encompasses ALL of the superclass attributes and references in a single entity definition. That is, the DTD element definition for a class would contain an invocation of only one entity, which contained the attributes and references of the class itself and an entity invocation of its immediate superclass. This forms a chain of entity definition and invocations going up to the topmost class in the inheritance hierarchy ending at the class in question.
Like the second approach, this method has two variants—one that creates a single entity for both the attributes and references of the parent class and one that creates separate entities for each. Compactness and complexity considerations for the variants are basically the same as in the second approach.

The advantages and disadvantages of this approach depend on the metamodel being used. For a metamodel with only single inheritance, the advantages are that the DTD produced by this approach is potentially the most compact and maintainable, since there are at most three entities created per class, and they contain information only from that class. The disadvantage is that the computational complexity is greater than the class-level approach, simply because the entities involved reference each other, requiring the DTD producer to be aware of the inheritance hierarchy.

When multiple inheritance is used, the advantages are reduced and the disadvantage is increased. In order to avoid duplication of elements in the expanded DTD, only entities from one branch of an inheritance tree can be used; attributes and references from the other branch must be explicitly listed, as in the entity-less approach. This results in a less-compact DTD. Likewise, the computational complexity is considerably increased, since the DTD production algorithm must take into account which side of the inheritance tree is being used and produce the proper explicit attribute and reference elements as well as entity definitions.

For example the UML 1.3, the DTD produced by this method is actually somewhat larger than that produced by the class-level entity approach. The reason is the multiple inheritances used, particular the Namespace/GeneralizableElement complex, which occurs high in the inheritance tree, meaning that almost every class definition includes additional attributes and references.

### 4.3 Rule Set 1: Simple DTD

#### 4.3.1 EBNF

The EBNF for rule set 1 is listed below with rule descriptions between sections:
4.3.1.1 Rule Set 1

1. `<DTD>` ::= `<1b:FixedContent>`
   `<1d:XMIAttList>?`
   `<2:PackageDTD>+

1a. `<XMIFixedAttribs>` ::= "%XMI.element.att;" "%XMI.link.att;"
1b. `<FixedDeclarations>` ::= //Fixed declarations//
1c. `<Namespace>` ::= ( //Name of namespace// " ::" )?
1d. `<XMIAttList>` ::= "<!ATTLIST" "XMI" ("xmlns:" //Name of namespace// "CDATA" "#IMPLIED")+
   ">"

Description:

1. A DTD consists of a set of fixed declarations plus declarations for the namespaces
   and contents of the Packages in the metamodel.

1a. The fixed XMI attributes present on the major elements provide element identity
   and element linking.

1b. The fixed declarations are listed in Section 4.4, “Fixed DTD Elements,” on
   page 4-13.

1c. A namespace is a namespace name followed by a ":". If no namespace name is
   given, the rule is a blank.

1d. The XMI element attribute declaration for the namespace, if used.

4.3.1.2 Rule Set 2

2. `<PackageDTD>` ::= { `<2:PackageDTD>
   | `<3:ClassDTD>
   | `<4:AttributeElmtDef>
   | `<7:CompositionDTD>
   | `<10:AssociationDTD> }*
   <9:PackageElementDef>

Description:

2. The DTD contribution from a Package consists of the declarations for any contained
   Packages, Classes, classifier level Attributes, containment aggregations,
   Associations without References, and an XML element definition for the Package
   itself.
4.3.1.3  Rule Set 3

3. \(<\text{ClassDTD}> \ ::= \ (<4:AttributeElmtDef> | <5:ReferenceElmtDef>)^* <6:ClassElementDef>\)

\textit{Description:}

3. The class DTD contribution consists of the element definitions for any Attributes and References of the Class and an element definition for the Class itself.

4.3.1.4  Rule Set 4

4. \(<\text{AttributeElmtDef}> \ ::= "<\text{!ELEMENT} <4a:AttribElmtName>\)
\n\begin{align*}
4c. \ & \ <4c:AttribContents> \\
\end{align*}
\begin{align*}
\{ "<\text{ATTLIST} <4a:AttribElmtName> \\
\ & \ "xmi.value" "(" <4g:AttribEnumList> ")" \\
\ & \ "\text{#REQUIRED}" "">" \\)
\begin{align*}
4d. \ & \ )? \end{align*}
\end{align*}
\begin{align*}
4a. \ & \ <\text{AttribElmtName}> \ ::= <6a:ClassElmtName> "." <4b:AttribName> \end{align*}
\begin{align*}
4b. \ & \ <\text{AttribName}> \ ::= \text{//Name of Attribute//} \end{align*}
\begin{align*}
4c. \ & \ <\text{AttribContents}> \ ::= <4d:AttribData> \end{align*}
\begin{align*}
\ & \ | <4e:AttribEnum> \\
\ & \ | <4f:AttribClasses> \end{align*}
\begin{align*}
4d. \ & \ <\text{AttribData}> \ ::= "(" "\text{#PCDATA}" "|" "\text{XMI.reference}" ")"*" \end{align*}
\begin{align*}
4e. \ & \ <\text{AttribEnum}> \ ::= "\text{EMPTY}" \end{align*}
\begin{align*}
4f. \ & \ <\text{AttribClasses}> \ ::= "(" <6a:ClassElmtName> \\
\ & \ | " <6a:ClassElmtName> \\)
\begin{align*}
\ & \ )*" \end{align*}
\begin{align*}
4g. \ & \ <\text{AttribEnumList}> \ ::= <4h:AttribEnum> ( "|" <4h:AttribEnum> )* \end{align*}
\begin{align*}
4h. \ & \ <\text{AttribEnum}> \ ::= \text{//Name of Enumeration Literal//} \end{align*}

\textit{Description:}

4. These rules define the declaration of an Attribute of the Classes of the metamodel as the content of an XML element. These metamodel Attributes can, in some cases, be expressed as XML attributes rather than element content. This is further specified in rule 6h and gives the document writer the ability to choose which representation is most convenient in a particular use in an XML document.

4a. 4b. The name of the XML element representing an Attribute of a Class is the element name of the Class containing the Attribute followed by a dot separator and the name of the Attribute.
An Attribute that can be expressed as a data value is expressed in terms of a string or reference to its content (4d) or an enumeration (4e, 4g, 4h). An Attribute that has a Class as its value is expressed in terms of the possible Class types that can be instances of its value (4f). If the Class has subclasses, the element name of each of its subclasses is included in the declaration.

Note – If the MOF Tag “org.omg.xmi.enumerationUnprefix” is attached to the DataType where the enumerated values of the Attribute are defined, the value of this Tag contains a prefix that will be removed from the values of enumeration literals before they are written in the DTD.

Although the DTD as produced by this grammar cannot restrict the interspersing of other Attribute values among the instances of the values of a multi-valued Attribute, the XML document production rules state that all values for the Attribute should be consecutive elements and not interspersed with other Attribute values.

4.3.1.5 Rule Set 5

5. <ReferenceElmtDef> ::= "<!ELEMENT" <5a:ReferenceElmtName>
                              <5c:RefContents> ">
5a. <ReferenceElmtName> ::= <6a:ClassElmtName> "." <5b:ReferenceName>
5b. <ReferenceName> ::= //Name of Reference//
5c. <RefContents> ::= "(" <6a:ClassElmtName>
                     ( "|" <6a:ClassElmtName> )* "")"*"

Description:

5. These rules define the declaration of a metamodel Reference as XML element content for linking by proxy. It is also possible to place the Reference in the attribute list of the XML element, as defined in rule 6i. This provides the ability to more conveniently represent References when the limited linking facilities available in such a case are sufficient.

5a. 5b. The name of the XML element representing a Reference is the element name of the Class containing the Reference, a dot separator, and the name of the Reference.

5c. The element name of the type of the Reference is given in the declaration. Any subclass of the type can, but need not, appear in the declaration as well. An XML linkage to a Class element will work if the target of the linkage is a member of a Class or one of its subclasses.
4.3.1.6 Rule Set 6

6.  `<ClassElementDef>` ::= "<!ELEMENT" `<6a:ClassElmtName>`
                  `<6b:ClassContents> ">
                  "<!ATTLIST" `<6a:ClassElmtName>`
                  `<6g:ClassAttListItems> ">

6a. `<ClassElmtName>` ::= `<1c:Namespace> //Name of Class//`

6b. `<ClassContents>` ::= "(" `<6d:ClassAttributes>` ?
                  `<6e:ClassReferences>` ?
                  `<6f:ClassCompositions>` ?
                  `<6c:Extension> ")" "">"

6c. `<Extension>` ::= "XML.extension"

6d. `<ClassAttributes>` ::= `<4a:AttribElmtName>`
                  ( "" `<4a:AttribElmtName>`) * "|

6e. `<ClassReferences>` ::= `<5a:ReferenceElmtName>`
                  ( "" `<5a:ReferenceElmtName>` ) * "|

6f. `<ClassCompositions>` ::= `<6a:ClassElmtName>`
                  ( "" `<6a:ClassElmtName>` ) * "|

6g. `<ClassAttListItems>` ::= `<6h:ClassAttribAtts> <1a:XMIFixedAttribs>`

6h. `<ClassAttribAtts>` ::= ( `<6i:ClassAttribRef>
                  | `<6j:ClassAttribData>`
                  | `<6k:ClassAttribEnum>` ) *

6i. `<ClassAttribRef>` ::= `<4b:AttribName> "IDREFS" "#IMPLIED`

6j. `<ClassAttribData>` ::= `<4b:AttribName> "CDATA" "#IMPLIED`

6k. `<ClassAttribEnum>` ::= `<4b:AttribName>`
                  ( "" `<4g:AttribEnumList> ")" "#IMPLIED`

6l. `<ClassAttribDflt>` ::= //Default value/

Description:

6. These rules describe the declaration of a Class in the metamodel as an XML element with XML attributes.

6a. The name of the XML element for the Class is name of the Class prefixed by the namespace, if present. When using nested packages, it is possible for the class names to conflict; if class names do conflict, you must assign namespaces to the appropriate packages and use namespace prefixes to make the class element names unique.

6b. 6c. The XML element for the Class contains XML elements for the contained non-derived Attributes, References and Compositions of the Class, plus an extension element, which can refer to a locally-defined subclass of this Class in the XMI.extensions section of the XML document.
6d. The XML element name for each non-derived Attribute of the Class is listed as part of the content model of the Class element. This includes the Attributes defined for the Class itself as well as all of the non-derived Attributes inherited from superclasses of the Class.

6e. The XML element name for each non-composite Reference of the Class is listed in the content model of the Class. A non-composite Reference is one where the aggregation of the exposedEnd of the Reference is not composite. The list includes the References defined for the Class itself, as well as all References inherited from the superclasses of the Class.

6f. The XML element name for each Class contained in this Class in the content model of the Class element. Here, containment means that the contained Class is either directly owned as an ownedElement of the Class or it is the type of a Reference of the Class, the aggregation of whose exposedEnd is composite. In addition to the element name of the contained Class, the element name of each subclass of the contained Class must also be listed.

6g. In addition to the standard identification and linkage attributes, the attribute list of the Class element can contain XML attributes for the Attributes and non-composite References of the Class, when the limited facilities of the XML attribute syntax allow expression of the necessary values.

6i. References (either directly owned by the Class or inherited) can be expressed as XML id reference XML attributes.

6j. Single-valued Attributes (direct or inherited) of a Class that have a string representation for their data are mapped to CDATA XML attributes. Multi-valued Attributes of a Class cannot be so expressed, since the XML attribute syntax does not allow repetition of values.

6l. If an Attribute is expressed as an XML attribute, its default value may be expressed in the DTD if there is an MOF Tag "org.omg.xmi.defaultValue" attached to the Attribute. The value of this Tag must be expressible as an XML attribute string.

### 4.3.1.7 Rule Set 7

7. \(<\text{CompositionDTD}> \quad ::= \quad <\text{CompositionElmtDef}>^*\)

**Description:**

7. Elements for Associations that represent compositions are described using rule 8.
4.3.1.8  Rule Set 8

8.  <CompositionElmtDef ::= "<!ELEMENT" <8a:RoleElmtName> 
    "(" <6f:ClassCompositions> ")"">"
8a. <RoleElmtName> ::= <6a:ClassElmtName> "." <8b:RoleName>
8b. <RoleName> ::= //Name of Role//

Description:
8. The composition element is generated for each Reference in the Package, which has 
an exposedEnd whose aggregation is composite. This element is used in the class 
contents XML element (6). It is a list of the Class that is the type of the Reference, 
as well as all of its subclasses.

8a. 8b. The name of the Reference XML element is the element name of the Class 
containing the Reference, followed by a dot and the name of the Reference.

4.3.1.9  Rule Set 9

9.  <PackageElementDef ::= "<!ELEMENT" <9a:PkgElmtName> <9c:PkgContents> "">
    "<!ATTLIST" <9a:PkgElmtName>
    <9h:PkgAttListItems>">
9a. <PkgElmtName> ::= <1c:Namespace> <9b:PkgName>
9b. <PkgName> ::= //Name of Package//
9c. <PkgContents> ::= "(" <9d:PkgAttributes> ?
    <9e:PkgClasses> ?
    <9f:PkgAssociations> ?
    <9g:PkgPackages> ?
    <6c:Extension> ")"">"
9d. <PkgAttributes> ::= <4a:AttribElmtName> ( "|" <4a:AttribElmtName>* "|" )
9e. <PkgClasses> ::= <6a:ClassElmtName> ( "|" <6a:ClassElmtName>* "|" )
9f. <PkgAssociations> ::= <12a:AssnElmtName> ( "|" <12a:AssnElmtName>* "|" )
9g. <PkgPackages> ::= <9b:PkgElmtName> ( "|" <9b:PkgElmtName>* "|" )
9h. <PkgAttListItems> ::= <9i:PkgAttribAtts> <1a:XMIFixedAttribs>
9i. <PkgAttribAtts> ::= <6h:ClassAttribAtts>

Description:
9. The DTD contribution from the Package consists of an XML element definition for 
the Package, with a content model specifying the contents of the Package.
9a. 9b. The name of the Package XML element.

9c. The Package contents consists of any classifier level Attributes, Associations without References, Classes, nested Packages and an extension reference.

9d. Classifier level Attributes of a Package are also known as static attributes. Such Attributes inherited from Packages from which this Package is derived are also included.

9e. Each Class (including subtypes) in the Package is listed. All classes corresponding to instances that can be in extents directly or indirectly within the package extent are listed.

9f. It is possible that the Package contains Associations that have no References (i.e., no Class contains a Reference which refers to an AssociationEnd owned by the Association). Every such Association contained in the Package or Package from which the Package is derived is listed as part of the Package contents in order that its information can be transmitted as part of the XML document.

9g. Nested Packages are listed. Nested Packages included in Packages from which this Package is derived are also included.

Note – When you inherit a package, you must include everything from that package.

9h. 9i. XML attributes for classifier level Attributes are generated following the same rules as those for instance level Attributes. The fixed identity and linking XML attributes are included.

### 4.3.1.10 Rule Set 10

10. `<AssociationDTD>` ::= `<11:AssociationEndDef>`
    `<11:AssociationEndDef>`
    `<12:AssociationDef>`

**Description:**

10. The XML elements for unreferenced Associations consist of definitions for its AssociationEnds and for the Association itself.
4.3.1.11  Rule Set 11

11. <AssociationEndDef> ::= "<!ELEMENT" <11a:AssocElmtName> "EMPTY" ">
   "<!ATTLIST" <11a:AssocElmtName>
   <11c:AssocEndAtts> ">
11a.<AssocEndElmtName> ::= <12a:AssnElmtName> "." <11b:AssocEndName>
11b.<AssocEndName> ::= //Name of AssociationEnd//
11c.<AssocEndAtts> ::= <1a:XMIFixedAttribs>

Description:
11. The declaration for an AssociationEnd XML element has no content model, though it has the standard set of XML attributes.
   11a.11b. The name of the AssociationEnd XML element is the element name of the association containing the AssociationEnd, a dot separator, and the name of the AssociationEnd.
   11c. The fixed identity and linking XML attributes are the AssociationEnd’s only XML attributes.

4.3.1.12  Rule Set 12

12. <AssociationDef> ::= "<!ELEMENT" <12a:AssnElmtName>
   <12c:AssnContents> ">
   "<!ATTLIST" <12a:AssnElmtName> <12d:AssnAtts> ">
12a.<AssnElmtName> ::= <1c:Namespace> <12b:AssnName>
12b.<AssnName> ::= //Name of Association//
12c.<AssnContents> ::= "(" <11a:AssocEndElmtName> "|"
   <11a:AssocEndElmtName> "|"
   <6c:Extension> ")"*
12d.<AssnAtts> ::= <1a:XMIFixedAttribs>

Description:
12, 12c. The declaration of an unreferenced Association consists of the names of its AssociationEnd XML elements.
   12a.12b. The name of the XML element representing the Association.
   12d. The fixed identity and linking XML attributes are the Association XML attributes.
4.4 Fixed DTD Elements

There are some elements of the DTD that are fixed, constituting a form of “boilerplate” necessary for every MOF DTD. These elements are described in this section. They should be included at the beginning of the generated DTD. Though, as elements, these need not be at the beginning of the DTD, the convention is to place them there.

The use of these fixed content elements means that any DOCTYPE declaration in an XMI-conformant transfer text should reference “XMI” as its root element. The “XMI” element includes the “XMI.content” element, which contains the actual transferred data. The content model of “XMI.content” then allows the transferred data to have any element as its effective root element.

Only the DTD content of the fixed elements is given here. For a complete description of the semantics of these elements, Section 3.6, “Metamodel Class Specification,” on page 3-14.

The FixedContent elements are:

```xml
<!DOCTYPE XMI SYSTEM "omgschema.dtd" PUBLIC "-//OMG//DTD XMI 1.2//EN" "http://www.omg.org/standards/OMG_DTD/xmi-1.2.dtd">
<!ELEMENT XMI (XMI.header?, XMI.content?, XMI.difference*, XMI.extensions*)>
<!ATTLIST XMI
  xmi.version CDATA #FIXED "1.2"
  timestamp CDATA #IMPLIED
  verified (true | false) #IMPLIED>
```

```
<!ELEMENT XMI.header (XMI.documentation?, XMI.model*, XMI.metamodel*, XMI.metametamodel*, XMI.import*)>
```

```
<!ELEMENT XMI.documentation (#PCDATA | XMI.owner | XMI.contact | XMI.longDescription | XMI.shortDescription | XMI.exporter | XMI.exporterVersion | XMI.notice)*>
```

```
<!ELEMENT XMI.owner ANY>
```
<!ELEMENT XMI.contact ANY >
<!ELEMENT XMI.longDescription ANY >
<!ELEMENT XMI.shortDescription ANY >
<!ELEMENT XMI.exporter ANY >
<!ELEMENT XMI.exporterVersion ANY >
<!ELEMENT XMI.exporterID ANY >
<!ELEMENT XMI.notice ANY >

<!-- _______________________________________________________________ -->
<!-- -->
<!-- XMI.element.att defines the attributes that each XML element -->
<!-- that corresponds to a metamodel class must have to conform to -->
<!-- the XMI specification. -->
<!-- _______________________________________________________________ -->
<!ENTITY % XMI.element.att
'xmi.id ID #IMPLIED 
xmi.label CDATA #IMPLIED 
xmi.uuid CDATA #IMPLIED ' >

<!-- _______________________________________________________________ -->
<!-- -->
<!-- XMI.link.att defines the attributes that each XML element that -->
<!-- corresponds to a metamodel class must have to enable it to -->
<!-- function as a simple XLink as well as refer to model -->
<!-- constructs within the same XMI file. -->
<!-- _______________________________________________________________ -->
<!ENTITY % XMI.link.att
'href CDATA #IMPLIED 
xmi.idref IDREF #IMPLIED ' >

<!-- _______________________________________________________________ -->
<!-- -->
<!-- XMI.model identifies the model(s) being transferred -->
<!-- _______________________________________________________________ -->
<!ELEMENT XMI.model ANY >
<!ATTLIST XMI.model
%XMI.link.att;
xmi.name CDATA #REQUIRED 
xmi.version CDATA #IMPLIED >

<!-- _______________________________________________________________ -->
<!-- -->
<!-- XMI.metamodel identifies the metamodel(s) for the transferred -->
<!-- data -->
<!ELEMENT XMI.metamodel ANY >
<!ATTLIST XMI.metamodel
%XMI.link.att;
xmi.name CDATA #REQUIRED
xmi.version CDATA #IMPLIED >

<!-- XMI.metametamodel identifies the metametamodel(s) for the transferred data -->
<!ELEMENT XMI.metametamodel ANY >
<!ATTLIST XMI.metametamodel
%XMI.link.att;
xmi.name CDATA #REQUIRED
xmi.version CDATA #IMPLIED >

<!-- XMI.import identifies imported metamodel(s) -->
<!ELEMENT XMI.import ANY >
<!ATTLIST XMI.import
%XMI.link.att;
xmi.name CDATA #REQUIRED
xmi.version CDATA #IMPLIED >

<!-- XMI.content is the actual data being transferred -->
<!ELEMENT XMI.content ANY >

<!-- XMI.extensions contains data to transfer that does not conform to the metamodel(s) in the header -->
<!ELEMENT XMI.extensions ANY >
<!ATTLIST XMI.extensions
xmi.extender CDATA #REQUIRED >
<!-- extension contains information related to a specific model -->
<!-- construct that is not defined in the metamodel(s) in the -->
<!-- header -->

<!ELEMENT XMI.extension ANY >
<!ATTLIST XMI.extension
  %XMI.element.att;
  %XMI.link.att;
  xmi.extender CDATA #REQUIRED
  xmi.extenderID CDATA #IMPLIED >

<!-- XMI.difference holds XML elements representing differences to -->
<!-- a base model -->

<!ELEMENT XMI.difference (XMI.difference | XMI.delete | XMI.add | XMI.replace)* >
<!ATTLIST XMI.difference
  %XMI.element.att;
  %XMI.link.att; >

<!-- XMI.delete represents a deletion from a base model -->

<!ELEMENT XMI.delete EMPTY >
<!ATTLIST XMI.delete
  %XMI.element.att;
  %XMI.link.att; >

<!-- XMI.add represents an addition to a base model -->

<!ELEMENT XMI.add ANY >
<!ATTLIST XMI.add
  %XMI.element.att;
  %XMI.link.att;
  xmi.position CDATA "-1" >
<!-- XMI.replace represents the replacement of a model construct with another model construct in a base model -->
<!ELEMENT XMI.replace ANY >
<!ATTLIST XMI.replace %XMI.element.att; %XMI.link.att; xmi.position CDATA "-1" >

<!-- XMI.reference may be used to refer to data types not defined in the metamodel -->
<!ELEMENT XMI.reference ANY >
<!ATTLIST XMI.reference %XMI.link.att; >

The following fixed DTD declarations are used only when required by the metamodel.

<!-- This section contains the declaration of XML elements representing data types -->
<!ELEMENT XMI.field ANY >
<!ELEMENT XMI.seqItem ANY >
<!ELEMENT XMI.sequence (XMI.seqItem)* >
<!ELEMENT XMI.enum EMPTY >
<!ATTLIST XMI.enum xmi.value CDATA #REQUIRED >
XML Generation Principles

Contents

This chapter includes the following topics.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Purpose”</td>
<td>5-1</td>
</tr>
<tr>
<td>“Introduction”</td>
<td>5-1</td>
</tr>
<tr>
<td>“Two Model Sources”</td>
<td>5-2</td>
</tr>
<tr>
<td>“Distinctions between Approaches”</td>
<td>5-12</td>
</tr>
</tbody>
</table>

5.1 Purpose

This section describes the manner in which XML Documents are generated to represent models. The subsequent section specifies the specific rules that XMI uses in this generation process.

5.2 Introduction

XMI defines the manner in which a model will be represented as an XML document. For a given model, each XMI-conforming implementation will produce an equivalent XML document.

XML document production is defined as a set of rules, which when applied to a model or model elements, produce an XML document. These rules can be applied to any model whose metamodel can be described by the Meta Object Facility (MOF). This section provides an informal description of the production of XML documents from models. Although it may appear from this description that XML production should be performed using certain algorithms, interfaces, or facilities, any implementation that
produces XML equivalent to the XML produced by the application of the specified production rules complies with XMI. The specific rules, and the specification of XML document equivalence is provided in Chapter 6, *XML Document Production*.

### 5.3 Two Model Sources

![SimpleGraph](image)

*Figure 5-1*  A very simple metamodel for graph modeling

XMI can be applied to any model whose metamodel can be described by the MOF. However, the MOF meta-metamodel does not require any specific construct or mechanism to be used to define, in a metamodel, what will constitute a model. This approach allows metamodelers greatest flexibility. XMI is not able to identify, for any metamodel, what will constitute a model. Therefore XMI, to provide greater flexibility in exchanging model information, provides two distinct methods of specifying the modeling elements that are used to generate an XML document.

#### 5.3.1 Production by Object Containment

Most metamodels are characterized by a composition hierarchy. Modeling elements of some type are composed of other modeling elements. In UML, for example, a Model is composed of Classes, UseCases, Packages, etc. Those elements in turn are composed of other elements. This composition is defined in metamodels using the MOF’s
composite form of Association. This composition must obey strict containment – an element cannot be contained in multiple compositions. To support models and model fragments as compositions, XMI provides for XML document production by object containment. Given a composite object, XMI’s rules define the XML document, which represents the composite object and all the contained objects in the composition hierarchy of which it is the root.

Consider a simple example. A very simple metamodel defines a language or set of constructs for developing graphs. The modeling elements Net, Node, Arc, and Token, and a supporting data type are defined. Figure 5-1 on page 5-2 shows this metamodel in UML notation. The metamodel is defined using the MOF Model. The MOF Model instances which compose the SimpleGraph metamodel are shown in Figure 5-2 (with much detail omitted).

*Figure 5-2*  Object diagram showing simple metamodel as an instance of the MOF Model
Since this metamodel is expressed via the MOF, its model instances can be represented in XML using the XMI generation rules. A simple model is shown in some net notation in Figure 5-3.

![Figure 5-3 Example Net as a model of the SimpleGraph metamodel](image)

As instances of the metamodel elements, the same model would form the object diagram in Figure 5-4.

![Figure 5-4 Objects forming the example SimpleGraph model](image)

The XML production rules for Production by Object Containment are applied to a single root object of a composition. In this example, the rules are applied to the Net instance, to form the XML document representing this model. The rules are applied throughout the composition hierarchy by navigating through the composition links. In addition, the rules make use of the model’s metamodel to represent the types of the values.
Each generated XML document begins with a prologue and the standard enclosing XML element’s start tag. Next comes the actual model, starting with the root object. For each object, including this root object, the element start tag is generated from the object’s metaclass name. In this example, it is:

```xml
<SimpleGraph.Net xmi.id='a1'>
```

The element attribute `xmi.id` provides a unique identifier with the document for this element.

Note that all names in XMI are fully qualified, based on the MOF description of their metamodel. The name of the item is formed by the sequence of containments and compositions, starting at the outermost package of the metamodel and separated by dots.

Next each attribute of the current object is used to generate XML. The attribute is enclosed in an element, defined by the name of the attribute, as found in the metamodel:

```xml
<SimpleGraph.Net.created>
```

Next the attribute value is written out as XML. In the example, the attribute is of type `DateTimeType`, as defined in the metamodel. `DateTimeType` is a struct with two fields, time, of type long, and timezone, of type string. The representation of struct values uses field tags as delimiters:

```xml
<XML.field>1873852</XML.field>
<XML.field>GMT</XML.field>
```

Then the attribute is completed with the corresponding end tag:

```xml
</SimpleGraph.Net.created>
```

Were there other attributes of the Net object, they would follow in a similar manner. These are followed by the Net object’s references.

The MOF supports the use of `References` in defining metamodels. A reference provides the object’s navigability to linked objects. Following the attributes of an object, each of its references are written. XMI considers references to be of two different types and treats them differently.

An object linked to another via a link defined in the metamodel as having an aggregation other than composite is considered to be a `normal` reference. On the other hand, if an object is linked to another object via a link defined in the metamodel as a composite association, with the composite end corresponding to link end of the composite object, then the reference used is a `composite` reference.

In XMI, all of the normal references of an object are written, followed by all of the composite references. In XMI, this composition is indicated by XML element containment.

In this example, there is a total of three Node objects and three Token objects contained by the Net object using composite references. The “nodes” reference will be expressed as:
<SimpleGraph.Net.nodes>
to indicate the Node objects it contains through the “nodes” reference. Then, for each
Node, the process of producing XML to represent an object is repeated. For the
example, the Node with the name NodeA is written out in XML, starting with the
element start tag:

<SimpleGraph.Node xmi.id='a2'>

the value of the attribute id of the XML element can be any unique value which is
XML-compliant. Just as before, all the attribute values are written out first. The node
class defines the attribute “name;” for this Node instance, the XML is:

<SimpleGraph.Node.name>NodeA</SimpleGraph.Node.name>

Next the normal (i.e., non-composite, non-component) references are written out.
These are the references defined by Associations that are not defined as composites at
either end. Since the Node class defines the Reference “marker;” and NodeA has
markers, the XML generated is:

<SimpleGraph.Node.marker>
  <SimpleGraph.Token xmi.idref='a5' />  
  <SimpleGraph.Token xmi.idref='a6' />  
</SimpleGraph.Node.marker>

Since this is a normal, rather than a composite, reference, the Token objects are not
written at this point. Rather, a reference is used to point within the document to the
elements that actually define the objects. A complete set of linking attributes is defined
in XMI; the xmi.idref could, for example, be replaced by an href to element definitions
in another location. See Section 3.5.1, “Necessary XMI Attributes;” on page 3-6, for a
discussion on linking attributes.

Next, the value of the Node’s "targetNodes" reference is written out as XML:

<SimpleGraph.Node.targetNodes>
  <SimpleGraph.Node xmi.idref='a3' />  
</SimpleGraph.Node.targetNodes>

This example illustrates the fact that, for references with multiplicities with upper
bounds that may be greater than one, it is not necessary to place all of the references
under a single tag. Although this is clearly wasteful of space in the XML document, it
is allowed.

Finally, for NodeA, any contained objects are written out. But since The Node class
does not define Node as a composite, this step is skipped. The XML for NodeA is
complete:

</SimpleGraph.Node>

This process is repeated for the other values of the Net’s nodes reference, NodeB and
NodeC:
<SimpleGraph.Node xmi.id='a3'>
  <SimpleGraph.Node.name>NodeB</SimpleGraph.Node.name>
  <SimpleGraph.Node.targetNodes>
    <SimpleGraph.Node xmi.id='a3' />
  </SimpleGraph.Node.targetNodes>
</SimpleGraph.Node>

<SimpleGraph.Node xmi.id='a4'>
  <SimpleGraph.Node.name>NodeC</SimpleGraph.Node.name>
  <SimpleGraph.Node.marker>
    <SimpleGraph.Node xmi.idref='a7' />
  </SimpleGraph.Node.marker>
</SimpleGraph.Node>

Notice that for NodeB, the “marker” reference element is omitted. When the lower bound of the multiplicity of an Attribute or a Reference is zero, and no value is present, the element tag may be omitted. In a similar fashion, the “target” reference element is absent for NodeC. The composite reference “nodes” is now fully represented, as is completed in the XML with a corresponding end tag:

</SimpleGraph.Net.nodes>

Next the Token objects contained via the tokens Reference of Net are written out as XML:

<SimpleGraph.Net.tokens>

Each Token object is written out as the other objects, starting with the attributes. The TokenColor data type is an enumeration. Attributes whose types are enumerations or boolean are represented is a special manner. Their value is represented as an element attribute value, to increase XML parser validation.

<SimpleGraph.Token xmi.id='a5' color='green' />

Since the value of the attribute is encoded in the tag of the empty element, a separate end tag is not used. The Token class is defined with the single attribute. If the class were derived from a supertype, the values of attributes and references defined in the supertype would also be written out as XML, preceding the attributes of the class itself. Unlike the Node class, the Token class has no composite references. The single reference defined for token provides the value of the owner, the Net object acting as the “net” object in the composite link. These references need not be written, since the XML element containment indicates the composition. They are useful when the contained object is reached via a link attribute.

The remaining Tokens from the Net’s “tokens” reference yield:

<SimpleGraph.Token xmi.id='a6' color='blue' />
<SimpleGraph.Token xmi.id='a7' color='red' />
</SimpleGraph.tokens>

At this point, all the values that make up the model have been written out as XML. The Net object is completed with the end tag:
All this XML will be embedded in the standard XML element, as described later. Also, sometimes object links will not be represented via references, and need to be represented in XML after the root element. For this simple model though, no unrepresented links remain.

5.3.2 MOF’s Role in XML Production

The specific generation rules rely on an MOF definition of the model’s metamodel. It would simply not be possible to define meaningful production rules that would work on any arbitrary model, regardless of its metamodel. The single meta-metamodel provides the commonality among models, allowing the metamodel information to be uniformly represented. In addition, the MOF defines standard interfaces for the model elements of instances of MOF-defined metamodels. These interfaces – from the MOF’s Reflective module – provide for access to an object’s metaclass, attribute values, and reference values, among other capabilities. The operations of these interfaces provide an unambiguous means of specifying the access of model elements’ metamodel and values.

In order for a metamodel to have its models interchanged through XMI, that metamodel must be representable through the MOF, as an instance of the MOF Model. However, this specification does not actually require an implementation to make use of an MOF, the MOF-defined Reflective interfaces, or even have metamodels represented as instances of the MOF model. The implementation must, however, conform to the generation rules. These rules are based on the metamodels defined via the MOF and the use of the operations in the Reflective interfaces.

5.3.3 Production by Package Extent

It may not always be possible or useful to represent a desired set of modeling elements through a composition hierarchy. For this reason, XMI defines a second set of rules for generating XML from modeling elements.

The MOF provides the Package element in support of metamodel development. At the metamodel level, Package objects are always the top-most (uncontained) elements. A Package will contain Classes and Associations, directly and possibly through nested Packages. In the IDL generated from a MOF metamodel, interfaces represent specific features of these Packages, Classes, and Associations, in the use of model development. For each Package, there is a corresponding subtype of RefPackage, an interface in the MOF’s Reflective module. Likewise, for each Class, there is a corresponding subtype of RefObject, and for each Association, a corresponding subtype of RefAssociation.

These interfaces define a structure which mirrors the metamodel structure. So the RefPackage subtype corresponding to the top-level Package in the metamodel contains all the other RefPackages, RefObjects, and RefAssociations. Each RefObject subtype object can provide all of the current objects of the class it represents; each RefAssociation subtype object can provide all the links corresponding to the Association it represents. The Package Extent, then, is the top-
level **RefPackage** subtype object, all the **RefPackage**, **RefObject**, and **RefAssociation** subtype objects it contains, and all the objects and links associated with them.

![Diagram](image_url)

*Figure 5-5  Generated interfaces from the SimpleGraph metamodel*

In this example, the IDL generation creates interfaces SimpleGraphPackage, NetClass, NodeClass, TokenClass, and Arc. Figure 5-5 shows some of the interfaces generated for the example SimpleGraph metamodel. Suppose two different Nets were modeled, with an Arc crossing from one net to the next, as shown in Figure 5-6.

These nets are shown in Figure 5-7 on page 5-11, as instances of the **SimpleGraph** metamodel. The dashed lines in that figure represent the extent the **NetClass**, **NodeClass**, and **TokenClass**. The extent of the **SimpleGraphPackage** includes those extents.

The rules for XML Production by Package Extent act upon the uncontained **RefPackage** instance, producing an XML document, which represents all the elements in the extent of that **RefPackage**. In the example, the rules are applied to the **SimpleGraphPackage** instance.

The same XML document prologue and enclosing element is required as was for Production by Object Containment. Then, the **SimplePackageClass** is traversed. For each **RefObject** instance, the extent is examined. Any object that is not participating as a component in a composition link becomes the starting point for generating XML.
For instance, from the **NodeClass**, all Node instances can be accessed. But since all are at the component end of a composition link, none are used in XML production. When the **NetClass** is accessed, though, each of the two objects in its extent are uncontained – not on the component end of a composition link. So, within one Net instance, XML is produced in the same manner as described before:

```xml
<SimpleGraph.Net xmi.id='a1'>
  <SimpleGraph.Net.created>
    <XML.field>1868128</XML.field>
    <XML.field>GMT</XML.field>
  </SimpleGraph.Net.created>
  <SimpleGraph.Net.nodes>
    <SimpleGraph.Node xmi.id='a2'>
      <SimpleGraph.Node.name>NodeX</SimpleGraph.Node.name>
      <SimpleGraph.Node.targetNodes>
        <SimpleGraph.Node xmi.id='a3' />
      </SimpleGraph.Node.targetNodes>
    </SimpleGraph.Node>
    <SimpleGraph.Node xmi.id='a3'>
      <SimpleGraph.Node.name>NodeW</SimpleGraph.Node.name>
      <SimpleGraph.Node.targetNodes>
        <SimpleGraph.Node xmi.id='a6' />  
      </SimpleGraph.Node.targetNodes>
    </SimpleGraph.Node>
  </SimpleGraph.Net.nodes>
</SimpleGraph.Net>
```

*Figure 5-6  Example of two nets with a connecting arc*
Similarly, the second Node in the NodeClass extent is used to produce the following XML:

```xml
</SimpleGraph.Net>
</SimpleGraph.nodes>
</SimpleGraph.Net>
```

Similarly, the second Node in the NodeClass extent is used to produce the following XML:

```xml
<SimpleGraph.Net xmi.id='a4'>
<SimpleGraph.Net.created>
<XML.field>1872537</XML.field>
<XML.field>GMT</XML.field>
</SimpleGraph.Net.created>
</SimpleGraph.Net.nodes>
<SimpleGraph.Net xmi.id='a5'>
<SimpleGraph.Node.name>NodeY</SimpleGraph.Node.name>
```

Figure 5-7  Objects representing multiple Nets and instances of RefPackage and RefObject subtypes
The Production by Package Extent is not unlike writing out an entire workspace, environment, or database. This approach is more desirable when:

- more than one containment hierarchy needs to be exchanged;
- there are interconnections among separate containment hierarchies that need to be replicated; or
- classifier-level attributes need to be replicated.

Conversely, creating XML using Production by Object Containment provides:

- finer granularity of the units of interchange; and
- rules definition less dependent upon the `RefPackage`, `RefAssociation`, and `RefObject` features.

5.4 Distinctions between Approaches

5.4.1 Distinctions between Approaches in Certain Situations

The examples above used very simple models. Some more complex models create situations in which the use each of the two approaches has different consequences.

5.4.2 External Links

Each of the Reference links in the examples referred to an XML element within the XML document. But references can also refer to objects without a representative XML element in the document. Consider the two nets in the second example above. If Production by Object Containment is used to produce XML representing the Net which
contains NodeW and NodeX, then the reference of NodeX to NodeZ must be an external link. Since NodeZ is not part of the Net which is used to produce the XML, it will not be represented in the generated document. Instead an href will be used, which can be resolved to navigate to a representation of the NodeZ object.

This distinction means that, for that example, result of Production by Package Extent would be different than applying Production by Object Containment to the two Net instances. In the latter approach, two XML documents are produced.

5.4.3 Links not Represented by References

On the example metamodel, each Association had a corresponding Reference defined for the class at one end. However, it is possible, and sometimes desirable or necessary, to define associations without a reference associated with either Association End. For instance, suppose in the SimpleGraph metamodel that the targetNodes Reference was not defined in the Nodes class. Under both approaches, the XML Node elements will not contain any references to the target Nodes. Instead, the links corresponding to the Arc association would be represented in the contents of an Arc element, which itself would be contained by the standard XMI.content element.

For Production by Package Extent, after the XML is produced from each of the uncontained objects (and their contents), each of the RefAssociation instances are examined for links in their extent that are not represented in the document. These links would be defined by Associations where no Reference is defined for either end.

For Production by Object Containment, the RefAssociation instances are also examined. However, the only links written out are those links not already represented by references in which the objects at both ends are in the containment hierarchy.

5.4.4 Classifier-level Attributes

The MOF supports the definition of classes with classifier-level attributes. At the time of model development, within an MOF, these attributes are part of and managed by the RefObject instances (the class proxies) contained by the RefPackage. For Production by object containment, the values of a classifier-level attribute are not included. Conversely, in Production by Package Extent, all classifier-level attributes are included in the XML document. This again highlights the distinction between the approaches. In programming languages classifier-level attributes, in the form of class variables or static members, are most often considered part of the programming environment. For instance, serialization techniques usually do not serialize these attributes.

5.4.5 Standard Elements

Model data placed in an XML document using the rules of XMI are contained in standard XML elements defined by XMI. XMI document is encapsulated in a set of standard XMI elements. These elements are described in “Requirements for XMI DTDs.”
XML Document Production

Contents

This chapter includes the following topics.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Introduction”</td>
<td>6-1</td>
</tr>
<tr>
<td>“ENBF Rules Representation”</td>
<td>6-1</td>
</tr>
</tbody>
</table>

6.1 Introduction

This chapter specifies the production of an XML document from a model. It is essential for successful model interchange that this specification be complete and unambiguous. It is also essential that all significant aspects of the metadata are included in the XML document and can be recovered from it.

XMI’s XML document production process is defined as a set of production rules. When these rules are applied to a model or model fragment, the result is an XML document. The inverse of these rules can be applied to an XML document to reconstruct the model or model fragment. In both cases, the rules are implicitly applied in the context of the specific metamodel for the metadata being interchanged.

The production rules are provided as a specification of the XML document production and consumption processes. They should not be viewed as prescribing any particular algorithm for XML producer or consumer implementations.

6.2 ENBF Rules Representation

The XML produced by XMI is represented here in Extended Backus Naur Form (EBNF). The following are the production rules:
1.  <Document> ::= <2:XMI>

Description:
1. A document is contained within an XMI element.

2.  <XMI> ::= "<XMI" <2a:Namespaces>
"version=" //XMI version//
("timestamp=" //timestamp//)?
("verified=" //verified//)?
"">
( <3:Header> )?
( <6:Content> )?
( <4:Differences> )?
( <5:Extensions> )?
"</XMI>"

2a. <Namespaces> ::= ( "xmlns:" <2c:NsName> "=" <2d:NsURI> )*
2b. <Namespace> ::= ( <2c:NsName> ":" )?
2c. <NsName> ::= //Name of namespace//
2d. <NsURI> ::= //URI of namespace//
2e. <Link> ::= "href=" //href// ( //XLink attributes// )?

Description:
2. An XMI element consists of namespace declarations, if any, and optional sections for a header, content, differences, and extensions. The version must be “1.2,” an optional timestamp is included, and an optional verification at source indicator of “true” or “false.”

2a. The set of namespace declarations for the XMI element. If there are multiple metamodels used with DTDs, the DTDs should be concatenated with the fixed declarations included only once. The XML attribute declarations for the XMI element’s attributes should all be included.

2b. The use of a namespace name, including a “::” separator. If the namespace name is blank, the result is the empty string.

2c. A particular namespace name.

2d. The logical URI of the namespace. Note that namespaces are resolved to logical URIs, as opposed to physical ones, so that there is no expectation that this URI will be resolved and that there will be any information at that location.

2e. A cross-document link that is intended to be compliant with a future XLink specification. The href contains the URI of the document to link to.
3. <Header> ::= "<XMI.header>"
   ( <3a:Documentation> )?
   ( <3b:Model> )*
   ( <3c:Metamodel> )*
   ( <3d:Metametamodel> )*
   ( <3e:Import> )*
   "</XMI.header>"

3a. <Documentation> ::= "<XMI.documentation>" //text//
   ( "<XMI.owner>" //text// "</XMI.owner>" )*
   ( "<XMI.contact>" //text// "</XMI.contact>" )*
   ( "<XMI.longDescription>" //text// "</XMI.longDescription>" )*
   ( "<XMI.shortDescription>" //text// "</XMI.shortDescription>" )*
   ( "<XMI.exporter>" //text// "</XMI.exporter>" )*
   ( "<XMI.exporterVersion>" //text// "</XMI.exporterVersion>" )*
   ( "<XMI.exporterID>" //text// "</XMI.exporterID>" )*
   ( "<XMI.notice>" //text// "</XMI.notice>" )*
   "</XMI.documentation>"

3b. <Model> ::= ( "<XMI.model xmi.name=" //name//
    "xmi.version=" //version//
    "2e:Link>? "">" //text//
    "</XMI.model>" )*

3c. <Metamodel> ::= ( "<XMI.metamodel xmi.name=" //name//
    "xmi.version=" //version//
    "2e:Link>? "">" //text//
    "</XMI.metamodel>" )*

3d. <Metametamodel> ::= ( "<XMI.metametamodel xmi.name=" //name//
    "xmi.version=" //version//
    "2e:Link>? "">" //text//
    "</XMI.metametamodel>" )*

3e. <Import> ::= ( "<XMI.import xmi.name=" //name//
    "xmi.version=" //version//
    "2e:Link>? "">" //text//
    "</XMI.import>" )*

**Description:**

3. An XMI header consists of optional documentation, model, metamodel, metametamodel, and import elements.

3a. The documentation element contains several optional fields, described in Chapter 3.
3b. The model tag is used to declare the name and version of the information in the XML.contents section. A link to additional metamodel data is optional. Documentation content is allowed.

3c. The metamodel tag is used to declare the name and version of the information in the metamodel instantiated in XML.contents section. A link to additional metamodel data is optional, with the physical URI where an XMI document containing the metamodel would be found. The name of the metamodel should match the name declared in the namespace in rule 1. Documentation content is allowed.

3d. The metametamodel tag is used to declare the name and version of the information in the metamodel instantiated in XML.contents section. A link to additional metametamodel data is optional, with the physical URI where an XMI document containing the metamodel would be found. Documentation content is allowed.

3e. The import tag is used to declare the name and version of the information of imported models in XML.contents section. A link to the imported models data is optional, with the physical URI where an XMI document containing the imported models would be found. The imports may be found from the MOF imports elements. Documentation content is allowed.

4. `<Differences>` ::= "<XMI.difference>"
   ( `<4:Differences>`
     | `<4a:Delete>`
     | `<4b:Add>`
     | `<4c:Replace>` )*
   "</XMI.difference>"

4a. `<Delete>` ::= "<XMI.delete" <2e:Link> "/>"

4b. `<Add>` ::= "<XMI.add" <2e:Link>
   ( "xmi.position="/position/" )? ">
   <6a:ContentElements>
   "</XMI.add>"

4c. `<Replace>` ::= "<XMI.replace" <2e:Link>
   ( "xmi.position="/position/" )? ">
   <6a:ContentElements>
   "</XMI.replace>"

**Description:**

4. A set of differences, in terms of nested differences, adds, deletes, and replacements.

4a. A link to a deleted element.

4b. A link to an element to add the contents to, and an optional position.

4c. A link to an element to replace the contents with those contained below, and an optional position.
5. \(<\text{Extensions}\>) \::= "<\text{XMI.extensions xmi.extender=}" //extender// "">"
\hspace{1em} //\text{Extension elements} //
\hspace{1em} "</\text{XMI.extensions}>"

**Description:**
5. A section for extensions, with an optional extender identifier. The contents are unrestricted.

6. \(<\text{Content}\>) \::= "<\text{XMI.content}>"
\hspace{1em} \(<6a:\text{ContentElements}>"</\text{XMI.content}>"

6a. \(<\text{ContentElements}\>) \::= ( \(<7:\text{ObjectAsElement}>\)*
\hspace{1em} ( \(<9:\text{ClassAttributes}>\)*
\hspace{1em} ( \(<10:\text{OtherLinks}>\) )?}

**Description:**
6. The XMI information to be interchanged.
6a. The contents are a set of top level objects, classifier level attributes, and other links.
7.  <ObjectAsElement>   ::= <7a:ObjectStart> <8:ObjectContents>  
    <7b:ElementEnd>
7a. <ObjectStart>       ::= "<" <7c:ElementName> 
    ( <7e:ElementId> )?  
    <7h:ElementAttributes> 
    
    
    
    
    
    
7b. <ElementEnd>        ::= "/" <7c:ElementName> 
7c. <ElementName>       ::= <2b:Namespace> <7d:Elmt> 
    ( "." <7d:Elmt>)?
7d. <Elmt>              ::= //Name of definition//
7e. <ElementId>         ::= ( "xmi.id=" //id// )?  
    ( "xmi.label=" //label// )?  
    ( "xmi.uuid=" //uuid// )?
7f. <ElementRef>        ::= ( "xmi.idref=" //reference id//  
    | <2e:Link> )?
7g. <ObjectRef>         ::= "<" <7c:ElementName> 
    ( <7e:ElementId> )? 
    <7f:ElementRef> 
    
7h. <ElementAttributes> ::= ( <7i:DataValueAtt>  
    | <7j:EnumValueAtt>  
    | <7k:RefValueAtt> )* 
7i. <DataValueAtt>      ::= "<" <7m:AttName> "=" //value// 
7j. <EnumValueAtt>      ::= "<" <7m:AttName> "=" //enumeration literal// 
7k. <RefValueAtt>       ::= "<" <7m:AttName> "=" <7l:RefValues> 
7l. <RefValues>         ::= ( //reference id// " " )* 
7m. <AttName>           ::= //Name of defining attribute//

**Description:**

7. An object has a starting element, contents, and a closing element. If the contents are empty, an alternative form is to include the element end tag as a "/" in the starting tag as an XML shortcut.

7a. The start tag of an element consists of the element name, identifying information, and attributes.

7b. The end tag name is the same as the start tag name, preceded with a "/."

7c. The element tag name is the name of the namespace followed by the element name. For class, package, and association instances, this name is the name of the type instantiated. For attributes, references, and links, the name is the name of the containing class, package or association, a ",," and the name of the attribute or reference.

7d. The tag name of an element is the name of its definition.
7e. The identifiers of an element are an optional id, label, and uuid. If the operation `refMofId()` defined by MOF returns a value, it is recommended that you put it in the value of the `xmi.uuid` XML attribute, but you are not required to do so.

7f. An element reference is a reference to another element by ID or a reference by an external Link.

7g. An object reference is a link by proxy element, which may declare identifiers that should match those identifiers of the other end of the link.

7h. The XML attributes of the element are data, enumeration, or reference attributes. Either the XML attribute or contained XML element for a particular model element or reference, but not both.

7i. An XML attribute for data represents a single-valued model attribute with a value expressed as an XML CDATA. If the datatype value is for one of the MOF 1.4 primitive datatypes, the lexical representation of the value is defined by the corresponding Schema Datatype in XML Schema Part 2: Datatypes, W3C Recommendation 02 May 2001. The following table shows the MOF 1.4 primitive datatypes and the corresponding schema datatypes:

<table>
<thead>
<tr>
<th>MOF 1.4 Datatype</th>
<th>Schema Datatype</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>boolean</td>
</tr>
<tr>
<td>Integer</td>
<td>int</td>
</tr>
<tr>
<td>Long</td>
<td>long</td>
</tr>
<tr>
<td>Float</td>
<td>float</td>
</tr>
<tr>
<td>Double</td>
<td>double</td>
</tr>
<tr>
<td>String</td>
<td>string</td>
</tr>
</tbody>
</table>

7j. An XML attribute for enumeration represents an enumerated model attribute with the value matching one of the allowed enumeration literals for the attribute’s type.

7k. 7l. An XML attribute for reference contains the XML ID of each referenced object, separated by a space.

7m. The name of the XML attribute is the name of the model attribute or reference.
8.  `<ObjectContents>` ::= `<8a:ContentElement>` *

8a.  `<ContentElement>` ::= `<8b:ItemHdr`
  `<8c:ItemContent>` *
  `<8d:ItemEnd>` ?

8b.  `<ItemHdr>` ::= "" `<8k:ItemTag`
  `(8h:AttribEnumValue "/?")?`
  "">

8c.  `<ItemContent>` ::= `<8e:AttribContent>`
  | `<8f:ReferenceContent>`
  | `<8g:CompositeContent>`

8d.  `<ItemEnd>` ::= "" `<8k:ItemTag>` ">

8e.  `<AttribContent>` ::= `<8i:AttribData>` | `<8j:AttribObject>`

8f.  `<ReferenceContent>` ::= `<8g:ObjectRef`

8g.  `<CompositeContent>` ::= `<7:ObjectAsElement`

8h.  `<AttribEnumValue>` ::= "xmi.value" "=" //enumeration literal//

8i.  `<AttribData>` ::= // value // | "" "XMI.reference" `<2e:Link` "/">

8j.  `<AttribObject>` ::= `<7:ObjectAsElement`

8k.  `<ItemTag>` ::= `<7c:ElementName`

**Description:**

8. The contents of an object is a (possibly-empty) sequence of `<ContentElement>`s, one or more per instance-level attribute or reference (composite or non-composite) of the object.

**Note** – If any value for some attribute or reference appears in the `<ObjectContents>` of an object, then *all* values of that attribute or reference must appear in the `<ObjectContents>`; no values may appear in the `<ElementAttributes>` (7h) of the object, and vice versa.

8a. A `<ContentElement>` consists of a header identifying the attribute or reference, followed by one or more element content elements with the actual value(s) of the attribute or reference, and (if needed) a terminating end of the element. For multi-valued attributes or references, `<ContentElement>`s for the attribute or reference may appear as many times as necessary to convey all values. Each `<ContentElement>` shall convey at least one value, but may contain as many as the XMI producer wishes to convey in it, each in its own XML element, except for data-valued attributes, as explained in the note, below. If the attribute or reference is ordered, the lexical ordering of the values shall correspond to the ordering in the source class.
Note – This syntax allows the XMI producer to include more than one value of a multi-valued attribute or reference in a <ContentElement>. However, for data-valued attributes (strings, integers, etc.), the rules of XML are such the values in the separate XML elements should be treated as though concatenated into a single value. Thus, for data-valued attributes, each value must appear in a separate <ElementContent> element.

Note – If the minimum cardinality of the attribute or reference is 0 and there are no values to be transmitted, the corresponding <ElementContent> for the attribute or reference shall not appear.

Note – If more than one <ContentElement> is used for an attribute or reference, all of the <ContentElements> for that attribute or reference shall appear together, with no intervening <ContentElement>s for some other attribute or reference.

8b. The XML tag of the <ItemHdr> for the element names the attribute or reference, with respect to the object of which it is a part. For single-valued attributes that have enumerated values, the value may be specified in the <ItemHdr> itself, in which case the <ItemContent> is not used and <ItemEnd> is optional.

8c. The <ItemContent> is the “value” of the attribute or reference. The particular format of the <ItemContent> for an attribute depends on the nature of the attribute or reference.

8d. The tag of an <ItemEnd>, if present, matches the tag of the ItemHdr.

8e. For attributes whose type is a DataType, the <AttribContent> is the value of that DataType, expressed as an XML string. If the value is for one of the MOF 1.4 primitive datatypes, the lexical representation of the value is defined as explained in the description of <DataValueAtt> (7i) in the previous section. Otherwise, the <AttribContent> is a Class, and the <AttribContent> is the Class object.

8f. For references that are not composite, the <ReferenceContent> form of the <ContentElement> is used. This form is an XMI reference to the <ObjectAsElement> of the type of the referencedEnd of the reference.

8g. For references that are composite, the <CompositeContent> form of the <ContentElement> is used. The <ObjectAsElement> of the composed object is thus embedded within <ObjectContent> of the object that composes it.

8h. An AttribEnumValue is used for attributes whose value is an enumeration.

8i. AttribData can be either the value of an attribute, expressed as a string, or a reference to remote content.

8j. If an Attribute has an object as its value, the object is contained as an element within the ObjectContents. In essence, it is treated much like a composed referenced object.
8k. The <ItemTag> is an <ElementName>. In this case, the ("." <7d:Elmt>) construct is present, since it is naming an attribute or reference component of a class.

9.  <ClassAttributes> ::= ( <8a:AttributeAsElmt> )*  

Description:
9. All classifier-level attributes are expressed using the XML element form.

10. <OtherLinks> ::= <10a:AssociationStart> ( <10c:AssociationEndRef> <10c:AssociationEndRef> )* <10b:AssociationEnd>
10a. <AssociationStart> ::= <7a:ObjectStart>
10b. <AssociationEnd> ::= <7b:ElementEnd>
10c. <AssociationEndRef> ::= <7a:ObjectStart> <7g:ObjectRef> <7b:ElementEnd>

Description:
10. All associations that have no references are placed here. Each associationEnd's links are contained as pairs of nested XML elements.

10a. The start tag of the association begins with the association name.
10b. The end tag of the association element.
10c. A reference to the linked element from the AssociationEnd.
A.1 Introduction

This appendix contains a normative DTD that represents the UML 1.1 metamodel. It was generated by a program that closely follows rule set 1 in the “XML DTD Production” chapter. The program created the DTD from a Rational Rose model representing the UML 1.1 metamodel as defined in the document “UML Semantics version 1.1, 1 September 1997”. The metamodel in that document was changed by adding role names to associations that did not have them.

As noted in Section 4.4.12 of the XMI submission, the XMI submitters are not attempting to create official metamodels for UML (or for the MOF). We propose that the groups that propose and maintain the standards for metamodels create official versions of the metamodels and DTDs corresponding to them.

In the meantime, there is a need to exchange MOF and UML metadata now, so this appendix presents a normative DTD for UML 1.1; it can be used to develop software.

One minor change from rule set 1 is that all composed roles are included at the beginning of the UML declarations in the DTD, whereas rule set 1 would place them in the packages that they belong to. That difference does not change the semantics of the DTD.

The source of the UML 1.1 metamodel may have outstanding issues that might affect the DTD included in this appendix.

A.2 UML DTD

```xml
<?xml version="1.0" encoding="UTF-8" />

<!-- Generated by: XMI Framework 1.0 -->
<!-- Date: Fri Oct 22 14:31:09 PDT 1999 -->
```
<!-- _______________________________________________________________ -->
<!-- -->
<!-- This section of the DTD contains XML declarations required by -->
<!-- XMI. -->
<!-- _______________________________________________________________ -->
<!-- _______________________________________________________________ -->
<!-- XMI is the top-level XML element for XMI transfer text -->
<!-- _______________________________________________________________ -->

<!ELEMENT XMI (XMI.header?, XMI.content?, XMI.difference*,
             XMI.extensions*) >
<!ATTLIST XMI
  xmi.version CDATA #FIXED "1.1"
  timestamp CDATA #IMPLIED
  verified (true | false) #IMPLIED
>
<!-- _______________________________________________________________ -->
<!-- XMI.header contains documentation and identifies the model, -->
<!-- metamodel, and metametamodel -->
<!-- _______________________________________________________________ -->
<!ELEMENT XMI.header (XMI.documentation?, XMI.model*, XMI.metamodel*,
                       XMI.metametamodel*, XMI.import*) >

<!-- _______________________________________________________________ -->
<!-- documentation for transfer data -->
<!-- _______________________________________________________________ -->
<!ELEMENT XMI.documentation (#PCDATA | XMI.owner | XMI.contact |
                        XMI.longDescription | XMI.shortDescription |
                        XMI.exporter | XMI.exporterVersion |
                        XMI.notice)* >
<!ELEMENT XMI.owner ANY >
<!ELEMENT XMI.contact ANY >
<!ELEMENT XMI.longDescription ANY >
<!ELEMENT XMI.shortDescription ANY >
<!ELEMENT XMI.exporter ANY >
<!ELEMENT XMI.exporterVersion ANY >
<!ELEMENT XMI.exporterID ANY >
<!ELEMENT XMI.notice ANY >
<!ENTITY % XMI.element.att
  "xmi.id ID #IMPLIED xmi.label CDATA #IMPLIED xmi.uuid CDATA #IMPLIED">

<!ENTITY % XMI.link.att
  "href CDATA #IMPLIED xmi.idref IDREF #IMPLIED xml:link CDATA #IMPLIED xlink:inline (true | false) #IMPLIED xlink:actuate (show | user) #IMPLIED xlink:content-role CDATA #IMPLIED xlink:title CDATA #IMPLIED xlink:show (embed | replace | new) #IMPLIED xlink:behavior CDATA #IMPLIED">

<!ENTITY % XMI.model
  "xmi.name CDATA #REQUIRED xmi.version CDATA #IMPLIED">

<!ENTITY % XMI.metamodel
  "xmi.name CDATA #REQUIRED xmi.version CDATA #IMPLIED">

<!ENTITY % XMI.metametamodel
  "xmi.name CDATA #REQUIRED xmi.version CDATA #IMPLIED">

<!ELEMENT XMI.model ANY>
<!ATTLIST XMI.model
  %XMI.link.att;
  xmi.name CDATA #REQUIRED
  xmi.version CDATA #IMPLIED>

<!ELEMENT XMI.metamodel ANY>
<!ATTLIST XMI.metamodel
  %XMI.link.att;
  xmi.name CDATA #REQUIRED
  xmi.version CDATA #IMPLIED>

<!ELEMENT XMI.metametamodel ANY>
<!ATTLIST XMI.metametamodel
  %XMI.link.att;
  xmi.name CDATA #REQUIRED
  xmi.version CDATA #IMPLIED>
<?xml version="1.0"?>

<xmi.name CDATA #REQUIRED>
<xmi.version CDATA #IMPLIED>

<!ELEMENT XMI.import ANY >
<!ATTLIST XMI.import
  %XMI.link.att;
  xmi.name CDATA #REQUIRED
  xmi.version CDATA #IMPLIED
>

<!ELEMENT XMI.content ANY >

<!ELEMENT XMI.extensions ANY >
<!ATTLIST XMI.extensions
  xmi.extender CDATA #REQUIRED
>

<!ELEMENT XMI.extension ANY >
<!ATTLIST XMI.extension
  %XMI.element.att;
  %XMI.link.att;
  xmi.extender CDATA #REQUIRED
  xmi.extenderID CDATA #IMPLIED
>

<!ELEMENT XMI.difference ANY >
<!ATTLIST XMI.difference
  %XMI.link.att;
  xmi.name CDATA #REQUIRED
  xmi.version CDATA #IMPLIED
>
<!ELEMENT XMI.difference (XMI.difference | XMI.delete | XMI.add | XMI.replace)* >
<!ATTLIST XMI.difference
%XMI.element.att;
%XMI.link.att;>

<!-- _______________________________________________________________ -->
<!-- -->
<!-- XMI.delete represents a deletion from a base model -->
<!-- _______________________________________________________________ -->
<!ELEMENT XMI.delete EMPTY >
<!ATTLIST XMI.delete
%XMI.element.att;
%XMI.link.att;>

<!-- _______________________________________________________________ -->
<!-- -->
<!-- XMI.add represents an addition to a base model -->
<!-- _______________________________________________________________ -->
<!ELEMENT XMI.add ANY >
<!ATTLIST XMI.add
%XMI.element.att;
%XMI.link.att;
xmi.position CDATA "-1">

<!-- _______________________________________________________________ -->
<!-- -->
<!-- XMI.replace represents the replacement of a model construct -->
<!-- with another model construct in a base model -->
<!-- _______________________________________________________________ -->
<!ELEMENT XMI.replace ANY >
<!ATTLIST XMI.replace
%XMI.element.att;
%XMI.link.att;
xmi.position CDATA "-1">

<!-- _______________________________________________________________ -->
<!-- -->
<!-- XMI.reference may be used to refer to data types not defined in -->
<!-- the metamodel -->
<!-- _______________________________________________________________ -->
<!ELEMENT XMI.reference ANY >
<!ATTLIST XMI.reference
%XMI.link.att;>

<!ATTLIST XMI
<!ELEMENT UML:Reception)* >

<!ELEMENT UML:BehavioralFeature.parameter (UML:Parameter)* >

<!ELEMENT UML:Dependency.subDependencies (UML:Dependency |
UML:Refinement | UML:Usage |
UML:Trace | UML:Binding)* >

<!ELEMENT UML:AssociationEnd.qualifier (UML:Attribute)* >

<!ELEMENT UML:Association.connection (UML:AssociationEnd |
UML:AssociationEndRole)* >

<!ELEMENT UML:Stereotype.requiredTag (UML:TaggedValue)* >

<!ELEMENT UML:Enumeration.literal (UML:EnumerationLiteral)* >

<!ELEMENT UML:Signal.parameter (UML:Parameter)* >

<!ELEMENT UML:ActionSequence.action (UML:Action |
UML:CreateAction |
UML:CallAction |
UML:LocalInvocation |
UML:ReturnAction |
UML:SendAction |
UML:UninterpretedAction |
UML:TerminateAction |
UML:DestroyAction)* >

<!ELEMENT UML:Action.actualArgument (UML:Argument)* >

<!ELEMENT UML:Link.linkRole (UML:LinkEnd)* >

<!ELEMENT UML:Instance.slot (UML:AttributeLink)* >

<!ELEMENT UML:Collaboration.interaction (UML:Interaction)* >

<!ELEMENT UML:Interaction.message (UML:Message)* >

<!ELEMENT UML:StateMachine.top (UML:State |
UML:CompositeState |
UML:SimpleState |
UML:SubmachineState |
UML:ActionState |
UML:ObjectFlowState |
UML:ActivityState)* >
<!ELEMENT UML:StateMachine.transitions (UML:Transition)* >

<!ELEMENT UML:Transition.guard (UML:Guard)* >

<!ELEMENT UML:Transition.effect (UML:ActionSequence)* >

<!ELEMENT UML:State.internalTransition (UML:Transition)* >

<!ELEMENT UML:State.entry (UML:ActionSequence)* >

<!ELEMENT UML:State.exit (UML:ActionSequence)* >

<!ELEMENT UML:CompositeState.substate (UML:StateVertex |
    UML:PseudoState | UML:State |
    UML:CompositeState |
    UML:SimpleState |
    UML:SubmachineState |
    UML:ActionState |
    UML:ObjectFlowState |
    UML:ActivityState)* >

<!ELEMENT UML:ActivityModel.partition (UML:Partition)* >

<!-- _______________________________________________________________ -->
<!-- -->
<!-- CLASS: Foundation.Auxiliary_Elements.Refinement -->
<!-- _______________________________________________________________ -->

<!ELEMENT UML:Refinement.mapping (UML:Mapping) >

<!ELEMENT UML:Refinement (UML:ModelElement.name |
    UML:ModelElement.visibility |
    UML:Dependency.description |
    UML:Refinement.mapping | XMI.extension |
    UML:ModelElement.binding |
    UML:ModelElement.template |
    UML:ModelElement.templateParameter |
    UML:ModelElement.implementation |
    UML:ModelElement.view |
    UML:ModelElement.presentation |
    UML:ModelElement.namespace |
    UML:ModelElement.constrain |
    UML:ModelElement.requirement |
    UML:ModelElement.provision |
    UML:ModelElement.stereotype |
    UML:ModelElement.elementReference |
    UML:ModelElement.collaboration |
    UML:ModelElement.behavior |
    UML:ModelElement.partition |
<!ATTLIST UML:Usage
    name CDATA #IMPLIED
    visibility (public | protected | private) #IMPLIED
    description CDATA #IMPLIED
    binding IDREFS #IMPLIED
    template IDREFS #IMPLIED
    templateParameter IDREFS #IMPLIED
    implementation IDREFS #IMPLIED
    view IDREFS #IMPLIED
    presentation IDREFS #IMPLIED
    namespace IDREFS #IMPLIED
    constraint IDREFS #IMPLIED
    requirement IDREFS #IMPLIED
    provision IDREFS #IMPLIED
    stereotype IDREFS #IMPLIED
    elementReference IDREFS #IMPLIED
    collaboration IDREFS #IMPLIED
    behavior IDREFS #IMPLIED
    partition IDREFS #IMPLIED
    owningDependency IDREFS #IMPLIED
    client IDREFS #IMPLIED
    supplier IDREFS #IMPLIED
>%

<-- _______________________________________________________________ -->
<-- -->
<-- CLASS: Foundation.Auxiliary_Elements.Trace -->
<-- _______________________________________________________________ -->

<!ELEMENT UML:Trace (UML:ModelElement.name |
    UML:ModelElement.visibility |
    UML:Dependency.description | XMI.extension |
    UML:ModelElement.binding |
    UML:ModelElement.template |
    UML:ModelElement.templateParameter |
    UML:ModelElement.implementation |
    UML:ModelElement.view |
    UML:ModelElement.presentation |
    UML:ModelElement.namespace |
    UML:ModelElement.constraint |
    UML:ModelElement.requirement |
    UML:ModelElement.provision |
    UML:ModelElement.stereotype |
    UML:ModelElement.elementReference |
    UML:ModelElement.collaboration |
    UML:ModelElement.behavior |
    UML:ModelElement.partition |
    UML:Dependency.owningDependency |
    UML:Dependency.client |
    UML:Dependency.supplier |
    UML:ModelElement.taggedValue |
UML:Dependency.subDependencies)*

<!ATTLIST UML:Trace
  name CDATA #IMPLIED
  visibility (public | protected | private) #IMPLIED
  description CDATA #IMPLIED
  binding IDREFS #IMPLIED
  template IDREFS #IMPLIED
  templateParameter IDREFS #IMPLIED
  implementation IDREFS #IMPLIED
  view IDREFS #IMPLIED
  presentation IDREFS #IMPLIED
  namespace IDREFS #IMPLIED
  constraint IDREFS #IMPLIED
  requirement IDREFS #IMPLIED
  provision IDREFS #IMPLIED
  stereotype IDREFS #IMPLIED
  elementReference IDREFS #IMPLIED
  collaboration IDREFS #IMPLIED
  behavior IDREFS #IMPLIED
  partition IDREFS #IMPLIED
  owningDependency IDREFS #IMPLIED
  client IDREFS #IMPLIED
  supplier IDREFS #IMPLIED
%XMI.element.att;
%XMI.link.att;
>

<!-- _______________________________________________________________ -->
<!-- -->
<!-- CLASS: Foundation.Auxiliary_Elements.Binding -->
<!-- _______________________________________________________________ -->

<!ELEMENT UML:Binding (UML:ModelElement.name |
  UML:ModelElement.visibility |
  UML:Dependency.description | XMI.extension |
  UML:ModelElement.binding |
  UML:ModelElement.template |
  UML:ModelElement.templateParameter |
  UML:ModelElement.implementation |
  UML:ModelElement.view |
  UML:ModelElement.presentation |
  UML:ModelElement.namespace |
  UML:ModelElement.constraint |
  UML:ModelElement.requirement |
  UML:ModelElement.provision |
  UML:ModelElement.stereotype |
  UML:ModelElement.elementReference |
  UML:ModelElement.collaboration |
  UML:ModelElement.behavior |
  UML:ModelElement.partition |
  UML:Dependency.owningDependency |
  UML:Dependency.client | UML:Dependency.supplier |
  UML:ModelElement.taggedValue |
  UML:Dependency.subDependencies |
  UML:Binding.argument)* >
<!ATTLIST UML:Binding
    name CDATA #IMPLIED
    visibility (public | protected | private) #IMPLIED
    description CDATA #IMPLIED
    binding IDREFS #IMPLIED
    template IDREFS #IMPLIED
    templateParameter IDREFS #IMPLIED
    implementation IDREFS #IMPLIED
    view IDREFS #IMPLIED
    presentation IDREFS #IMPLIED
    namespace IDREFS #IMPLIED
    constraint IDREFS #IMPLIED
    requirement IDREFS #IMPLIED
    provision IDREFS #IMPLIED
    stereotype IDREFS #IMPLIED
    elementReference IDREFS #IMPLIED
    collaboration IDREFS #IMPLIED
    behavior IDREFS #IMPLIED
    partition IDREFS #IMPLIED
    owningDependency IDREFS #IMPLIED
    client IDREFS #IMPLIED
    supplier IDREFS #IMPLIED
%XMI.element.att;
%XMI.link.att;
>
<!-- _______________________________________________________________ -->
<!-- -->
<!-- _______________________________________________________________ -->
<!ELEMENT UML:Node.component (UML:Component)* >
<!ELEMENT UML:Node (UML:ModelElement.name | UML:ModelElement.visibility |
    UML:GeneralizableElement.isRoot | UML:GeneralizableElement.isLeaf |
    UML:GeneralizableElement.isAbstract | XMI.extension |
    UML:ModelElement.binding | UML:ModelElement.template |
    UML:ModelElement.templateParameter |
    UML:ModelElement.implementation |
    UML:ModelElement.view |
    UML:ModelElement.presentation |
    UML:ModelElement.namespace |
    UML:ModelElement.constraint |
    UML:ModelElement.requirement |
    UML:ModelElement.provision |
    UML:ModelElement.stereotype |
    UML:ModelElement.elementReference |
    UML:ModelElement.collaboration |
    UML:ModelElement.behavior |
    UML:ModelElement.partition |
    UML:GeneralizableElement.generalization |
    UML:GeneralizableElement.specialization |
    UML:Classifier.parameter |
    UML:Classifier.structuralFeature |
<!ATTLIST UML:Node
    name CDATA #IMPLIED
    visibility (public | protected | private) #IMPLIED
    isRoot (false | true) #IMPLIED
    isLeaf (false | true) #IMPLIED
    isAbstract (false | true) #IMPLIED
    binding IDREFS #IMPLIED
    template IDREFS #IMPLIED
    templateParameter IDREFS #IMPLIED
    implementation IDREFS #IMPLIED
    view IDREFS #IMPLIED
    presentation IDREFS #IMPLIED
    namespace IDREFS #IMPLIED
    constraint IDREFS #IMPLIED
    requirement IDREFS #IMPLIED
    provision IDREFS #IMPLIED
    stereotype IDREFS #IMPLIED
    elementReference IDREFS #IMPLIED
    ModelElement.collaboration IDREFS #IMPLIED
    behavior IDREFS #IMPLIED
    partition IDREFS #IMPLIED
    generalization IDREFS #IMPLIED
    specialization IDREFS #IMPLIED
    parameter IDREFS #IMPLIED
    structuralFeature IDREFS #IMPLIED
    specification IDREFS #IMPLIED
    realization IDREFS #IMPLIED
    associationEnd IDREFS #IMPLIED
    participant IDREFS #IMPLIED
    createAction IDREFS #IMPLIED
    instance IDREFS #IMPLIED
    Classifier.collaboration IDREFS #IMPLIED
    classifierRole IDREFS #IMPLIED
    classifierInState IDREFS #IMPLIED
    component IDREFS #IMPLIED
%XMI.element.att;
%XMI.link.att;
<!ELEMENT UML:Component.deployment (UML:Node)* >

<!ELEMENT UML:Component.implements (UML:ModelElement)* >

<!ELEMENT UML:Component (UML:ModelElement.name |
UML:ModelElement.visibility |
UML:GeneralizableElement.isRoot |
UML:GeneralizableElement.isLeaf |
UML:GeneralizableElement.isAbstract |
XML.extension | UML:ModelElement.binding |
UML:ModelElement.template |
UML:ModelElement.templateParameter |
UML:ModelElement.implementation |
UML:ModelElement.view |
UML:ModelElement.presentation |
UML:ModelElement.namespace |
UML:ModelElement.constraint |
UML:ModelElement.requirement |
UML:ModelElement.provision |
UML:ModelElement.stereotype |
UML:ModelElement.elementReference |
UML:ModelElement.collaboration |
UML:ModelElement.behavior |
UML:ModelElement.partition |
UML:GeneralizableElement.generalization |
UML:GeneralizableElement.specialization |
UML:Classifier.parameter |
UML:Classifier.structuralFeature |
UML:Classifier.specification |
UML:Classifier.realization |
UML:Classifier.associationEnd |
UML:Classifier.participant |
UML:Classifier.createAction |
UML:Classifier.instance |
UML:Classifier.collaboration |
UML:Classifier.classifierRole |
UML:Classifier.classifierInState |
UML:Component.deployment |
UML:Component.implements |
UML:ModelElement.taggedValue |
UML:Namespace.ownedElement |
UML:Classifier.feature)* >

<!ATTLIST UML:Component
  name CDATA #IMPLIED
  visibility (public | protected | private) #IMPLIED
  isRoot (false | true) #IMPLIED
  isLeaf (false | true) #IMPLIED
  isAbstract (false | true) #IMPLIED
  binding IDREFS #IMPLIED
  template IDREFS #IMPLIED
  templateParameter IDREFS #IMPLIED
  implementation IDREFS #IMPLIED
  view IDREFS #IMPLIED
  presentation IDREFS #IMPLIED
namespaces IDREFS #IMPLIED
constraint IDREFS #IMPLIED
requirement IDREFS #IMPLIED
provision IDREFS #IMPLIED
stereotype IDREFS #IMPLIED
elementReference IDREFS #IMPLIED
ModelElement.collaboration IDREFS #IMPLIED
behavior IDREFS #IMPLIED
partition IDREFS #IMPLIED
generalization IDREFS #IMPLIED
specialization IDREFS #IMPLIED
parameter IDREFS #IMPLIED
structuralFeature IDREFS #IMPLIED
specification IDREFS #IMPLIED
realization IDREFS #IMPLIED
associationEnd IDREFS #IMPLIED
participant IDREFS #IMPLIED
createAction IDREFS #IMPLIED
instance IDREFS #IMPLIED
Classifier.collaboration IDREFS #IMPLIED
classifierRole IDREFS #IMPLIED
classifierInState IDREFS #IMPLIED
deployment IDREFS #IMPLIED
implements IDREFS #IMPLIED
%XMI.element.att;
%XMI.link.att;
>
<!-- _______________________________________________________________ -->
<!-- -->
<!-- CLASS: Foundation.Auxiliary_Elements.Comment -->
<!-- _______________________________________________________________ -->
<!ELEMENT UML:Comment (UML:ModelElement.name |
               UML:ModelElement.visibility | XMI.extension |
               UML:ModelElement.binding |
               UML:ModelElement.template |
               UML:ModelElement.templateParameter |
               UML:ModelElement.implementation |
               UML:ModelElement.view |
               UML:ModelElement.presentation |
               UML:ModelElement.namespace |
               UML:ModelElement.constraint |
               UML:ModelElement.requirement |
               UML:ModelElement.provision |
               UML:ModelElement.stereotype |
               UML:ModelElement.elementReference |
               UML:ModelElement.collaboration |
               UML:ModelElement.behavior |
               UML:ModelElement.partition |
               UML:ModelElement.taggedValue)* >
<!ATTLIST UML:Comment
    name CDATA #IMPLIED
    visibility (public | protected | private) #IMPLIED
    binding IDREFS #IMPLIED
template IDREFS #IMPLIED
templateParameter IDREFS #IMPLIED
implementation IDREFS #IMPLIED
view IDREFS #IMPLIED
presentation IDREFS #IMPLIED
namespace IDREFS #IMPLIED
constraint IDREFS #IMPLIED
requirement IDREFS #IMPLIED
provision IDREFS #IMPLIED
stereotype IDREFS #IMPLIED
elementReference IDREFS #IMPLIED
collaboration IDREFS #IMPLIED
behavior IDREFS #IMPLIED
partition IDREFS #IMPLIED
%XMI.element.att;
%XMI.link.att;
>
<!-- _______________________________________________________________ -->
<!-- CLASS: Foundation.Auxiliary_Elements.ViewElement -->
<!-- _______________________________________________________________ -->
<!ELEMENT UML:ViewElement.model (UML:ModelElement)* >
<!ELEMENT UML:ViewElement.presentation (UML:Presentation)* >
<!ELEMENT UML:ViewElement (XMI.extension | UML:ViewElement.model | UML:ViewElement.presentation)* >
<!ELEMENT UML:Presentation.geometry (UML:Geometry) >
<!ELEMENT UML:Presentation.style (UML:GraphicMarker) >
<!ELEMENT UML:Presentation.model (UML:ModelElement)* >
<!ELEMENT UML:Presentation.viewElement (UML:ViewElement)* >
<!ATTLIST UML:Presentation
model IDREFS #IMPLIED
presentation IDREFS #IMPLIED
%XMI.element.att;
%XMI.link.att;
>
viewElement IDREFS #IMPLIED
  %XMI.element.att;
  %XMI.link.att;
>
<!-- _______________________________________________________________ -->
<!-- -->
<!-- CLASS: Foundation.Core.Element -->
<!-- _______________________________________________________________ -->
<!ELEMENT UML:Element (XMI.extension)* >
<!ATTLIST UML:Element
  %XMI.element.att;
  %XMI.link.att;
>
<!-- _______________________________________________________________ -->
<!-- -->
<!-- CLASS: Foundation.Core.ModelElement -->
<!-- _______________________________________________________________ -->
<!ELEMENT UML:ModelElement.name (#PCDATA | XMI.reference)* >
<!ELEMENT UML:ModelElement.visibility EMPTY >
<!ATTLIST UML:ModelElement.visibility
  xmi.value (public | protected | private) #REQUIRED
>
<!ELEMENT UML:ModelElement.binding (UML:Binding)* >
<!ELEMENT UML:ModelElement.template (UML:ModelElement)* >
<!ELEMENT UML:ModelElement.templateParameter (UML:ModelElement)* >
<!ELEMENT UML:ModelElement.implementation (UML:Component)* >
<!ELEMENT UML:ModelElement.view (UML:ViewElement)* >
<!ELEMENT UML:ModelElement.presentation (UML:Presentation)* >
<!ELEMENT UML:ModelElement.namespace (UML:Namespace)* >
<!ELEMENT UML:ModelElement.constraint (UML:Constraint)* >
<!ELEMENT UML:ModelElement.requirement (UML:Dependency)* >
<!ELEMENT UML:ModelElement.provision (UML:Dependency)* >
<!ELEMENT UML:ModelElement.stereotype (UML:Stereotype)* >
<!ELEMENT UML:ModelElement.elementReference (UML:ElementReference)* >
<!ELEMENT UML:ModelElement.collaboration (UML:Collaboration)* >
<!ELEMENT UML:ModelElement.behavior (UML:StateMachine)* >
<!ELEMENT UML:ModelElement.partition (UML:Partition)* >

<!ELEMENT UML:ModelElement (UML:ModelElement.name |
  UML:ModelElement.visibility | XMI.extension |
  UML:ModelElement.binding |
  UML:ModelElement.template |
  UML:ModelElement.templateParameter |
  UML:ModelElement.implementation |
  UML:ModelElement.view |
  UML:ModelElement.presentation |
  UML:ModelElement.namespace |
  UML:ModelElement.constraint |
  UML:ModelElement.requirement |
  UML:ModelElement.provision |
  UML:ModelElement.stereotype |
  UML:ModelElement.elementReference |
  UML:ModelElement.collision |
  UML:ModelElement.behavior |
  UML:ModelElement.partition |
  UML:ModelElement.taggedValue)* >

<!ATTLIST UML:ModelElement
  name CDATA #IMPLIED
  visibility (public | protected | private) #IMPLIED
  binding IDREFS #IMPLIED
  template IDREFS #IMPLIED
  templateParameter IDREFS #IMPLIED
  implementation IDREFS #IMPLIED
  view IDREFS #IMPLIED
  presentation IDREFS #IMPLIED
  namespace IDREFS #IMPLIED
  constraint IDREFS #IMPLIED
  requirement IDREFS #IMPLIED
  provision IDREFS #IMPLIED
  stereotype IDREFS #IMPLIED
  elementReference IDREFS #IMPLIED
  collaboration IDREFS #IMPLIED
  behavior IDREFS #IMPLIED
  partition IDREFS #IMPLIED
%XMI.element.att;
%XMI.link.att;
>

<!-- _______________________________________________________________ -->
<!-- -->
<!-- CLASS: Foundation.Core.Namespace -->
<!-- _______________________________________________________________ -->

<!ELEMENT UML:Namespace (UML:ModelElement.name |
  UML:ModelElement.visibility | XMI.extension |
  UML:ModelElement.binding |
  UML:ModelElement.template |
  UML:ModelElement.templateParameter |
  UML:ModelElement.implementation |
  UML:ModelElement.view |
UML:ModelElement.presentation |  
UML:ModelElement.namespace |  
UML:ModelElement.constraint |  
UML:ModelElement.requirement |  
UML:ModelElement.provision |  
UML:ModelElement.stereotype |  
UML:ModelElement.elementReference |  
UML:ModelElement.collaboration |  
UML:ModelElement.behavior |  
UML:ModelElement.partition |  
UML:ModelElement.taggedValue |  
UML:Namespace.ownedElement)* >

<!ATTLIST UML:Namespace
  name CDATA #IMPLIED
  visibility (public | protected | private) #IMPLIED
  binding IDREFS #IMPLIED
  template IDREFS #IMPLIED
  templateParameter IDREFS #IMPLIED
  implementation IDREFS #IMPLIED
  view IDREFS #IMPLIED
  presentation IDREFS #IMPLIED
  namespace IDREFS #IMPLIED
  constraint IDREFS #IMPLIED
  requirement IDREFS #IMPLIED
  provision IDREFS #IMPLIED
  stereotype IDREFS #IMPLIED
  elementReference IDREFS #IMPLIED
  collaboration IDREFS #IMPLIED
  behavior IDREFS #IMPLIED
  partition IDREFS #IMPLIED
%XMI.element.att;
%XMI.link.att;
>

<!-- ______________________________________________________________________ -->
<!-- -->
<!-- CLASS: Foundation.Core.GeneralizableElement -->
<!-- -->
<!-- ______________________________________________________________________ -->

<!ELEMENT UML:GeneralizableElement.isRoot EMPTY >
<!ATTLIST UML:GeneralizableElement.isRoot
  xmi.value (false | true) #REQUIRED
>
<!ELEMENT UML:GeneralizableElement.isLeaf EMPTY >
<!ATTLIST UML:GeneralizableElement.isLeaf
  xmi.value (false | true) #REQUIRED
>
<!ELEMENT UML:GeneralizableElement.isAbstract EMPTY >
<!ATTLIST UML:GeneralizableElement.isAbstract
  xmi.value (false | true) #REQUIRED
>
<!ELEMENT UML:GeneralizableElement.generalization
<!ELEMENT UML:GeneralizableElement (UML:ModelElement.name |
    UML:ModelElement.visibility |
    UML:GeneralizableElement.isRoot |
    UML:GeneralizableElement.isLeaf |
    UML:GeneralizableElement.isAbstract |
    XMI.extension |
    UML:ModelElement.binding |
    UML:ModelElement.template |
    UML:ModelElement.templateParameter |
    UML:ModelElement.implementation |
    UML:ModelElement.view |
    UML:ModelElement.presentation |
    UML:ModelElement.namespace |
    UML:ModelElement.constraint |
    UML:ModelElement.requirement |
    UML:ModelElement.provision |
    UML:ModelElement.stereotype |
    UML:ModelElement.elementReference |
    UML:ModelElement.collaboration |
    UML:ModelElement.behavior |
    UML:ModelElement.partition |
    UML:GeneralizableElement.generalization |
    UML:GeneralizableElement.specialization |
    UML:ModelElement.taggedValue |
    UML:Namespace.ownedElement)* >

<!ATTLIST UML:GeneralizableElement
    name CDATA #IMPLIED
    visibility (public | protected | private) #IMPLIED
    isRoot (false | true) #IMPLIED
    isLeaf (false | true) #IMPLIED
    isAbstract (false | true) #IMPLIED
    binding IDREFS #IMPLIED
    template IDREFS #IMPLIED
    templateParameter IDREFS #IMPLIED
    implementation IDREFS #IMPLIED
    view IDREFS #IMPLIED
    presentation IDREFS #IMPLIED
    namespace IDREFS #IMPLIED
    constraint IDREFS #IMPLIED
    requirement IDREFS #IMPLIED
    provision IDREFS #IMPLIED
    stereotype IDREFS #IMPLIED
    elementReference IDREFS #IMPLIED
    collaboration IDREFS #IMPLIED
    behavior IDREFS #IMPLIED
    partition IDREFS #IMPLIED
    generalization IDREFS #IMPLIED
    specialization IDREFS #IMPLIED
%XMI.element.att;
%XMI.link.att;
<!ELEMENT UML:Classifier.parameter (UML:Parameter)* >
<!ELEMENT UML:Classifier.structuralFeature (UML:StructuralFeature)* >
<!ELEMENT UML:Classifier.specification (UML:Classifier)* >
<!ELEMENT UML:Classifier.realization (UML:Classifier)* >
<!ELEMENT UML:Classifier.associationEnd (UML:AssociationEnd)* >
<!ELEMENT UML:Classifier.participant (UML:AssociationEnd)* >
<!ELEMENT UML:Classifier.createAction (UML:CreateAction)* >
<!ELEMENT UML:Classifier.instance (UML:Instance)* >
<!ELEMENT UML:Classifier.collaboration (UML:Collaboration)* >
<!ELEMENT UML:Classifier.classifierRole (UML:ClassifierRole)* >
<!ELEMENT UML:Classifier.classifierInState (UML:ClassifierInState)* >
<!ELEMENT UML:Classifier (UML:ModelElement.name |
UML:ModelElement.visibility |
UML:GeneralizableElement.isRoot |
UML:GeneralizableElement.isLeaf |
UML:GeneralizableElement.isAbstract |
XMI.extension |
UML:ModelElement.binding |
UML:ModelElement.template |
UML:ModelElement.templateParameter |
UML:ModelElement.implementation |
UML:ModelElement.view |
UML:ModelElement.presentation |
UML:ModelElement.namespace |
UML:ModelElement.constraint |
UML:ModelElement.requirement |
UML:ModelElement.provision |
UML:ModelElement.stereotype |
UML:ModelElement.elementReference |
UML:ModelElement.collaboration |
UML:ModelElement.behavior |
UML:ModelElement.partition |
UML:GeneralizableElement.generalization |
UML:GeneralizableElement.specialization |
UML:Classifier.parameter |
UML:Classifier.structuralFeature |
UML:Classifier.specification |
UML:Classifier.realization |
<!ATTLIST UML:Classifier
    name CDATA #IMPLIED
    visibility (public | protected | private) #IMPLIED
    isRoot (false | true) #IMPLIED
    isLeaf (false | true) #IMPLIED
    isAbstract (false | true) #IMPLIED
    binding IDREFS #IMPLIED
    template IDREFS #IMPLIED
    templateParameter IDREFS #IMPLIED
    implementation IDREFS #IMPLIED
    view IDREFS #IMPLIED
    presentation IDREFS #IMPLIED
    namespace IDREFS #IMPLIED
    constraint IDREFS #IMPLIED
    requirement IDREFS #IMPLIED
    provision IDREFS #IMPLIED
    stereotype IDREFS #IMPLIED
    elementReference IDREFS #IMPLIED
    ModelElement.collaboration IDREFS #IMPLIED
    behavior IDREFS #IMPLIED
    partition IDREFS #IMPLIED
    generalization IDREFS #IMPLIED
    specialization IDREFS #IMPLIED
    parameter IDREFS #IMPLIED
    structuralFeature IDREFS #IMPLIED
    specification IDREFS #IMPLIED
    realization IDREFS #IMPLIED
    associationEnd IDREFS #IMPLIED
    participant IDREFS #IMPLIED
    createAction IDREFS #IMPLIED
    instance IDREFS #IMPLIED
    Classifier.collaboration IDREFS #IMPLIED
    classifierRole IDREFS #IMPLIED
    classifierInState IDREFS #IMPLIED
    %XMI.element.att;
    %XMI.link.att;>

<!-- ______________________________________________________________________ -->
<!-- -->
<!-- CLASS: Foundation.Core.Interface -->
<!-- -->
<!ELEMENT UML:Interface (UML:ModelElement.name |
UML:GeneralizableElement.isRoot |
UML:GeneralizableElement.isLeaf |
UML:GeneralizableElement.isAbstract |
XML.extension | UML:ModelElement.binding |
UML:ModelElement.template |
UML:ModelElement.templateParameter |
UML:ModelElement.implementation |
UML:ModelElement.view |
UML:ModelElement.presentation |
UML:ModelElement.namespace |
UML:ModelElement.constraint |
UML:ModelElement.requirement |
UML:ModelElement.provision |
UML:ModelElement.stereotype |
UML:ModelElement.elementReference |
UML:ModelElement.collaboration |
UML:ModelElement.behavior |
UML:ModelElement.partition |
UML:GeneralizableElement.generalization |
UML:GeneralizableElement.specialization |
UML:Classifier.parameter |
UML:Classifier.structuralFeature |
UML:Classifier.specification |
UML:Classifier.realization |
UML:Classifier.associationEnd |
UML:Classifier.participant |
UML:Classifier.createAction |
UML:Classifier.instance |
UML:Classifier.collaboration |
UML:Classifier.classifierRole |
UML:Classifier.classifierInState |
UML:ModelElement.taggedValue |
UML:Namespace.ownedElement |
UML:Classifier.feature)* >
<!ATTLIST UML:Interface
  name CDATA #IMPLIED
  visibility (public | protected | private) #IMPLIED
  isRoot (false | true) #IMPLIED
  isLeaf (false | true) #IMPLIED
  isAbstract (false | true) #IMPLIED
  binding IDREFS #IMPLIED
  template IDREFS #IMPLIED
  templateParameter IDREFS #IMPLIED
  implementation IDREFS #IMPLIED
  view IDREFS #IMPLIED
  presentation IDREFS #IMPLIED
  namespace IDREFS #IMPLIED
  constraint IDREFS #IMPLIED
  requirement IDREFS #IMPLIED
  provision IDREFS #IMPLIED
  stereotype IDREFS #IMPLIED
  elementReference IDREFS #IMPLIED
  ModelElement.collaboration IDREFS #IMPLIED
  behavior IDREFS #IMPLIED
  partition IDREFS #IMPLIED

generalization IDREFS #IMPLIED
specialization IDREFS #IMPLIED
parameter IDREFS #IMPLIED
structuralFeature IDREFS #IMPLIED
specification IDREFS #IMPLIED
realization IDREFS #IMPLIED
associationEnd IDREFS #IMPLIED
participant IDREFS #IMPLIED
createAction IDREFS #IMPLIED
instance IDREFS #IMPLIED
Classifier.collaboration IDREFS #IMPLIED
classifierRole IDREFS #IMPLIED
classifierInState IDREFS #IMPLIED
%XMI.element.att;
%XMI.link.att;
>
<!-- _______________________________________________________________ -->
<!-- -->
<!-- CLASS: Foundation.Core.Class -->
<!-- _______________________________________________________________ -->
<!ELEMENT UML:Class.isActive EMPTY >
<!ATTLIST UML:Class.isActive
  xmi.value (false | true) #REQUIRED
>
<!ELEMENT UML:Class (UML:ModelElement.name |
  UML:ModelElement.visibility |
  UML:GeneralizableElement.isRoot |
  UML:GeneralizableElement.isLeaf |
  UML:GeneralizableElement.isAbstract |
  UML:Class.isActive | XMI.extension |
  UML:ModelElement.binding |
  UML:ModelElement.template |
  UML:ModelElement.templateParameter |
  UML:ModelElement.implementation |
  UML:ModelElement.view |
  UML:ModelElement.presentation |
  UML:ModelElement.namespace |
  UML:ModelElement.constraint |
  UML:ModelElement.requirement |
  UML:ModelElement.provision |
  UML:ModelElement.stereotype |
  UML:ModelElement.elementReference |
  UML:ModelElement.collaboration |
  UML:ModelElement.behavior |
  UML:ModelElement.partition |
  UML:GeneralizableElement.generalization |
  UML:GeneralizableElement.specialization |
  UML:Classifier.parameter |
  UML:Classifier.structuralFeature |
  UML:Classifier.specification |
  UML:Classifier.realization |
  UML:Classifier.associationEnd |
<!ATTLIST UML:Class
    name CDATA #IMPLIED
    visibility (public | protected | private) #IMPLIED
    isRoot (false | true) #IMPLIED
    isLeaf (false | true) #IMPLIED
    isAbstract (false | true) #IMPLIED
    isActive (false | true) #IMPLIED
    binding IDREFS #IMPLIED
    template IDREFS #IMPLIED
    templateParameter IDREFS #IMPLIED
    implementation IDREFS #IMPLIED
    view IDREFS #IMPLIED
    presentation IDREFS #IMPLIED
    namespace IDREFS #IMPLIED
    constraint IDREFS #IMPLIED
    requirement IDREFS #IMPLIED
    provision IDREFS #IMPLIED
    stereotype IDREFS #IMPLIED
    elementReference IDREFS #IMPLIED
    ModelElement.collaboration IDREFS #IMPLIED
    behavior IDREFS #IMPLIED
    partition IDREFS #IMPLIED
    generalization IDREFS #IMPLIED
    specialization IDREFS #IMPLIED
    parameter IDREFS #IMPLIED
    structuralFeature IDREFS #IMPLIED
    specification IDREFS #IMPLIED
    realization IDREFS #IMPLIED
    associationEnd IDREFS #IMPLIED
    participant IDREFS #IMPLIED
    createAction IDREFS #IMPLIED
    instance IDREFS #IMPLIED
    Classifier.collaboration IDREFS #IMPLIED
    classifierRole IDREFS #IMPLIED
    classifierInState IDREFS #IMPLIED
%XMI.element.att;
%XMI.link.att;
>
<!-- _______________________________________________________________ -->
<!-- --->
<!--
    CLASS: Foundation.Core.DataType
-->
<!-- _______________________________________________________________ -->
<!ELEMENT UML:DataType (UML:ModelElement.name |
<!ATTLIST UML:DataType
    name CDATA #IMPLIED
    visibility (public | protected | private) #IMPLIED
    isRoot (false | true) #IMPLIED
    isLeaf (false | true) #IMPLIED
    isAbstract (false | true) #IMPLIED
    binding IDREFS #IMPLIED
    template IDREFS #IMPLIED
    templateParameter IDREFS #IMPLIED
    implementation IDREFS #IMPLIED
    view IDREFS #IMPLIED
    presentation IDREFS #IMPLIED
    namespace IDREFS #IMPLIED
    constraint IDREFS #IMPLIED
    requirement IDREFS #IMPLIED
    provision IDREFS #IMPLIED
    stereotype IDREFS #IMPLIED
    elementReference IDREFS #IMPLIED
    ModelElement.collaboration IDREFS #IMPLIED
    behavior IDREFS #IMPLIED
    partition IDREFS #IMPLIED>
generalization IDREFS #IMPLIED
specialization IDREFS #IMPLIED
parameter IDREFS #IMPLIED
structuralFeature IDREFS #IMPLIED
specification IDREFS #IMPLIED
realization IDREFS #IMPLIED
associationEnd IDREFS #IMPLIED
participant IDREFS #IMPLIED
createAction IDREFS #IMPLIED
instance IDREFS #IMPLIED
Classifier.collaboration IDREFS #IMPLIED
classifierRole IDREFS #IMPLIED
classifierInState IDREFS #IMPLIED
%XMI.element.att;
%XMI.link.att;

<!ELEMENT UML:Feature.ownerScope EMPTY >
<!ATTLIST UML:Feature.ownerScope
 xmi.value (classifier | instance) #REQUIRED

<!ELEMENT UML:Feature.owner (UML:Classifier)* >
<!ELEMENT UML:Feature.classifierRole (UML:ClassifierRole)* >
<!ELEMENT UML:Feature (UML:ModelElement.name |
    UML:ModelElement.visibility |
    UML:Feature.ownerScope | XMI.extension |
    UML:ModelElement.binding |
    UML:ModelElement.template |
    UML:ModelElement.templateParameter |
    UML:ModelElement.implementation |
    UML:ModelElement.view |
    UML:ModelElement.presentation |
    UML:ModelElement.namespace |
    UML:ModelElement.constraint |
    UML:ModelElement.requirement |
    UML:ModelElement.provision |
    UML:ModelElement.stereotype |
    UML:ModelElement.elementReference |
    UML:ModelElement.collaboration |
    UML:ModelElement.behavior |
    UML:ModelElement.partition | UML:Feature.owner |
    UML:Feature.classifierRole |
    UML:ModelElement.taggedValue)* >
<!ATTLIST UML:Feature
    name CDATA #IMPLIED
    visibility (public | protected | private) #IMPLIED
    ownerScope (classifier | instance) #IMPLIED
binding IDREFS #IMPLIED
template IDREFS #IMPLIED
templateParameter IDREFS #IMPLIED
implementation IDREFS #IMPLIED
view IDREFS #IMPLIED
presentation IDREFS #IMPLIED
namespace IDREFS #IMPLIED
constraint IDREFS #IMPLIED
requirement IDREFS #IMPLIED
provision IDREFS #IMPLIED
stereotype IDREFS #IMPLIED
elementReference IDREFS #IMPLIED
collaboration IDREFS #IMPLIED
behavior IDREFS #IMPLIED
partition IDREFS #IMPLIED
owner IDREFS #IMPLIED
classifierRole IDREFS #IMPLIED
%XMI.element.att;
%XMI.link.att;

<!ELEMENT UML:StructuralFeature.multiplicity (#PCDATA |
                             XMI.reference)* >

<!ELEMENT UML:StructuralFeature.changeable EMPTY >
<!ATTLIST UML:StructuralFeature.changeable
      xmi.value (none | frozen | addOnly) #REQUIRED

<!ELEMENT UML:StructuralFeature.targetScope EMPTY >
<!ATTLIST UML:StructuralFeature.targetScope
      xmi.value (classifier | instance) #REQUIRED

<!ELEMENT UML:StructuralFeature.type (UML:Classifier)* >

<!ELEMENT UML:StructuralFeature (UML:ModelElement.name |
                            UML:ModelElement.visibility |
                            UML:Feature.ownerScope |
                            UML:StructuralFeature.multiplicity |
                            UML:StructuralFeature.changeable |
                            UML:StructuralFeature.targetScope |
                            XMI.extension |
                            UML:ModelElement.binding |
                            UML:ModelElement.template |
                            UML:ModelElement.templateParameter |
                            UML:ModelElement.implementation |
                            UML:ModelElement.view |
                            UML:ModelElement.presentation |
                            UML:ModelElement.namespace |
<!ATTLIST UML:StructuralFeature
  name CDATA #IMPLIED
  visibility (public | protected | private) #IMPLIED
  ownerScope (classifier | instance) #IMPLIED
  multiplicity CDATA #IMPLIED
  changeable (none | frozen | addOnly) #IMPLIED
  targetScope (classifier | instance) #IMPLIED
  binding IDREFS #IMPLIED
  template IDREFS #IMPLIED
  templateParameter IDREFS #IMPLIED
  implementation IDREFS #IMPLIED
  view IDREFS #IMPLIED
  presentation IDREFS #IMPLIED
  namespace IDREFS #IMPLIED
  constraint IDREFS #IMPLIED
  requirement IDREFS #IMPLIED
  provision IDREFS #IMPLIED
  stereotype IDREFS #IMPLIED
  elementReference IDREFS #IMPLIED
  collaboration IDREFS #IMPLIED
  behavior IDREFS #IMPLIED
  partition IDREFS #IMPLIED
  owner IDREFS #IMPLIED
  classifierRole IDREFS #IMPLIED
  type IDREFS #IMPLIED
  namespace IDREFS #IMPLIED
  constraint IDREFS #IMPLIED
  requirement IDREFS #IMPLIED
  provision IDREFS #IMPLIED
  stereotype IDREFS #IMPLIED
  elementReference IDREFS #IMPLIED
  collaboration IDREFS #IMPLIED
  behavior IDREFS #IMPLIED
  partition IDREFS #IMPLIED
  owner IDREFS #IMPLIED
  classifierRole IDREFS #IMPLIED
  type IDREFS #IMPLIED
  %XMI.element.att;
  %XMI.link.att;
>
<!-- _______________________________________________________________ -->
<!-- -->
<!-- CLASS: Foundation.Core.BehavioralFeature -->
<!-- _______________________________________________________________ -->

<!ELEMENT UML:BehavioralFeature.isQuery EMPTY >
<!ATTLIST UML:BehavioralFeature.isQuery
  xmi.value (false | true) #REQUIRED
>
<!ELEMENT UML:BehavioralFeature.raisedException (UML:Exception)* >
<!ELEMENT UML:BehavioralFeature (UML:ModelElement.name |
<!ATTLIST UML:BehavioralFeature
    name CDATA #IMPLIED
    visibility (public | protected | private) #IMPLIED
    ownerScope (classifier | instance) #IMPLIED
    isQuery (false | true) #IMPLIED
    binding IDREFS #IMPLIED
    template IDREFS #IMPLIED
    templateParameter IDREFS #IMPLIED
    implementation IDREFS #IMPLIED
    view IDREFS #IMPLIED
    presentation IDREFS #IMPLIED
    namespace IDREFS #IMPLIED
    constraint IDREFS #IMPLIED
    requirement IDREFS #IMPLIED
    provision IDREFS #IMPLIED
    stereotype IDREFS #IMPLIED
    elementReference IDREFS #IMPLIED
    collaboration IDREFS #IMPLIED
    behavior IDREFS #IMPLIED
    partition IDREFS #IMPLIED
    owner IDREFS #IMPLIED
    classifierRole IDREFS #IMPLIED
    raisedException IDREFS #IMPLIED
%XMI.element.att;
%XMI.link.att;
>

<!-- _______________________________________________________________ -->
<!-- -->
<!-- _______________________________________________________________ -->

<!-- CLASS: Foundation.Core.Operation -->
<!-- -->
<!-- _______________________________________________________________ -->
<!ELEMENT UML:Operation.specification (#PCDATA | XMI.reference)* >

<!ELEMENT UML:Operation.isPolymorphic EMPTY >
<!ATTLIST UML:Operation.isPolymorphic
   xmi.value (false | true) #REQUIRED
>

<!ELEMENT UML:Operation.concurrency EMPTY >
<!ATTLIST UML:Operation.concurrency
   xmi.value (sequential | guarded | concurrent) #REQUIRED
>

<!ELEMENT UML:Operation.method (UML:Method)* >

<!ELEMENT UML:Operation.collaboration (UML:Collaboration)* >

<!ELEMENT UML:Operation.occurrence (UML:CallEvent)* >

<!ELEMENT UML:Operation (UML:ModelElement.name |
   UML:ModelElement.visibility |
   UML:Feature.ownerScope |
   UML:BehavioralFeature.isQuery |
   UML:Operation.specification |
   UML:Operation.isPolymorphic |
   UML:Operation.concurrency |
   XMI.extension |
   UML:ModelElement.binding |
   UML:ModelElement.template |
   UML:ModelElement.templateParameter |
   UML:ModelElement.implementation |
   UML:ModelElement.view |
   UML:ModelElement.presentation |
   UML:ModelElement.namespace |
   UML:ModelElement.constraint |
   UML:ModelElement.requirement |
   UML:ModelElement.provision |
   UML:ModelElement.stereotype |
   UML:ModelElement.elementReference |
   UML:ModelElement.collaboration |
   UML:ModelElement.behavior |
   UML:ModelElement.partition | UML:Feature.owner |
   UML:Feature.classifierRole |
   UML:BehavioralFeature.raisedException |
   UML:Request.action |
   UML:Request.messageInstance |
   UML:Operation.method |
   UML:Operation.collaboration |
   UML:Operation.occurrence |
   UML:ModelElement.taggedValue |
   UML:BehavioralFeature.parameter)* >

<!ATTLIST UML:Operation
   name CDATA #IMPLIED
   visibility (public | protected | private) #IMPLIED
   ownerScope (classifier | instance) #IMPLIED
   isQuery (false | true) #IMPLIED
   specification CDATA #IMPLIED
isPolymorphic (false | true) #IMPLIED
concurrency (sequential | guarded | concurrent) #IMPLIED
binding IDREFS #IMPLIED
template IDREFS #IMPLIED
templateParameter IDREFS #IMPLIED
implementation IDREFS #IMPLIED
view IDREFS #IMPLIED
presentation IDREFS #IMPLIED
namespace IDREFS #IMPLIED
constraint IDREFS #IMPLIED
requirement IDREFS #IMPLIED
provision IDREFS #IMPLIED
stereotype IDREFS #IMPLIED
elementReference IDREFS #IMPLIED
ModelElement.collaboration IDREFS #IMPLIED
behavior IDREFS #IMPLIED
partition IDREFS #IMPLIED
owner IDREFS #IMPLIED
classifierRole IDREFS #IMPLIED
raisedException IDREFS #IMPLIED
action IDREFS #IMPLIED
messageInstance IDREFS #IMPLIED
method IDREFS #IMPLIED
Operation.collaboration IDREFS #IMPLIED
occurrence IDREFS #IMPLIED
%XMI.element.att;
%XMI.link.att;

</!

<!ELEMENT UML:Method.body (UML:ProcedureExpression) >
<!ELEMENT UML:Method.specification (UML:Operation)* >
<!ELEMENT UML:Method (UML:ModelElement.name |
    UML:ModelElement.visibility |
    UML:Feature.ownerScope |
    UML:BehavioralFeature.isQuery | UML:Method.body |
    XMI.extension | UML:ModelElement.binding |
    UML:ModelElement.template |
    UML:ModelElement.templateParameter |
    UML:ModelElement.implementation |
    UML:ModelElement.view |
    UML:ModelElement.presentation |
    UML:ModelElement.namespace |
    UML:ModelElement.constraint |
    UML:ModelElement.requirement |
    UML:ModelElement.provision |
    UML:ModelElement.stereotype |
    UML:ModelElement.elementReference |
    UML:ModelElement.collaboration |>
<!ATTLIST UML:Method
  name CDATA #IMPLIED
  visibility (public | protected | private) #IMPLIED
  ownerScope (classifier | instance) #IMPLIED
  isQuery (false | true) #IMPLIED
  binding IDREFS #IMPLIED
  template IDREFS #IMPLIED
  templateParameter IDREFS #IMPLIED
  implementation IDREFS #IMPLIED
  view IDREFS #IMPLIED
  presentation IDREFS #IMPLIED
  namespace IDREFS #IMPLIED
  constraint IDREFS #IMPLIED
  requirement IDREFS #IMPLIED
  provision IDREFS #IMPLIED
  stereotype IDREFS #IMPLIED
  elementReference IDREFS #IMPLIED
  collaboration IDREFS #IMPLIED
  behavior IDREFS #IMPLIED
  partition IDREFS #IMPLIED
  owner IDREFS #IMPLIED
  classifierRole IDREFS #IMPLIED
  raisedException IDREFS #IMPLIED
  specification IDREFS #IMPLIED
  %XMI.element.att;
  %XMI.link.att;
>
<!ELEMENT UML:Parameter.defaultValue (UML:Expression |
  UML:ProcedureExpression |
  UML:ObjectSetExpression |
  UML:TimeExpression |
  UML:BooleanExpression)>
<!ELEMENT UML:Parameter.kind EMPTY >
<!ATTLIST UML:Parameter.kind
  xmi.value (in | inout | out | return) #REQUIRED
>
<!ELEMENT UML:Parameter.behavioralFeature (UML:BehavioralFeature)* >
<!ELEMENT UML:Parameter.type (UML:Classifier)* >
<!ELEMENT UML:Parameter.signal (UML:Signal)* >


<!ATTLIST UML:Parameter
    name CDATA #IMPLIED
    visibility (public | protected | private) #IMPLIED
    kind (in | inout | out | return) #IMPLIED
    binding IDREFS #IMPLIED
    template IDREFS #IMPLIED
    templateParameter IDREFS #IMPLIED
    implementation IDREFS #IMPLIED
    view IDREFS #IMPLIED
    presentation IDREFS #IMPLIED
    namespace IDREFS #IMPLIED
    constraint IDREFS #IMPLIED
    requirement IDREFS #IMPLIED
    provision IDREFS #IMPLIED
    stereotype IDREFS #IMPLIED
    elementReference IDREFS #IMPLIED
    collaboration IDREFS #IMPLIED
    behavior IDREFS #IMPLIED
    partition IDREFS #IMPLIED
    behavioralFeature IDREFS #IMPLIED
    type IDREFS #IMPLIED
    signal IDREFS #IMPLIED
%XMI.element.att;
%XMI.link.att;
>
<!-- _______________________________________________________________ -->
<!-- CLASS: Foundation.Core.Constraint -->
<!-- _______________________________________________________________ -->

<!ELEMENT UML:Constraint.body (UML:BooleanExpression) >
<!ELEMENT UML:Constraint.constrainedElement (UML:ModelElement)* >
<!ELEMENT UML:Constraint.constrainedStereotype (UML:Stereotype)* >
<!ATTLIST UML:Constraint
 name CDATA #IMPLIED
 visibility (public | protected | private) #IMPLIED
 binding IDREFS #IMPLIED
 template IDREFS #IMPLIED
 templateParameter IDREFS #IMPLIED
 implementation IDREFS #IMPLIED
 view IDREFS #IMPLIED
 presentation IDREFS #IMPLIED
 namespace IDREFS #IMPLIED
 constraint IDREFS #IMPLIED
 requirement IDREFS #IMPLIED
 provision IDREFS #IMPLIED
 stereotype IDREFS #IMPLIED
 elementReference IDREFS #IMPLIED
 collaboration IDREFS #IMPLIED
 behavior IDREFS #IMPLIED
 partition IDREFS #IMPLIED
 constrainedElement IDREFS #IMPLIED
 constrainedStereotype IDREFS #IMPLIED
 %XMI.element.att; %XMI.link.att; >
<!ELEMENT UML:Dependency.description (#PCDATA | XMI.reference)* >

<!ELEMENT UML:Dependency.owningDependency (UML:Dependency)* >

<!ELEMENT UML:Dependency.client (UML:ModelElement)* >

<!ELEMENT UML:Dependency.supplier (UML:ModelElement)* >

<!ELEMENT UML:Dependency (UML:ModelElement.name |
UML:ModelElement.visibility |
UML:Dependency.description | XMI.extension |
UML:ModelElement.binding |
UML:ModelElement.template |
UML:ModelElement.templateParameter |
UML:ModelElement.implementation |
UML:ModelElement.view |
UML:ModelElement.presentation |
UML:ModelElement.namespase |
UML:ModelElement.constraint |
UML:ModelElement.requirement |
UML:ModelElement.provision |
UML:ModelElement.stereotype |
UML:ModelElement.elementReference |
UML:ModelElement.collaboration |
UML:ModelElement.behavior |
UML:ModelElement.partition |
UML:Dependency.owningDependency |
UML:Dependency.client |
UML:Dependency.supplier |
UML:ModelElement.taggedValue |
UML:Dependency.subDependencies)* >

<!ATTLIST UML:Dependency
name CDATA #IMPLIED
visibility (public | protected | private) #IMPLIED
description CDATA #IMPLIED
binding IDREFS #IMPLIED
template IDREFS #IMPLIED
templateParameter IDREFS #IMPLIED
implementation IDREFS #IMPLIED
view IDREFS #IMPLIED
presentation IDREFS #IMPLIED
namespace IDREFS #IMPLIED
constraint IDREFS #IMPLIED
requirement IDREFS #IMPLIED
provision IDREFS #IMPLIED
stereotype IDREFS #IMPLIED
elementReference IDREFS #IMPLIED
collaboration IDREFS #IMPLIED
behavior IDREFS #IMPLIED
partition IDREFS #IMPLIED
owningDependency IDREFS #IMPLIED
client IDREFS #IMPLIED
supplier IDREFS #IMPLIED
%XMI.element.att;
%XMI.link.att;
<!-- ________________________________ -->
<!-- -->
<!-- CLASS: Foundation.Core.Generalization -->
<!-- ________________________________ -->

<!ELEMENT UML:Generalization.discriminator (#PCDATA | XMI.reference)* >
<!ELEMENT UML:Generalization.subtype (UML:GeneralizableElement)* >
<!ELEMENT UML:Generalization.supertype (UML:GeneralizableElement)* >
<!ELEMENT UML:Generalization (UML:ModelElement.name |
  UML:ModelElement.visibility |
  UML:Generalization.discriminator |
  XML.extension | UML:ModelElement.binding |
  UML:ModelElement.template |
  UML:ModelElement.templateParameter |
  UML:ModelElement.implementation |
  UML:ModelElement.view |
  UML:ModelElement.presentation |
  UML:ModelElement.namespace |
  UML:ModelElement.constraint |
  UML:ModelElement.requirement |
  UML:ModelElement.provision |
  UML:ModelElement.stereotype |
  UML:ModelElement.elementReference |
  UML:ModelElement.collaboration |
  UML:ModelElement.behavior |
  UML:ModelElement.partition |
  UML:Generalization.subtype |
  UML:Generalization.supertype |
  UML:ModelElement.taggedValue)* >

<!ATTLIST UML:Generalization
  name CDATA #IMPLIED
  visibility (public | protected | private) #IMPLIED
  discriminator CDATA #IMPLIED
  binding IDREFS #IMPLIED
  template IDREFS #IMPLIED
  templateParameter IDREFS #IMPLIED
  implementation IDREFS #IMPLIED
  view IDREFS #IMPLIED
  presentation IDREFS #IMPLIED
  namespace IDREFS #IMPLIED
  constraint IDREFS #IMPLIED
  requirement IDREFS #IMPLIED
  provision IDREFS #IMPLIED
  stereotype IDREFS #IMPLIED
  elementReference IDREFS #IMPLIED
  collaboration IDREFS #IMPLIED
  behavior IDREFS #IMPLIED
  partition IDREFS #IMPLIED
  subtype IDREFS #IMPLIED
  supertype IDREFS #IMPLIED
<!ELEMENT UML:AssociationEnd.isNavigable EMPTY >
<!ATTLIST UML:AssociationEnd.isNavigable
  xmi.value (false | true) #REQUIRED
>
<!ELEMENT UML:AssociationEnd.isOrdered EMPTY >
<!ATTLIST UML:AssociationEnd.isOrdered
  xmi.value (false | true) #REQUIRED
>
<!ELEMENT UML:AssociationEnd.aggregation EMPTY >
<!ATTLIST UML:AssociationEnd.aggregation
  xmi.value (none | shared | composite) #REQUIRED
>
<!ELEMENT UML:AssociationEnd.multiplicity (#PCDATA | XMI.reference)* >
<!ELEMENT UML:AssociationEnd.changeable EMPTY >
<!ATTLIST UML:AssociationEnd.changeable
  xmi.value (none | frozen | addOnly) #REQUIRED
>
<!ELEMENT UML:AssociationEnd.targetScope EMPTY >
<!ATTLIST UML:AssociationEnd.targetScope
  xmi.value (classifier | instance) #REQUIRED
>
<!ELEMENT UML:AssociationEnd.type (UML:Classifier)* >
<!ELEMENT UML:AssociationEnd.specification (UML:Classifier)* >
<!ELEMENT UML:AssociationEnd.association (UML:Association)* >
<!ELEMENT UML:AssociationEnd.linkEnd (UML:LinkEnd)* >
<!ELEMENT UML:AssociationEnd.associationEndRole
  (UML:AssociationEndRole)* >
<!ELEMENT UML:AssociationEnd (UML:ModelElement.name |
  UML:ModelElement.visibility | UML:AssociationEnd.isNavigable |
  UML:AssociationEnd.isOrdered | UML:AssociationEnd.aggregation |
  UML:AssociationEnd.multiplicity | UML:AssociationEnd.changeable |
  UML:AssociationEnd.targetScope |
<!ATTLIST UML:AssociationEnd
  name CDATA #IMPLIED
  visibility (public | protected | private) #IMPLIED
  isNavigable (false | true) #IMPLIED
  isOrdered (false | true) #IMPLIED
  aggregation (none | shared | composite) #IMPLIED
  multiplicity CDATA #IMPLIED
  changeable (none | frozen | addOnly) #IMPLIED
  targetScope (classifier | instance) #IMPLIED
  binding IDREFS #IMPLIED
  template IDREFS #IMPLIED
  templateParameter IDREFS #IMPLIED
  implementation IDREFS #IMPLIED
  view IDREFS #IMPLIED
  presentation IDREFS #IMPLIED
  namespace IDREFS #IMPLIED
  constraint IDREFS #IMPLIED
  requirement IDREFS #IMPLIED
  provision IDREFS #IMPLIED
  stereotype IDREFS #IMPLIED
  elementReference IDREFS #IMPLIED
  collaboration IDREFS #IMPLIED
  behavior IDREFS #IMPLIED
  partition IDREFS #IMPLIED
  type IDREFS #IMPLIED
  specification IDREFS #IMPLIED
  association IDREFS #IMPLIED
  linkEnd IDREFS #IMPLIED
  associationEndRole IDREFS #IMPLIED
%XMI.extension att;
%XMI.link att;
<!ELEMENT UML:Association (UML:ModelElement.name |
    UML:ModelElement.visibility | UML:GeneralizableElement.isRoot |
    UML:GeneralizableElement.isLeaf | UML:GeneralizableElement.isAbstract |
    XML:extension | UML:ModelElement.binding |
    UML:ModelElement.template | UML:ModelElement.templateParameter |
    UML:ModelElement.implementation | UML:ModelElement.view |
    UML:ModelElement.presentation | UML:ModelElement.namespace |
    UML:ModelElement.constraint | UML:ModelElement.requirement |
    UML:ModelElement.provision | UML:ModelElement.stereotype |
    UML:ModelElement.elementReference |
    UML:ModelElement.collaboration |
    UML:ModelElement.behavior |
    UML:ModelElement.partition |
    UML:GeneralizableElement.generalization |
    UML:GeneralizableElement.specialization |
    UML:Association.link |
    UML:Association.associationEnd |
    UML:ModelElement.taggedValue |
    UML:Namespace.ownedElement |
    UML:Association.connection)* >

<!ATTLIST UML:Association
    name CDATA #IMPLIED
    visibility (public | protected | private) #IMPLIED
    isRoot (false | true) #IMPLIED
    isLeaf (false | true) #IMPLIED
    isAbstract (false | true) #IMPLIED
    binding IDREFS #IMPLIED
    template IDREFS #IMPLIED
    templateParameter IDREFS #IMPLIED
    implementation IDREFS #IMPLIED
    view IDREFS #IMPLIED
    presentation IDREFS #IMPLIED
    namespace IDREFS #IMPLIED
    constraint IDREFS #IMPLIED
    requirement IDREFS #IMPLIED
    provision IDREFS #IMPLIED
    stereotype IDREFS #IMPLIED
    elementReference IDREFS #IMPLIED
    collaboration IDREFS #IMPLIED
behavior IDREFS #IMPLIED
partition IDREFS #IMPLIED
generalization IDREFS #IMPLIED
specialization IDREFS #IMPLIED
link IDREFS #IMPLIED
associationEnd IDREFS #IMPLIED
%XMI.element.att;
%XMI.link.att;
>
<!-- ___________________________ -->
<!-- CLASS: Foundation.Core.AssociationClass -->
<!-- ___________________________ -->
<!ATTLIST UML:AssociationClass
    name CDATA #IMPLIED
    visibility (public | protected | private) #IMPLIED
    isRoot (false | true) #IMPLIED
    isLeaf (false | true) #IMPLIED
    isAbstract (false | true) #IMPLIED
    isActive (false | true) #IMPLIED
    binding IDREFS #IMPLIED
    template IDREFS #IMPLIED
    templateParameter IDREFS #IMPLIED
    implementation IDREFS #IMPLIED
    view IDREFS #IMPLIED
    presentation IDREFS #IMPLIED
    namespace IDREFS #IMPLIED
    constraint IDREFS #IMPLIED
    requirement IDREFS #IMPLIED
    provision IDREFS #IMPLIED
    stereotype IDREFS #IMPLIED
    elementReference IDREFS #IMPLIED
    ModelElement.collection IDREFS #IMPLIED
    behavior IDREFS #IMPLIED
    partition IDREFS #IMPLIED
    generalization IDREFS #IMPLIED
    specialization IDREFS #IMPLIED
    parameter IDREFS #IMPLIED
    structuralFeature IDREFS #IMPLIED
    specification IDREFS #IMPLIED
    realization IDREFS #IMPLIED
    Classifier.associationEnd IDREFS #IMPLIED
    participant IDREFS #IMPLIED
    createAction IDREFS #IMPLIED
    instance IDREFS #IMPLIED
    Classifier.collection IDREFS #IMPLIED
    classifierRole IDREFS #IMPLIED
    classifierInState IDREFS #IMPLIED
    link IDREFS #IMPLIED
    Association.associationEnd IDREFS #IMPLIED
    %XMI.multiValuedAtt;
    %XMI.multiValuedAtt;
>
<!ELEMENT UML:Attribute.initialValue (UML:Expression |
    UML:ProcedureExpression |
    UML:ObjectSetExpression |
    UML:TimeExpression |
    UML:BooleanExpression) >

<!ELEMENT UML:Attribute.associationEnd (UML:AssociationEnd)* >

<!ELEMENT UML:Attribute.attributeLink (UML:AttributeLink)* >
<!ELEMENT UML:Attribute (UML:ModelElement.name |
    UML:ModelElement.visibility | 
    UML:Feature.ownerScope | 
    UML:StructuralFeature.multiplicity | 
    UML:StructuralFeature.changeable | 
    UML:StructuralFeature.targetScope | 
    UML:Attribute.initialValue | XMI.extension | 
    UML:ModelElement.binding | 
    UML:ModelElement.template | 
    UML:ModelElement.templateParameter | 
    UML:ModelElement.implementation | 
    UML:ModelElement.view | 
    UML:ModelElement.presentation | 
    UML:ModelElement.namespace | 
    UML:ModelElement.constraint | 
    UML:ModelElement.requirement | 
    UML:ModelElement.provision | 
    UML:ModelElement.stereotype | 
    UML:ModelElement.elementReference | 
    UML:ModelElement.collision | 
    UML:ModelElement.behavior | 
    UML:ModelElement.partition | UML:Feature.owner | 
    UML:Feature.classifierRole | 
    UML:StructuralFeature.type | 
    UML:Attribute.associationEnd | 
    UML:Attribute.attributeLink | 
    UML:ModelElement.taggedValue)* >

<!ATTLIST UML:Attribute
    name CDATA #IMPLIED
    visibility (public | protected | private) #IMPLIED
    ownerScope (classifier | instance) #IMPLIED
    multiplicity CDATA #IMPLIED
    changeable (none | frozen | addOnly) #IMPLIED
    targetScope (classifier | instance) #IMPLIED
    binding IDREFS #IMPLIED
    template IDREFS #IMPLIED
    templateParameter IDREFS #IMPLIED
    implementation IDREFS #IMPLIED
    view IDREFS #IMPLIED
    presentation IDREFS #IMPLIED
    namespace IDREFS #IMPLIED
    constraint IDREFS #IMPLIED
    requirement IDREFS #IMPLIED
    provision IDREFS #IMPLIED
    stereotype IDREFS #IMPLIED
    elementReference IDREFS #IMPLIED
    collaboration IDREFS #IMPLIED
    behavior IDREFS #IMPLIED
    partition IDREFS #IMPLIED
    owner IDREFS #IMPLIED
    classifierRole IDREFS #IMPLIED
    type IDREFS #IMPLIED
    associationEnd IDREFS #IMPLIED
    attributeLink IDREFS #IMPLIED
%XMI.element.att;
%XMI.link.att;

<!ELEMENT UML:TaggedValue.tag (#PCDATA | XMI.reference)* >
<!ELEMENT UML:TaggedValue.value (#PCDATA | XMI.reference)* >
<!ELEMENT UML:TaggedValue.modelElement (UML:ModelElement)* >
<!ELEMENT UML:TaggedValue.stereotype (UML:Stereotype)* >
<!ATTLIST UML:TaggedValue
tag CDATA #IMPLIED
value CDATA #IMPLIED
modelElement IDREFS #IMPLIED
stereotype IDREFS #IMPLIED
%XMI.element.att;
%XMI.link.att;

<!ELEMENT UML:Stereotype.icon (UML:Geometry) >
<!ELEMENT UML:Stereotype.baseClass (#PCDATA | XMI.reference)* >
<!ELEMENT UML:Stereotype.extendedElement (UML:ModelElement)* >
<!ELEMENT UML:Stereotype.stereotypeConstraint (UML:Constraint)* >
<!ATTLIST UML:Stereotype
    name CDATA #IMPLIED
    visibility (public | protected | private) #IMPLIED
    isRoot (false | true) #IMPLIED
    isLeaf (false | true) #IMPLIED
    isAbstract (false | true) #IMPLIED
    baseClass CDATA #IMPLIED
    binding IDREFS #IMPLIED
    template IDREFS #IMPLIED
    templateParameter IDREFS #IMPLIED
    implementation IDREFS #IMPLIED
    view IDREFS #IMPLIED
    presentation IDREFS #IMPLIED
    namespace IDREFS #IMPLIED
    constraint IDREFS #IMPLIED
    requirement IDREFS #IMPLIED
    provision IDREFS #IMPLIED
    stereotype IDREFS #IMPLIED
    elementReference IDREFS #IMPLIED
    collaboration IDREFS #IMPLIED
    behavior IDREFS #IMPLIED
    partition IDREFS #IMPLIED
    generalization IDREFS #IMPLIED
    specialization IDREFS #IMPLIED
    extendedElement IDREFS #IMPLIED
    stereotypeConstraint IDREFS #IMPLIED
    %XMI.element.att;
    %XMI.link.att;>

<!ELEMENT UML:Enumeration (UML:ModelElement.name | UML:ModelElement.visibility | UML:GeneralizableElement.isRoot | UML:GeneralizableElement.isLeaf | UML:GeneralizableElement.isAbstract |
<!ATTLIST UML:Enumeration
  name CDATA #IMPLIED
  visibility (public | protected | private) #IMPLIED
  isRoot (false | true) #IMPLIED
  isLeaf (false | true) #IMPLIED
  isAbstract (false | true) #IMPLIED
  binding IDREFS #IMPLIED
  template IDREFS #IMPLIED
  templateParameter IDREFS #IMPLIED
  implementation IDREFS #IMPLIED
  view IDREFS #IMPLIED
  presentation IDREFS #IMPLIED
  namespace IDREFS #IMPLIED
  constraint IDREFS #IMPLIED
  requirement IDREFS #IMPLIED
  provision IDREFS #IMPLIED
  stereotype IDREFS #IMPLIED
  elementReference IDREFS #IMPLIED
  ModelElement.collaboration IDREFS #IMPLIED
  behavior IDREFS #IMPLIED
  partition IDREFS #IMPLIED
  generalization IDREFS #IMPLIED
  specialization IDREFS #IMPLIED>
parameter IDREFS #IMPLIED
structuralFeature IDREFS #IMPLIED
specification IDREFS #IMPLIED
realization IDREFS #IMPLIED
associationEnd IDREFS #IMPLIED
participant IDREFS #IMPLIED
createAction IDREFS #IMPLIED
instance IDREFS #IMPLIED
Classifier.collaboration IDREFS #IMPLIED
classifierRole IDREFS #IMPLIED
classifierInState IDREFS #IMPLIED
%XMI.element.att;
%XMI.link.att;

<!ATTLIST UML:EnumerationLiteral
    name CDATA #IMPLIED
    enumeration IDREFS #IMPLIED
%XMI.element.att;
%XMI.link.att;

<!-- _______________________________________________________________ -->
<!-- -->
<!--[CLASS: Foundation.Data_Types.Primitive]-->
<!-- _______________________________________________________________ -->
<!ELEMENT UML:Primitive (UML:ModelElement.name |
    UML:ModelElement.visibility |
    UML:GeneralizableElement.isRoot |
    UML:GeneralizableElement.isLeaf |
    UML:GeneralizableElement.isAbstract |
    XMI.extension | UML:ModelElement.binding |
    UML:ModelElement.template |
    UML:ModelElement.templateParameter |
    UML:ModelElement.implementation |
    UML:ModelElement.view |
    UML:ModelElement.presentation |
    UML:ModelElement.namespace |
    UML:ModelElement.constraint |
    UML:ModelElementrequirement |
    UML:ModelElement.provision |
    UML:ModelElement.stereotype |
<!ATTLIST UML:Primitive
    name CDATA #IMPLIED
    visibility (public | protected | private) #IMPLIED
    isRoot (false | true) #IMPLIED
    isLeaf (false | true) #IMPLIED
    isAbstract (false | true) #IMPLIED
    binding IDREFS #IMPLIED
    template IDREFS #IMPLIED
    templateParameter IDREFS #IMPLIED
    implementation IDREFS #IMPLIED
    view IDREFS #IMPLIED
    presentation IDREFS #IMPLIED
    namespace IDREFS #IMPLIED
    constraint IDREFS #IMPLIED
    requirement IDREFS #IMPLIED
    provision IDREFS #IMPLIED
    stereotype IDREFS #IMPLIED
    elementReference IDREFS #IMPLIED
    ModelElement.elementReference IDREFS #IMPLIED
    ModelElement.collaboration IDREFS #IMPLIED
    ModelElement.behavior IDREFS #IMPLIED
    ModelElement.partition IDREFS #IMPLIED
    GeneralizableElement.generalization IDREFS #IMPLIED
    GeneralizableElement.specialization IDREFS #IMPLIED
    Classifier.parameter IDREFS #IMPLIED
    Classifier.structuralFeature IDREFS #IMPLIED
    Classifier.specification IDREFS #IMPLIED
    Classifier.realization IDREFS #IMPLIED
    Classifier.associationEnd IDREFS #IMPLIED
    Classifier.participant IDREFS #IMPLIED
    Classifier.createClassAction IDREFS #IMPLIED
    Classifier.instance IDREFS #IMPLIED
    Classifier.classification IDREFS #IMPLIED
    Classifier.classifierRole IDREFS #IMPLIED
    Classifier.classifierInState IDREFS #IMPLIED
    ModelElement.taggedValue IDREFS #IMPLIED
    Namespace.ownedElement IDREFS #IMPLIED
    Classifier.feature)* >

%XMI.element.att;
presentation IDREFS #IMPLIED
namespace IDREFS #IMPLIED
constraint IDREFS #IMPLIED
requirement IDREFS #IMPLIED
provision IDREFS #IMPLIED
stereotype IDREFS #IMPLIED
elementReference IDREFS #IMPLIED
ModelElement.collaboration IDREFS #IMPLIED
behavior IDREFS #IMPLIED
partition IDREFS #IMPLIED
generalization IDREFS #IMPLIED
specialization IDREFS #IMPLIED
parameter IDREFS #IMPLIED
structuralFeature IDREFS #IMPLIED
specification IDREFS #IMPLIED
realization IDREFS #IMPLIED
associationEnd IDREFS #IMPLIED
participant IDREFS #IMPLIED
createAction IDREFS #IMPLIED
instance IDREFS #IMPLIED
Classifier.collaboration IDREFS #IMPLIED
classifierRole IDREFS #IMPLIED
classifierInState IDREFS #IMPLIED
%XMI.element.att;
%XMI.link.att;

<!-- _______________________________________________________________ -->
<!-- -->
<!-- CLASS: Foundation.Data_Types.MultiplicityRange -->
<!-- _______________________________________________________________ -->
<!ELEMENT UML:MultiplicityRange.lower (#PCDATA | XMI.reference)* >
<!ELEMENT UML:MultiplicityRange.upper (#PCDATA | XMI.reference)* >
<!ELEMENT UML:MultiplicityRange (UML:MultiplicityRange.lower | UML:MultiplicityRange.upper | XMI.extension)* >
<!ATTLIST UML:MultiplicityRange
  lower CDATA #IMPLIED
  upper CDATA #IMPLIED
  %XMI.element.att;
  %XMI.link.att;
>

<!-- _______________________________________________________________ -->
<!-- -->
<!-- CLASS: Foundation.Data_Types.Geometry -->
<!-- _______________________________________________________________ -->
<!ELEMENT UML:Geometry.body (#PCDATA | XMI.reference)* >
<!ELEMENT UML:Geometry (UML:Geometry.body | XMI.extension)* >
<!ATTLIST UML:Geometry
  body CDATA #IMPLIED
  %XMI.element.att;
  %XMI.link.att;
>
<!ELEMENT UML:GraphicMarker.body (#PCDATA | XMI.reference)* >
<!ELEMENT UML:GraphicMarker (UML:GraphicMarker.body | XMI.extension)* >
<!ATTLIST UML:GraphicMarker
    body CDATA #IMPLIED
    %XMI.element.att;
    %XMI.link.att;
>
<!ELEMENT UML:Mapping.body (#PCDATA | XMI.reference)* >
<!ELEMENT UML:Mapping (UML:Mapping.body | XMI.extension)* >
<!ATTLIST UML:Mapping
    body CDATA #IMPLIED
    %XMI.element.att;
    %XMI.link.att;
>
<!ELEMENT UML:Expression.language (#PCDATA | XMI.reference)* >
<!ELEMENT UML:Expression.body (#PCDATA | XMI.reference)* >
<!ELEMENT UML:Expression (UML:Expression.language | UML:Expression.body | XMI.extension)* >
<!ATTLIST UML:Expression
    language CDATA #IMPLIED
    body CDATA #IMPLIED
    %XMI.element.att;
    %XMI.link.att;
>
<!-- _______________________________________________________________ -->

<!ELEMENT UML:Package.elementReference (UML:ElementReference)* >

<!ELEMENT UML:Package (UML:ModelElement.name |
UML:ModelElement.visibility |
UML:GeneralizableElement.isRoot |
UML:GeneralizableElement.isLeaf |
XMI.extension | UML:ModelElement.binding |
UML:ModelElement.template |
UML:ModelElement.templateParameter |
UML:ModelElement.implementation |
UML:ModelElement.view |
UML:ModelElement.presentation |
UML:ModelElement.namespace |
UML:ModelElement.constraint |
UML:ModelElement.requirement |
UML:ModelElement.provision |
UML:ModelElement.stereotype |
UML:ModelElement.elementReference |
UML:ModelElement.collaboration |
UML:ModelElement.behavior |
UML:ModelElement.partition |
UML:GeneralizableElement.generalization |
UML:GeneralizableElement.specialization |
UML:Package.elementReference |
UML:ModelElement.taggedValue |
UML:Namespace.ownedElement)* >

<!ATTLIST UML:Package
name CDATA #IMPLIED
visibility (public | protected | private) #IMPLIED
isRoot (false | true) #IMPLIED
isLeaf (false | true) #IMPLIED
isAbstract (false | true) #IMPLIED
binding IDREFS #IMPLIED
template IDREFS #IMPLIED
templateParameter IDREFS #IMPLIED
implementation IDREFS #IMPLIED
view IDREFS #IMPLIED
presentation IDREFS #IMPLIED
namespace IDREFS #IMPLIED
constraint IDREFS #IMPLIED
requirement IDREFS #IMPLIED
provision IDREFS #IMPLIED
stereotype IDREFS #IMPLIED
ModelElement.elementReference IDREFS #IMPLIED
collaboration IDREFS #IMPLIED
behavior IDREFS #IMPLIED
partition IDREFS #IMPLIED
generalization IDREFS #IMPLIED
specialization IDREFS #IMPLIED
Package.elementReference IDREFS #IMPLIED
%XMI.element.att;
%XMI.link.att;
<!ELEMENT UML:Subsystem.isInstantiable EMPTY >
<!ATTLIST UML:Subsystem.isInstantiable
xmi.value (false | true) #REQUIRED >

<!ELEMENT UML:Subsystem (UML:ModelElement.name |
UML:ModelElement.visibility |
UML:GeneralizableElement.isRoot |
UML:GeneralizableElement.isLeaf |
UML:GeneralizableElement.isAbstract |
UML:Subsystem.isInstantiable | XMI.extension |
UML:ModelElement.binding |
UML:ModelElement.template |
UML:ModelElement.templateParameter |
UML:ModelElement.implementation |
UML:ModelElement.view |
UML:ModelElement.presentation |
UML:ModelElement.namespace |
UML:ModelElement.constraint |
UML:ModelElement.requirement |
UML:ModelElement.provision |
UML:ModelElement.stereotype |
UML:ModelElement.elementReference |
UML:Classifier.parameter |
UML:Classifier.structuralFeature |
UML:Classifier.specification |
UML:Classifier.realization |
UML:Classifier.associationEnd |
UML:Classifier.participant |
UML:Classifier.createAction |
UML:Classifier.instance |
UML:Classifier.collaboration |
UML:Classifier.classifierRole |
UML:Classifier.classifierInState |
UML:Package.elementReference |
UML:ModelElement.taggedValue |
UML:Namespace.ownedElement |
UML:Classifier.feature)* >

<!ATTLIST UML:Subsystem
name CDATA #IMPLIED
visibility (public | protected | private) #IMPLIED
isRoot (false | true) #IMPLIED
isLeaf (false | true) #IMPLIED
isAbstract (false | true) #IMPLIED
isInstantiable (false | true) #IMPLIED
binding IDREFS #IMPLIED
template IDREFS #IMPLIED
templateParameter IDREFS #IMPLIED
implementation IDREFS #IMPLIED
view IDREFS #IMPLIED
presentation IDREFS #IMPLIED
namespace IDREFS #IMPLIED
constraint IDREFS #IMPLIED
requirement IDREFS #IMPLIED
provision IDREFS #IMPLIED
stereotype IDREFS #IMPLIED
ModelElement.elementReference IDREFS #IMPLIED
ModelElement.collaboration IDREFS #IMPLIED
behavior IDREFS #IMPLIED
partition IDREFS #IMPLIED
generalization IDREFS #IMPLIED
specialization IDREFS #IMPLIED
parameter IDREFS #IMPLIED
structuralFeature IDREFS #IMPLIED
specification IDREFS #IMPLIED
realization IDREFS #IMPLIED
associationEnd IDREFS #IMPLIED
participant IDREFS #IMPLIED
createAction IDREFS #IMPLIED
instance IDREFS #IMPLIED
Classifier.collaboration IDREFS #IMPLIED
classifierRole IDREFS #IMPLIED
classifierInState IDREFS #IMPLIED
Package.elementReference IDREFS #IMPLIED
%XMI.element.att;
%XMI.link.att;

<!ELEMENT UML:Model (UML:ModelElement.name |
UML:ModelElement.visibility |
UML:GeneralizableElement.isRoot |
UML:GeneralizableElement.isLeaf |
UML:GeneralizableElement.isAbstract |
XMI.extension |
UML:ModelElement.binding |
UML:ModelElement.template |
UML:ModelElement.templateParameter |
UML:ModelElement.implementation |
UML:ModelElement.view |
UML:ModelElement.presentation |
UML:ModelElement.namespace |
UML:ModelElement.constraint |
UML:ModelElement.requirement |
UML:ModelElement.provision |
<!ATTLIST UML:Model
  name CDATA #IMPLIED
  visibility (public | protected | private) #IMPLIED
  isRoot (false | true) #IMPLIED
  isLeaf (false | true) #IMPLIED
  isAbstract (false | true) #IMPLIED
  binding IDREFS #IMPLIED
  template IDREFS #IMPLIED
  templateParameter IDREFS #IMPLIED
  implementation IDREFS #IMPLIED
  view IDREFS #IMPLIED
  presentation IDREFS #IMPLIED
  namespace IDREFS #IMPLIED
  constraint IDREFS #IMPLIED
  requirement IDREFS #IMPLIED
  provision IDREFS #IMPLIED
  stereotype IDREFS #IMPLIED
  ModelElement.elementReference IDREFS #IMPLIED
  collaboration IDREFS #IMPLIED
  behavior IDREFS #IMPLIED
  partition IDREFS #IMPLIED
  generalization IDREFS #IMPLIED
  specialization IDREFS #IMPLIED
  Package.elementReference IDREFS #IMPLIED
%XMI.element.att;
%XMI.link.att;
>

<!-- ________________________________ -->
<!ELEMENT UML:ElementReference.visibility EMPTY >
<!ATTLIST UML:ElementReference.visibility
  xmi.value (public | protected | private) #REQUIRED
>
<!ELEMENT UML:ElementReference.alias (#PCDATA | XMI.reference)* >
<!ELEMENT UML:ElementReference.referencedElement (UML:ModelElement)* >
<!ELEMENT UML:ElementReference.package (UML:Package)* >
<!ELEMENT UML:ElementReference (UML:ElementReference.visibility |
elementReference IDREFS #IMPLIED
collaboration IDREFS #IMPLIED
behavior IDREFS #IMPLIED
partition IDREFS #IMPLIED
action IDREFS #IMPLIED
messageInstance IDREFS #IMPLIED
%XMI.element.att;
%XMI.link.att;

<!-- _______________________________________________________________ -->
<!-- -->
<!-- CLASS: Behavioral_Elements.Common_Behavior.Signal -->
<!-- _______________________________________________________________ -->

<!ELEMENT UML:Signal.reception (UML:Reception)* >
<!ELEMENT UML:Signal.occurrence (UML:SignalEvent)* >
<!ELEMENT UML:Signal (UML:ModelElement.name |
UML:ModelElement.visibility |
UML:GeneralizableElement.isRoot |
UML:GeneralizableElement.isLeaf |
UML:GeneralizableElement.isAbstract |
XML.extension | UML:ModelElement.binding |
UML:ModelElement.template |
UML:ModelElement.templateParameter |
UML:ModelElement.implementation |
UML:ModelElement.view |
UML:ModelElement.presentation |
UML:ModelElement.namespace |
UML:ModelElement.constraint |
UML:ModelElement.requirement |
UML:ModelElement.provision |
UML:ModelElement.stereotype |
UML:ModelElement.elementReference |
UML:ModelElement.collision |
UML:ModelElement.behavior |
UML:ModelElement.partition |
UML:GeneralizableElement.generalization |
UML:GeneralizableElement.specialization |
UML:Request.action | UML:Request.messageInstance |
UML:Signal.reception | UML:Signal.occurrence |
UML:ModelElement.taggedValue |
UML:Namespace.ownedElement |
UML:Signal.parameter)* >
<!ATTLIST UML:Signal
name CDATA #IMPLIED
visibility (public | protected | private) #IMPLIED
isRoot (false | true) #IMPLIED
isLeaf (false | true) #IMPLIED
isAbstract (false | true) #IMPLIED
binding IDREFS #IMPLIED
template IDREFS #IMPLIED
templateParameter IDREFS #IMPLIED

<!-- CLASS: Behavioral_Elements.Common_Behavior.Exception -->

<!-- _______________________________________________________________ -->

<!ELEMENT UML:Exception.context (UML:BehavioralFeature)* >

<!-- _______________________________________________________________ -->

<!-- _______________________________________________________________ -->

<!-- _______________________________________________________________ -->
<!ATTLIST UML:Exception
  name CDATA #IMPLIED
  visibility (public | protected | private) #IMPLIED
  isRoot (false | true) #IMPLIED
  isLeaf (false | true) #IMPLIED
  isAbstract (false | true) #IMPLIED
  binding IDREFS #IMPLIED
  template IDREFS #IMPLIED
  templateParameter IDREFS #IMPLIED
  implementation IDREFS #IMPLIED
  view IDREFS #IMPLIED
  presentation IDREFS #IMPLIED
  namespace IDREFS #IMPLIED
  constraint IDREFS #IMPLIED
  requirement IDREFS #IMPLIED
  provision IDREFS #IMPLIED
  stereotype IDREFS #IMPLIED
  elementReference IDREFS #IMPLIED
  collaboration IDREFS #IMPLIED
  behavior IDREFS #IMPLIED
  partition IDREFS #IMPLIED
  generalization IDREFS #IMPLIED
  specialization IDREFS #IMPLIED
  action IDREFS #IMPLIED
  messageInstance IDREFS #IMPLIED
  reception IDREFS #IMPLIED
  occurrence IDREFS #IMPLIED
  context IDREFS #IMPLIED
  %XMI.element.att;
  %XMI.link.att;
>

<!-- _______________________________ -->
<!-- CLASS: Behavioral_Elements.Common_Behavior.Reception -->
<!-- _______________________________ -->

<!ELEMENT UML:Reception.isPolymorphic EMPTY >
<!ATTLIST UML:Reception.isPolymorphic
  xmi.value (false | true) #REQUIRED
>
<!ELEMENT UML:Reception.specification (#PCDATA | XMI.reference)* >

<!ELEMENT UML:Reception.signal (UML:Signal)* >

<!ELEMENT UML:Reception (UML:ModelElement.name |
  UML:ModelElement.taggedValue |
  UML:Namespace.ownedElement |
  UML:Signal.parameter)* >

<!-- _______________________________ -->

<!ELEMENT UML:ModelElement (UML:ModelElement.taggedValue |
  UML:Namespace.ownedElement |
  UML:Signal.parameter)* >

<!ELEMENT UML:Namespace (UML:Namespace.taggedValue |
  UML:Namespace.ownedElement |
  UML:Namespace.namespace)* >

<!ELEMENT UML:Namespace.ownedElement (UML:ModelElement |
  UML:Namespace.ownedElement | XMI.extension)* >

<!ELEMENT UML:ModelElement (UML:ModelElement.taggedValue |
  UML:Namespace.ownedElement |
  UML:Signal.parameter)* >

<!ELEMENT UML:Signal (UML:ModelElement.name |
  UML:ModelElement.taggedValue |
  UML:Namespace.ownedElement |
  UML:Signal.parameter)* >

<!ELEMENT UML:Feature (UML:ModelElement.name |
  UML:ModelElement.taggedValue |
  UML:Namespace.ownedElement |
  UML:Signal.parameter)* >

<!ELEMENT UML:BehavioralFeature (UML:ModelElement.name |
  UML:ModelElement.taggedValue |
  UML:Namespace.ownedElement |
  UML:Signal.parameter)* >

<!ELEMENT UML:Reception.isPolymorphic EMPTY >
<!ATTLIST UML:Reception.isPolymorphic
  xmi.value (false | true) #REQUIRED
>
<!ELEMENT UML:Reception.specification (#PCDATA | XMI.reference)* >

<!ELEMENT UML:Reception.signal (UML:Signal)* >

<!ELEMENT UML:Reception (UML:ModelElement.name |
  UML:ModelElement.taggedValue |
  UML:Namespace.ownedElement |
  UML:Signal.parameter)* >

<!ELEMENT UML:Namespace (UML:Namespace.taggedValue |
  UML:Namespace.ownedElement |
  UML:Namespace.namespace)* >

<!ELEMENT UML:Namespace.ownedElement (UML:ModelElement |
  UML:Namespace.ownedElement | XMI.extension)* >

<!ELEMENT UML:ModelElement (UML:ModelElement.taggedValue |
  UML:Namespace.ownedElement |
  UML:Signal.parameter)* >

<!ELEMENT UML:Signal (UML:ModelElement.name |
  UML:ModelElement.taggedValue |
  UML:Namespace.ownedElement |
  UML:Signal.parameter)* >

<!ELEMENT UML:Feature (UML:ModelElement.name |
  UML:ModelElement.taggedValue |
  UML:Namespace.ownedElement |
  UML:Signal.parameter)* >

<!ELEMENT UML:BehavioralFeature (UML:ModelElement.name |
  UML:ModelElement.taggedValue |
  UML:Namespace.ownedElement |
  UML:Signal.parameter)* >

<!ELEMENT UML:Reception.isPolymorphic EMPTY >
<!ATTLIST UML:Reception.isPolymorphic
  xmi.value (false | true) #REQUIRED
>
<!ELEMENT UML:Reception.specification (#PCDATA | XMI.reference)* >

<!ELEMENT UML:Reception.signal (UML:Signal)* >

<!ELEMENT UML:Reception (UML:ModelElement.name |
  UML:ModelElement.taggedValue |
  UML:Namespace.ownedElement |
  UML:Signal.parameter)* >

<!ELEMENT UML:Namespace (UML:Namespace.taggedValue |
  UML:Namespace.ownedElement |
  UML:Namespace.namespace)* >

<!ELEMENT UML:Namespace.ownedElement (UML:ModelElement |
  UML:Namespace.ownedElement | XMI.extension)* >

<!ELEMENT UML:ModelElement (UML:ModelElement.taggedValue |
  UML:Namespace.ownedElement |
  UML:Signal.parameter)* >

<!ELEMENT UML:Signal (UML:ModelElement.name |
  UML:ModelElement.taggedValue |
  UML:Namespace.ownedElement |
  UML:Signal.parameter)* >

<!ELEMENT UML:Feature (UML:ModelElement.name |
  UML:ModelElement.taggedValue |
  UML:Namespace.ownedElement |
  UML:Signal.parameter)* >

<!ELEMENT UML:BehavioralFeature (UML:ModelElement.name |
  UML:ModelElement.taggedValue |
  UML:Namespace.ownedElement |
  UML:Signal.parameter)* >

<!ELEMENT UML:Argument.action (UML:Action)* >

<!ELEMENT UML:Argument (UML:Argument.value | XMI.extension | UML:Argument.action)* >

<!ATTLIST UML:Argument
    action IDREFS #IMPLIED
    %XMI.element.att;
    %XMI.link.att;
>

<!ELEMENT UML:ActionSequence.transition (UML:Transition)* >

<!ELEMENT UML:ActionSequence.state (UML:State)* >

<!ELEMENT UML:ActionSequence.state2 (UML:State)* >


<!ATTLIST UML:ActionSequence
    name CDATA #IMPLIED
    visibility (public | protected | private) #IMPLIED
    binding IDREFS #IMPLIED
    template IDREFS #IMPLIED
    templateParameter IDREFS #IMPLIED
    implementation IDREFS #IMPLIED
<!ELEMENT UML:Action.recurrence (UML:Expression |
  UML:ProcedureExpression |
  UML:ObjectSetExpression |
  UML:TimeExpression |
  UML:BooleanExpression) >

<!ELEMENT UML:Action.target (UML:ObjectSetExpression) >

<!ELEMENT UML:Action.isAsynchronous EMPTY >
<!ATTLIST UML:Action.isAsynchronous
  xmi.value (false | true) #REQUIRED
>

<!ELEMENT UML:Action.script (#PCDATA | XMI.reference)* >

<!ELEMENT UML:Action.request (UML:Request)* >

<!ELEMENT UML:Action.message (UML:Message)* >

<!ELEMENT UML:Action.actionSequence (UML:ActionSequence)* >

<!ELEMENT UML:Action (UML:ModelElement.name |
  UML:ModelElement.visibility |
  UML:Action.recurrence | UML:Action.target |
  UML:Action.isAsynchronous | UML:Action.script |
  XMI.extension | UML:ModelElement.binding |
  UML:ModelElement.template |
  UML:ModelElement.templateParameter |
  UML:ModelElement.implementation |
  UML:ModelElement.view |
  UML:ModelElement.presentation |
  UML:ModelElement.namespace |
<ATTLIST UML:Action
    name CDATA #IMPLIED
    visibility (public | protected | private) #IMPLIED
    isAsynchronous (false | true) #IMPLIED
    script CDATA #IMPLIED
    binding IDREFS #IMPLIED
    template IDREFS #IMPLIED
    templateParameter IDREFS #IMPLIED
    implementation IDREFS #IMPLIED
    view IDREFS #IMPLIED
    presentation IDREFS #IMPLIED
    namespace IDREFS #IMPLIED
    constraint IDREFS #IMPLIED
    requirement IDREFS #IMPLIED
    provision IDREFS #IMPLIED
    stereotype IDREFS #IMPLIED
    elementReference IDREFS #IMPLIED
    collaboration IDREFS #IMPLIED
    behavior IDREFS #IMPLIED
    partition IDREFS #IMPLIED
    request IDREFS #IMPLIED
    message IDREFS #IMPLIED
    actionSequence IDREFS #IMPLIED
>%

<!-- _______________________________________________________________ -->
<!-- -->
<!-- CLASS: Behavioral_Elements.Common_Behavior.CreateAction -->
<!-- _______________________________________________________________ -->

<!ELEMENT UML:CreateAction.instantiation (UML:Classifier)* >
<!ELEMENT UML:CreateAction (UML:ModelElement.name |
    UML:ModelElement.visibility |
    UML:Action.recurrence | UML:Action.target |
    UML:Action.isAsynchronous |
    UML:Action.script | XMI.extension |
    UML:ModelElement.binding |
    UML:ModelElement.template |
    UML:ModelElement.templateParameter |
    UML:ModelElement.implementation |
    UML:ModelElement.view |
<!ELEMENT UML:CallAction.mode EMPTY >
<!ATTLIST UML:CallAction.mode
  xmi.value (synchronous | asynchronous) #REQUIRED >

<!ELEMENT UML:CallAction.name EMPTY >
<!ATTLIST UML:CallAction.name
  name CDATA #IMPLIED
  visibility (public | protected | private) #IMPLIED
>
<!ATTLIST UML:CallAction
  name CDATA #IMPLIED
  visibility (public | protected | private) #IMPLIED
  isAsynchronous (false | true) #IMPLIED
  script CDATA #IMPLIED
  mode (synchronous | asynchronous) #IMPLIED
  binding IDREFS #IMPLIED
  template IDREFS #IMPLIED
  templateParameter IDREFS #IMPLIED
  implementation IDREFS #IMPLIED
  view IDREFS #IMPLIED
  presentation IDREFS #IMPLIED
  namespace IDREFS #IMPLIED
  constraint IDREFS #IMPLIED
  requirement IDREFS #IMPLIED
  provision IDREFS #IMPLIED
  stereotype IDREFS #IMPLIED
  elementReference IDREFS #IMPLIED
  collaboration IDREFS #IMPLIED
  behavior IDREFS #IMPLIED
  partition IDREFS #IMPLIED
  request IDREFS #IMPLIED
  message IDREFS #IMPLIED
  actionSequence IDREFS #IMPLIED
%XMI.element.att;
%XMI.link.att;
>
</!-- _______________________________________________________________ -->

<!-- _______________________________________________________________ -->

<!-- CLASS: Behavioral_Elements.Common_Behavior.LocalInvocation -->

<!-- _______________________________________________________________ -->

<!--  -->
<!--  -->
<!--  -->
<!ELEMENT UML:LocalInvocation (UML:ModelElement.name |
UML:ModelElement.visibility |
UML:Action.recurrence | UML:Action.target |
UML:Action.isAsynchronous |
UML:Action.script | XMI.extension |
UML:ModelElement.binding |
UML:ModelElement.template |
UML:ModelElement.templateParameter |
UML:ModelElement.implementation |
UML:ModelElement.view |
UML:ModelElement.presentation |
UML:ModelElement.namespace |
UML:ModelElement.constraint |
UML:ModelElement.requirement |
UML:ModelElement.provision |
UML:ModelElement.stereotype |
UML:ModelElement.elementReference |
UML:ModelElement.collaboration |
UML:ModelElement.behavior |
UML:ModelElement.partition |
UML:Action.request | UML:Action.message |
UML:Action.actionSequence |
UML:ModelElement.taggedValue |
UML:Action.actualArgument)* >

<!ATTLIST UML:LocalInvocation
name CDATA #IMPLIED
visibility (public | protected | private) #IMPLIED
isAsynchronous (false | true) #IMPLIED
script CDATA #IMPLIED
binding IDREFS #IMPLIED
template IDREFS #IMPLIED
templateParameter IDREFS #IMPLIED
implementation IDREFS #IMPLIED
view IDREFS #IMPLIED
presentation IDREFS #IMPLIED
namespace IDREFS #IMPLIED
constraint IDREFS #IMPLIED
requirement IDREFS #IMPLIED
provision IDREFS #IMPLIED
stereotype IDREFS #IMPLIED
elementReference IDREFS #IMPLIED
collaboration IDREFS #IMPLIED
behavior IDREFS #IMPLIED
partition IDREFS #IMPLIED
request IDREFS #IMPLIED
message IDREFS #IMPLIED
actionSequence IDREFS #IMPLIED
%XMI.element.att; | %XMI.link.att;>

<!-- _______________________________________________________________ -->
<!-- -->
<!-- CLASS: Behavioral_Elements.Common_Behavior.ReturnAction -->
<!-- _______________________________________________________________ -->
<!ELEMENT UML:ReturnAction (UML:ModelElement.name |
UML:ModelElement.visibility |
UML:Action.recurrence | UML:Action.target | 
UML:Action.isAsynchronous |
UML:Action.script | XMI.extension |
UML:ModelElement.binding |
UML:ModelElement.template |
UML:ModelElement.templateParameter |
UML:ModelElement.implementation |
UML:ModelElement.view |
UML:ModelElement.presentation |
UML:ModelElement.namespace |
UML:ModelElement.constraint |
UML:ModelElement.requirement |
UML:ModelElement.provision |
UML:ModelElement.stereotype |
UML:ModelElement.elementReference |
UML:ModelElement.collaboration |
UML:ModelElement.behavior |
UML:ModelElement.partition |
UML:Action.request | UML:Action.message |
UML:Action.actionSequence |
UML:ModelElement.taggedValue |
UML:Action.actualArgument)* >

<!ATTLIST UML:ReturnAction
name CDATA #IMPLIED
visibility (public | protected | private) #IMPLIED
isAsynchronous (false | true) #IMPLIED
script CDATA #IMPLIED
binding IDREFS #IMPLIED
template IDREFS #IMPLIED
templateParameter IDREFS #IMPLIED
implementation IDREFS #IMPLIED
view IDREFS #IMPLIED
presentation IDREFS #IMPLIED
namespace IDREFS #IMPLIED
constraint IDREFS #IMPLIED
requirement IDREFS #IMPLIED
provision IDREFS #IMPLIED
stereotype IDREFS #IMPLIED
elementReference IDREFS #IMPLIED
collaboration IDREFS #IMPLIED
behavior IDREFS #IMPLIED
partition IDREFS #IMPLIED
request IDREFS #IMPLIED
message IDREFS #IMPLIED
actionSequence IDREFS #IMPLIED
%XMI.element.att;
%XMI.link.att;
>

<!-- _______________________________________________________________ -->
<!-- -->
<!-- CLASS: Behavioral_Elements.Common_Behavior.SendAction -->
<!-- -->

<!ATTLIST UML:SendAction
  name CDATA #IMPLIED
  visibility (public | protected | private) #IMPLIED
  isAsynchronous (false | true) #IMPLIED
  script CDATA #IMPLIED
  binding IDREFS #IMPLIED
  template IDREFS #IMPLIED
  templateParameter IDREFS #IMPLIED
  implementation IDREFS #IMPLIED
  view IDREFS #IMPLIED
  presentation IDREFS #IMPLIED
  namespace IDREFS #IMPLIED
  constraint IDREFS #IMPLIED
  requirement IDREFS #IMPLIED
  provision IDREFS #IMPLIED
  stereotype IDREFS #IMPLIED
  elementReference IDREFS #IMPLIED
  collaboration IDREFS #IMPLIED
  behavior IDREFS #IMPLIED
  partition IDREFS #IMPLIED
  request IDREFS #IMPLIED
  message IDREFS #IMPLIED
  actionSequence IDREFS #IMPLIED
  %XMI.element.att;
  %XMI.link.att;>
<!-- _______________________________________________________________ -->
<!ELEMENT UML:UninterpretedAction.body (#PCDATA | XMI.reference)* >
<!ATTLIST UML:UninterpretedAction name CDATA #IMPLIED
visibility (public | protected | private) #IMPLIED
isAsynchronous (false | true) #IMPLIED
script CDATA #IMPLIED
body CDATA #IMPLIED
binding IDREFS #IMPLIED
template IDREFS #IMPLIED
templateParameter IDREFS #IMPLIED
implementation IDREFS #IMPLIED
view IDREFS #IMPLIED
presentation IDREFS #IMPLIED
namespace IDREFS #IMPLIED
constraint IDREFS #IMPLIED
requirement IDREFS #IMPLIED
provision IDREFS #IMPLIED
stereotype IDREFS #IMPLIED
elementReference IDREFS #IMPLIED
collaboration IDREFS #IMPLIED
behavior IDREFS #IMPLIED
partition IDREFS #IMPLIED
request IDREFS #IMPLIED
message IDREFS #IMPLIED
actionSequence IDREFS #IMPLIED
%XMI.element.att;
%XMI.link.att;
>
<!-- _______________________________________________________________ -->
<!-- -->
<!-- CLASS: Behavioral_Elements.Common_Behavior.TerminateAction -->
<!-- _______________________________________________________________ -->

<!ELEMENT UML:TerminateAction (UML:ModelElement.name |
  UML:Action.isAsynchronous | UML:Action.script | XMI.extension |
  UML:ModelElement.binding | UML:ModelElement.template |
  UML:ModelElement.templateParameter | UML:ModelElement.implementation |
  UML:ModelElement.view |
  UML:ModelElement.presentation |
  UML:ModelElement.namespace |
  UML:ModelElement.constraint |
  UML:ModelElement.requirement |
  UML:ModelElement.provision |
  UML:ModelElement.stereotype |
  UML:ModelElement.elementReference |
  UML:ModelElement.collaboration |
  UML:ModelElement.behavior |
  UML:ModelElement.partition |
  UML:Action.request | UML:Action.message |
  UML:Action.actionSequence |
  UML:ModelElement.taggedValue |
  UML:Action.actualArgument)* >

<!ATTLIST UML:TerminateAction
  name CDATA #IMPLIED
  visibility (public | protected | private) #IMPLIED
  isAsynchronous (false | true) #IMPLIED
  script CDATA #IMPLIED
  binding IDREFS #IMPLIED
  template IDREFS #IMPLIED
  templateParameter IDREFS #IMPLIED
  implementation IDREFS #IMPLIED
  view IDREFS #IMPLIED
  presentation IDREFS #IMPLIED
  namespace IDREFS #IMPLIED
  constraint IDREFS #IMPLIED
  requirement IDREFS #IMPLIED
  provision IDREFS #IMPLIED
  stereotype IDREFS #IMPLIED
  elementReference IDREFS #IMPLIED
  collaboration IDREFS #IMPLIED
  behavior IDREFS #IMPLIED
  partition IDREFS #IMPLIED
  request IDREFS #IMPLIED
<!ELEMENT UML:DestroyAction (UML:ModelElement.name |
  UML:Action.isAsynchronous | UML:Action.script | XML.extension |
  UML:ModelElement.binding | UML:ModelElement.template |
  UML:ModelElement.templateParameter | UML:ModelElement.implementation |
  UML:ModelElement.view | UML:ModelElement.presentation | UML:ModelElement.namespace |
  UML:ModelElement.constraint | UML:ModelElement.requirement |
  UML:ModelElement.provision | UML:ModelElement.stereotype |
  UML:ModelElement.elementReference | UML:ModelElement.collaboration |
  UML:ModelElement.behavior | UML:ModelElement.partition |
  UML:Action.request | UML:Action.message |
  UML:Action.actionSequence |
  UML:ModelElement.taggedValue |
  UML:Action.actualArgument)*>

<!ATTLIST UML:DestroyAction
  name CDATA #IMPLIED
  visibility (public | protected | private) #IMPLIED
  isAsynchronous (false | true) #IMPLIED
  script CDATA #IMPLIED
  binding IDREFS #IMPLIED
  template IDREFS #IMPLIED
  templateParameter IDREFS #IMPLIED
  implementation IDREFS #IMPLIED
  view IDREFS #IMPLIED
  presentation IDREFS #IMPLIED
  namespace IDREFS #IMPLIED
  constraint IDREFS #IMPLIED
  requirement IDREFS #IMPLIED
  provision IDREFS #IMPLIED
  stereotype IDREFS #IMPLIED
  elementReference IDREFS #IMPLIED
  collaboration IDREFS #IMPLIED
  behavior IDREFS #IMPLIED
  partition IDREFS #IMPLIED
<!ELEMENT UML:Link.association (UML:Association)*>
<!ATTLIST UML:Link
name CDATA #IMPLIED
visibility (public | protected | private) #IMPLIED
binding IDREFS #IMPLIED
template IDREFS #IMPLIED
templateParameter IDREFS #IMPLIED
implementation IDREFS #IMPLIED:view IDREFS #IMPLIED
presentation IDREFS #IMPLIED
namespace IDREFS #IMPLIED
constraint IDREFS #IMPLIED
requirement IDREFS #IMPLIED
provision IDREFS #IMPLIED
stereotype IDREFS #IMPLIED
elementReference IDREFS #IMPLIED
collaboration IDREFS #IMPLIED
behavior IDREFS #IMPLIED
partition IDREFS #IMPLIED
association IDREFS #IMPLIED>%

<!-- _______________________________________________________________ -->
<!-- -->
<!-- CLASS: Behavioral_Elements.Common_Behavior.Link -->
<!-- _______________________________________________________________ -->

<!ELEMENT UML:Link.association (UML:Association)*>
<!ATTLIST UML:Link
name CDATA #IMPLIED
visibility (public | protected | private) #IMPLIED
binding IDREFS #IMPLIED
template IDREFS #IMPLIED
templateParameter IDREFS #IMPLIED
implementation IDREFS #IMPLIED:view IDREFS #IMPLIED
presentation IDREFS #IMPLIED
namespace IDREFS #IMPLIED
constraint IDREFS #IMPLIED
requirement IDREFS #IMPLIED
provision IDREFS #IMPLIED
stereotype IDREFS #IMPLIED
elementReference IDREFS #IMPLIED
collaboration IDREFS #IMPLIED
behavior IDREFS #IMPLIED
partition IDREFS #IMPLIED
association IDREFS #IMPLIED>%
<!--
<!-- CLASS: Behavioral_Elements.Common_Behavior.LinkEnd
<!-- _______________________________________________________________ -->

<!ELEMENT UML:LinkEnd.link (UML:Link)* >

<!ELEMENT UML:LinkEnd.associationEnd (UML:AssociationEnd)* >

<!ELEMENT UML:LinkEnd.instance (UML:Instance)* >

<!ELEMENT UML:LinkEnd (UML:ModelElement.name |
    UML:ModelElement.visibility | XMI.extension |
    UML:ModelElement.binding | |
    UML:ModelElement.template | |
    UML:ModelElement.templateParameter | |
    UML:ModelElement.implementation | |
    UML:ModelElement.view | |
    UML:ModelElement.presentation | |
    UML:ModelElement.namespace | |
    UML:ModelElement.constraint | |
    UML:ModelElement.requirement | |
    UML:ModelElement.provision | |
    UML:ModelElement.stereotype | |
    UML:ModelElement.elementReference | |
    UML:ModelElement.collaboration | |
    UML:ModelElement.behavior | |
    UML:ModelElement.partition | UML:LinkEnd.link |
    UML:LinkEnd.associationEnd | UML:LinkEnd.instance |
    UML:ModelElement.taggedValue)* >

<!ATTLIST UML:LinkEnd
    name CDATA #IMPLIED
    visibility (public | protected | private) #IMPLIED
    binding IDREFS #IMPLIED
    template IDREFS #IMPLIED
    templateParameter IDREFS #IMPLIED
    implementation IDREFS #IMPLIED
    view IDREFS #IMPLIED
    presentation IDREFS #IMPLIED
    namespace IDREFS #IMPLIED
    constraint IDREFS #IMPLIED
    requirement IDREFS #IMPLIED
    provision IDREFS #IMPLIED
    stereotype IDREFS #IMPLIED
    elementReference IDREFS #IMPLIED
    collaboration IDREFS #IMPLIED
    behavior IDREFS #IMPLIED
    partition IDREFS #IMPLIED
    link IDREFS #IMPLIED
    associationEnd IDREFS #IMPLIED
    instance IDREFS #IMPLIED
%XMI.element.att;
%XMI.link.att;
>
<!-- CLASS: Behavioral_Elements.Common_Behavior.Instance -->
<!ELEMENT UML:Instance.linkEnd (UML:LinkEnd)* >
<!ELEMENT UML:Instance.messageInstance3 (UML:MessageInstance)* >
<!ELEMENT UML:Instance.messageInstance2 (UML:MessageInstance)* >
<!ELEMENT UML:Instance.messageInstance (UML:MessageInstance)* >
<!ELEMENT UML:Instance.attributeLink (UML:AttributeLink)* >
<!ELEMENT UML:Instance.classifier (UML:Classifier)* >
<!ELEMENT UML:Instance (UML:ModelElement.name |
  UML:ModelElement.visibility | XML.extension |
  UML:ModelElement.binding |
  UML:ModelElement.template |
  UML:ModelElement.templateParameter |
  UML:ModelElement.implementation |
  UML:ModelElement.view |
  UML:ModelElement.presentation |
  UML:ModelElement.namespace |
  UML:ModelElement.constraint |
  UML:ModelElement.requirement |
  UML:ModelElement.provision |
  UML:ModelElement.stereotype |
  UML:ModelElement.elementReference |
  UML:ModelElement.behavior |
  UML:ModelElement.partition |
  UML:Instance.linkEnd |
  UML:Instance.messageInstance3 |
  UML:Instance.messageInstance2 |
  UML:Instance.messageInstance |
  UML:Instance.attributeLink |
  UML:Instance.classifier |
  UML:ModelElement.taggedValue |
  UML:Instance.slot)* >
<!ATTLIST UML:Instance
  name CDATA #IMPLIED
  visibility (public | protected | private) #IMPLIED
  binding IDREFS #IMPLIED
  template IDREFS #IMPLIED
  templateParameter IDREFS #IMPLIED
  implementation IDREFS #IMPLIED
  view IDREFS #IMPLIED
  presentation IDREFS #IMPLIED
  namespace IDREFS #IMPLIED
  constraint IDREFS #IMPLIED
  requirement IDREFS #IMPLIED
  provision IDREFS #IMPLIED
  stereotype IDREFS #IMPLIED
elementReference IDREFS #IMPLIED
collaboration IDREFS #IMPLIED
behavior IDREFS #IMPLIED
partition IDREFS #IMPLIED
linkEnd IDREFS #IMPLIED
messageInstance3 IDREFS #IMPLIED
messageInstance2 IDREFS #IMPLIED
messageInstance IDREFS #IMPLIED
attributeLink IDREFS #IMPLIED
classifier IDREFS #IMPLIED
%XMI.element.att;
%XMI.link.att;
>
<!-- _______________________________________________________________ -->
<!-- -->
<!-- CLASS: Behavioral_Elements.Common_Behavior.AttributeLink -->
<!-- _______________________________________________________________ -->

<!ELEMENT UML:AttributeLink.instance (UML:Instance)* >

<!ELEMENT UML:AttributeLink.attribute (UML:Attribute)* >

<!ELEMENT UML:AttributeLink.value (UML:Instance)* >

<!ELEMENT UML:AttributeLink (UML:ModelElement.name |
UML:ModelElement.visibility | XMI.extension |
UML:ModelElement.binding |
UML:ModelElement.template |
UML:ModelElement.templateParameter |
UML:ModelElement.implementation |
UML:ModelElement.view |
UML:ModelElement.presentation |
UML:ModelElement.namespace |
UML:ModelElement.constraint |
UML:ModelElement.requirement |
UML:ModelElement.provision |
UML:ModelElement.stereotype |
UML:ModelElement.elementReference |
UML:ModelElement.collaboration |
UML:ModelElement.behavior |
UML:ModelElement.partition |
UML:AttributeLink.instance |
UML:AttributeLink.attribute |
UML:AttributeLink.value |
UML:ModelElement.taggedValue)* >

<!ATTLIST UML:AttributeLink
name CDATA #IMPLIED
visibility (public | protected | private) #IMPLIED
binding IDREFS #IMPLIED
template IDREFS #IMPLIED
templateParameter IDREFS #IMPLIED
implementation IDREFS #IMPLIED
view IDREFS #IMPLIED
presentation IDREFS #IMPLIED
namespace IDREFS #IMPLIED
constraint IDREFS #IMPLIED
requirement IDREFS #IMPLIED
provision IDREFS #IMPLIED
stereotype IDREFS #IMPLIED
elementReference IDREFS #IMPLIED
collaboration IDREFS #IMPLIED
behavior IDREFS #IMPLIED
instance IDREFS #IMPLIED
attribute IDREFS #IMPLIED
value IDREFS #IMPLIED
%XMI.element.att;
%XMI.link.att;

<!-- _______________________________________________________________ -->
<!-- -->
<!-- CLASS: Behavioral Elements Common Behavior Object -->
<!-- _______________________________________________________________ -->
<!ELEMENT UML: Object (UML: ModelElement.name |
    UML: ModelElement.visibility | XMI.extension |
    UML: ModelElement.binding |
    UML: ModelElement.template |
    UML: ModelElement.templateParameter |
    UML: ModelElement.implementation |
    UML: ModelElement.view |
    UML: ModelElement.presentation |
    UML: ModelElement.namespace |
    UML: ModelElement.constraint |
    UML: ModelElement.requirement |
    UML: ModelElement.provision |
    UML: ModelElement.stereotype |
    UML: ModelElement.elementReference |
    UML: ModelElement.collaboration |
    UML: ModelElement.behavior |
    UML: ModelElement.partition | UML: Instance.linkEnd |
    UML: Instance.messageInstance3 |
    UML: Instance.messageInstance2 |
    UML: Instance.messageInstance |
    UML: Instance.attributeLink |
    UML: Instance.classifier |
    UML: ModelElement.taggedValue | UML: Instance.slot)* >
<!ATTLIST UML: Object
    name CDATA #IMPLIED
    visibility (public | protected | private) #IMPLIED
    binding IDREFS #IMPLIED
    template IDREFS #IMPLIED
    templateParameter IDREFS #IMPLIED
    implementation IDREFS #IMPLIED
    view IDREFS #IMPLIED
    presentation IDREFS #IMPLIED
    namespace IDREFS #IMPLIED
    constraint IDREFS #IMPLIED
<!ATTLIST UML:DataValue
  name CDATA #IMPLIED
  visibility (public | protected | private) #IMPLIED
  binding IDREFS #IMPLIED
  template IDREFS #IMPLIED
  templateParameter IDREFS #IMPLIED
  implementation IDREFS #IMPLIED
  view IDREFS #IMPLIED
<!ELEMENT UML:LinkObject (UML:ModelElement.name | 
    UML:ModelElement.visibility | XMI.extension | 
    UML:ModelElement.binding | 
    UML:ModelElement.template | 
    UML:ModelElement.templateParameter | 
    UML:ModelElement.implementation | 
    UML:ModelElement.view | 
    UML:ModelElement.presentation | 
    UML:ModelElement.namespace | 
    UML:ModelElement.constraint | 
    UML:ModelElement.requirement | 
    UML:ModelElement.provision | 
    UML:ModelElement.stereotype | 
    UML:ModelElement.elementReference | 
    UML:ModelElement.taggedValue | 
    UML:ModelElement.classifier | 
    UML:Link.association | 
    UML:Instance.slot | UML:Link.linkRole)* >

<!ATTLIST UML:LinkObject
    name CDATA #IMPLIED
    visibility (public | protected | private) #IMPLIED
    binding IDREFS #IMPLIED
    template IDREFS #IMPLIED
```xml
<templateParameter IDREFS #IMPLIED
implementation IDREFS #IMPLIED
view IDREFS #IMPLIED
presentation IDREFS #IMPLIED
namespace IDREFS #IMPLIED
constraint IDREFS #IMPLIED
requirement IDREFS #IMPLIED
provision IDREFS #IMPLIED
stereotype IDREFS #IMPLIED
elementReference IDREFS #IMPLIED
collaboration IDREFS #IMPLIED
behavior IDREFS #IMPLIED
partition IDREFS #IMPLIED
linkEnd IDREFS #IMPLIED
messageInstance3 IDREFS #IMPLIED
messageInstance2 IDREFS #IMPLIED
attributeLink IDREFS #IMPLIED
classifier IDREFS #IMPLIED
association IDREFS #IMPLIED
%XMI.element.att;
%XMI.link.att;
>
<!-- _______________________________________________________________ -->
<!-- -->
<!-- CLASS: Behavioral_Elements.Common_Behavior.MessageInstance -->
<!-- _______________________________________________________________ -->
<!ELEMENT UML:MessageInstance.specification (UML:Request)* >
<!ELEMENT UML:MessageInstance.sender (UML:Instance)* >
<!ELEMENT UML:MessageInstance.receiver (UML:Instance)* >
<!ELEMENT UML:MessageInstance.argument (UML:Instance)* >
<!ELEMENT UML:MessageInstance (UML:ModelElement.name |
UML:ModelElement.visibility |
XMI.extension | UML:ModelElement.binding |
UML:ModelElement.template |
UML:ModelElement.templateParameter |
UML:ModelElement.implementation |
UML:ModelElement.view |
UML:ModelElement.presentation |
UML:ModelElement.namespace |
UML:ModelElement.constraint |
UML:ModelElement.requirement |
UML:ModelElement.provision |
UML:ModelElement.stereotype |
UML:ModelElement.elementReference |
UML:ModelElement.collaboration |
UML:ModelElement.behavior |
UML:ModelElement.partition |
UML:MessageInstance.specification |
<!ATTLIST UML:MessageInstance
  name CDATA #IMPLIED
  visibility (public | protected | private) #IMPLIED
  binding IDREFS #IMPLIED
  template IDREFS #IMPLIED
  templateParameter IDREFS #IMPLIED
  implementation IDREFS #IMPLIED
  view IDREFS #IMPLIED
  presentation IDREFS #IMPLIED
  namespace IDREFS #IMPLIED
  constraint IDREFS #IMPLIED
  requirement IDREFS #IMPLIED
  provision IDREFS #IMPLIED
  stereotype IDREFS #IMPLIED
  elementReference IDREFS #IMPLIED
  collaboration IDREFS #IMPLIED
  behavior IDREFS #IMPLIED
  partition IDREFS #IMPLIED
  specification IDREFS #IMPLIED
  sender IDREFS #IMPLIED
  receiver IDREFS #IMPLIED
  argument IDREFS #IMPLIED
>%

<![XMI.element.att;]
<![XMI.link.att;]>

<!ELEMENT UML:Collaboration.representedClassifier (UML:Classifier)* >
<!ELEMENT UML:Collaboration.representedOperation (UML:Operation)* >
<!ELEMENT UML:Collaboration.constrainingElement (UML:ModelElement)* >
<!ELEMENT UML:Collaboration (UML:ModelElement.name |
  UML:ModelElement.visibility | XMI.extension |
  UML:ModelElement.bind |
  UML:ModelElement.template |
  UML:ModelElement.templateParameter |
  UML:ModelElement.implementation |
  UML:ModelElement.view |
  UML:ModelElement.presentation |
  UML:ModelElement.namespace |
  UML:ModelElement.constraint |
  UML:ModelElement.requirement |
  UML:ModelElement.provision |
  UML:ModelElement.stereotype |
  UML:ModelElement.elementReference |>
<!ATTLIST UML:Collaboration
    name CDATA #IMPLIED
    visibility (public | protected | private) #IMPLIED
    binding IDREFS #IMPLIED
    template IDREFS #IMPLIED
    templateParameter IDREFS #IMPLIED
    implementation IDREFS #IMPLIED
    view IDREFS #IMPLIED
    presentation IDREFS #IMPLIED
    namespace IDREFS #IMPLIED
    constraint IDREFS #IMPLIED
    requirement IDREFS #IMPLIED
    provision IDREFS #IMPLIED
    stereotype IDREFS #IMPLIED
    elementReference IDREFS #IMPLIED
    collaboration IDREFS #IMPLIED
    behavior IDREFS #IMPLIED
    partition IDREFS #IMPLIED
    representedClassifier IDREFS #IMPLIED
    representedOperation IDREFS #IMPLIED
    constrainingElement IDREFS #IMPLIED
    %XMI.element.att;
    %XMI.link.att;
>
<!-- _______________________________________________________________ -->
<!-- -->
<!-- CLASS: Behavioral_Elements.Collaborations.Interaction -->
<!-- _______________________________________________________________ -->

<!ELEMENT UML:Interaction.context (UML:Collaboration)* >

<!ELEMENT UML:Interaction (UML:ModelElement.name |
UML:ModelElement.visibility | XMI.extension |
UML:ModelElement.binding |
UML:ModelElement.template |
UML:ModelElement.templateParameter |
UML:ModelElement.implementation |
UML:ModelElement.view |
UML:ModelElement.presentation |
UML:ModelElement.namespace |
UML:ModelElement.constraint |
UML:ModelElement.requirement |
UML:ModelElement.provision |
UML:ModelElement.stereotype |
UML:ModelElement.elementReference |
<!ATTLIST UML:Interaction
  name CDATA #IMPLIED
  visibility (public | protected | private) #IMPLIED
  binding IDREFS #IMPLIED
  template IDREFS #IMPLIED
  templateParameter IDREFS #IMPLIED
  implementation IDREFS #IMPLIED
  view IDREFS #IMPLIED
  presentation IDREFS #IMPLIED
  namespace IDREFS #IMPLIED
  constraint IDREFS #IMPLIED
  requirement IDREFS #IMPLIED
  provision IDREFS #IMPLIED
  stereotype IDREFS #IMPLIED
  elementReference IDREFS #IMPLIED
  collaboration IDREFS #IMPLIED
  behavior IDREFS #IMPLIED
  partition IDREFS #IMPLIED
  context IDREFS #IMPLIED
%XMI.element.att;
%XMI.link.att;
>

<!-- _______________________________________________________________ -->
<!-- -->
<!-- CLASS: Behavioral_Elements.Collaborations.AssociationRole -->
<!-- _______________________________________________________________ -->
<!ELEMENT UML:AssociationRole.multiplicity (#PCDATA | XMI.reference)* >
<!ELEMENT UML:AssociationRole.base (UML:Association)* >
<!ELEMENT UML:AssociationRole.namespace (UML:Collaboration)* >
<!ELEMENT UML:AssociationRole (UML:ModelElement.name |
  UML:ModelElement.visibility |
  UML:GeneralizableElement.isRoot |
  UML:GeneralizableElement.isLeaf |
  UML:GeneralizableElement.isAbstract |
  UML:AssociationRole.multiplicity |
  XMI.extension | UML:ModelElement.binding |
  UML:ModelElement.template |
  UML:ModelElement.templateParameter |
  UML:ModelElement.implementation |
  UML:ModelElement.view |
  UML:ModelElement.presentation |
  UML:ModelElement.namespace |
  UML:ModelElement.constraint |
  UML:ModelElement.requirement |
<!ATTLIST UML:AssociationRole
  name CDATA #IMPLIED
  visibility (public | protected | private) #IMPLIED
  isRoot (false | true) #IMPLIED
  isLeaf (false | true) #IMPLIED
  isAbstract (false | true) #IMPLIED
  multiplicity CDATA #IMPLIED
  binding IDREFS #IMPLIED
  template IDREFS #IMPLIED
  templateParameter IDREFS #IMPLIED
  implementation IDREFS #IMPLIED
  view IDREFS #IMPLIED
  presentation IDREFS #IMPLIED
  ModelElement.namespace IDREFS #IMPLIED
  constraint IDREFS #IMPLIED
  requirement IDREFS #IMPLIED
  provision IDREFS #IMPLIED
  stereotype IDREFS #IMPLIED
  elementReference IDREFS #IMPLIED
  collaboration IDREFS #IMPLIED
  behavior IDREFS #IMPLIED
  partition IDREFS #IMPLIED
  generalization IDREFS #IMPLIED
  specialization IDREFS #IMPLIED
  link IDREFS #IMPLIED
  associationEnd IDREFS #IMPLIED
  base IDREFS #IMPLIED
  AssociationRole.namespace IDREFS #IMPLIED
  %XMI.element.att;
  %XMI.link.att;
>
</!

<!-- _______________________________ -->

<!-- CLASS: Behavioral_Elements.Collaborations.AssociationEndRole -->

<!-- _______________________________ -->

<!ELEMENT UML:AssociationEndRole.associationRole
  (UML:AssociationRole)* >
<!ELEMENT UML:AssociationEndRole.base (UML:AssociationEnd)* >


<!ATTLIST UML:AssociationEndRole
    name CDATA #IMPLIED
    visibility (public | protected | private) #IMPLIED
    isNavigable (false | true) #IMPLIED
    isOrdered (false | true) #IMPLIED
    aggregation (none | shared | composite) #IMPLIED
    multiplicity CDATA #IMPLIED
    changeable (none | frozen | addOnly) #IMPLIED
    targetScope (classifier | instance) #IMPLIED
    binding IDREFS #IMPLIED
    template IDREFS #IMPLIED
    templateParameter IDREFS #IMPLIED
    implementation IDREFS #IMPLIED
    view IDREFS #IMPLIED
    presentation IDREFS #IMPLIED
    namespace IDREFS #IMPLIED
    constraint IDREFS #IMPLIED
    requirement IDREFS #IMPLIED
    provision IDREFS #IMPLIED
    stereotype IDREFS #IMPLIED
elementReference IDREFS #IMPLIED
collaboration IDREFS #IMPLIED
behavior IDREFS #IMPLIED
partition IDREFS #IMPLIED
type IDREFS #IMPLIED
specification IDREFS #IMPLIED
association IDREFS #IMPLIED
linkEnd IDREFS #IMPLIED
associationEndRole IDREFS #IMPLIED
associationRole IDREFS #IMPLIED
base IDREFS #IMPLIED
%XMI.element.att;
%XMI.link.att;

<!-- _______________________________________________________________ -->
<!-- -->
<!-- CLASS: Behavioral_Elements.Collaborations.Message -->
<!-- _______________________________________________________________ -->

<!ELEMENT UML:Message.interaction (UML:Interaction)* >
<!ELEMENT UML:Message.predecessor (UML:Message)* >
<!ELEMENT UML:Message.message2 (UML:Message)* >
<!ELEMENT UML:Message.message (UML:Message)* >
<!ELEMENT UML:Message.activator (UML:Message)* >
<!ELEMENT UML:Message.action (UML:Action)* >
<!ELEMENT UML:Message.receiver (UML:ClassifierRole)* >
<!ELEMENT UML:Message.sender (UML:ClassifierRole)* >
<!ELEMENT UML:Message (UML:ModelElement.name |
UML:ModelElement.visibility | XMI.extension |
UML:ModelElement.binding |
UML:ModelElement.template |
UML:ModelElement.templateParameter |
UML:ModelElement.implementation |
UML:ModelElement.view |
UML:ModelElement.presentation |
UML:ModelElement.namespace |
UML:ModelElement.constraint |
UML:ModelElement.requirement |
UML:ModelElement.provision |
UML:ModelElement.stereotype |
UML:ModelElement.elementReference |
UML:ModelElement.collaboration |
UML:ModelElement.behavior |
UML:ModelElement.partition |
UML:Message.interaction | UML:Message.predecessor |
UML:Message.message2 | UML:Message.message |
<!ATTLIST UML:Message
  name CDATA #IMPLIED
  visibility (public | protected | private) #IMPLIED
  binding IDREFS #IMPLIED
  template IDREFS #IMPLIED
  templateParameter IDREFS #IMPLIED
  implementation IDREFS #IMPLIED
  view IDREFS #IMPLIED
  presentation IDREFS #IMPLIED
  namespace IDREFS #IMPLIED
  constraint IDREFS #IMPLIED
  requirement IDREFS #IMPLIED
  provision IDREFS #IMPLIED
  stereotype IDREFS #IMPLIED
  elementReference IDREFS #IMPLIED
  collaboration IDREFS #IMPLIED
  behavior IDREFS #IMPLIED
  partition IDREFS #IMPLIED
  interaction IDREFS #IMPLIED
  predecessor IDREFS #IMPLIED
  message2 IDREFS #IMPLIED
  message IDREFS #IMPLIED
  activator IDREFS #IMPLIED
  action IDREFS #IMPLIED
  receiver IDREFS #IMPLIED
  sender IDREFS #IMPLIED
  %XMI.element.att;
  %XMI.link.att;
>

<!ELEMENT UML:ClassifierRole.multiplicity (#PCDATA | XMI.reference)* >
<!ELEMENT UML:ClassifierRole.associationEndRole (UML:AssociationEndRole)* >
<!ELEMENT UML:ClassifierRole.namespace (UML:Collaboration)* >
<!ELEMENT UML:ClassifierRole.message2 (UML:Message)* >
<!ELEMENT UML:ClassifierRole.message (UML:Message)* >
<!ELEMENT UML:ClassifierRole.base (UML:Classifier)* >
<!ELEMENT UML:ClassifierRole.availableFeature (UML:Feature)* >
<!ELEMENT UML:ClassifierRole (UML:ModelElement.name | UML:ModelElement.visibility |
<!DOCTYPE DTD [ 
<!ATTLIST UML:ClassifierRole 
  name CDATA #IMPLIED 
  visibility (public | protected | private) #IMPLIED 
  isRoot (false | true) #IMPLIED 
  isLeaf (false | true) #IMPLIED 
  isAbstract (false | true) #IMPLIED 
  multiplicity CDATA #IMPLIED 
  binding IDREFS #IMPLIED 
  template IDREFS #IMPLIED 
  templateParameter IDREFS #IMPLIED 
  implementation IDREFS #IMPLIED 
  view IDREFS #IMPLIED 
  presentation IDREFS #IMPLIED 
  ModelElement.namespace IDREFS #IMPLIED]>

UML:GeneralizableElement.isRoot |
UML:GeneralizableElement.isLeaf |
UML:GeneralizableElement.isAbstract |
UML:ClassifierRole.multiplicity |
XML.extension | UML:ModelElement.binding |
UML:ModelElement.template |
UML:ModelElement.templateParameter |
UML:ModelElement.implementation |
UML:ModelElement.view |
UML:ModelElement.presentation |
UML:ModelElement.namespace |
UML:ModelElement.constraint |
UML:ModelElement.requirement |
UML:ModelElement.provision |
UML:ModelElement.stereotype |
UML:ModelElement.elementReference |
UML:ModelElement.collaboration |
UML:ModelElement.behavior |
UML:ModelElement.partition |
UML:GeneralizableElement.generalization |
UML:GeneralizableElement.specialization |
UML:Classifier.parameter |
UML:Classifier.structuralFeature |
UML:Classifier.specification |
UML:Classifier.realization |
UML:Classifier.associationEnd |
UML:Classifier.participant |
UML:Classifier.createAction |
UML:Classifier.instance |
UML:Classifier.collaboration |
UML:Classifier.classifierRole |
UML:Classifier.classifierInState |
UML:ClassifierRole.associationEndRole |
UML:ClassifierRole.namespace |
UML:ClassifierRole.message2 |
UML:ClassifierRole.message |
UML:ClassifierRole.base |
UML:ClassifierRole.availableFeature |
UML:ModelElement.taggedValue |
UML:Namespace.ownedElement |
UML:Classifier.feature)* >
constraint IDREFS #IMPLIED
requirement IDREFS #IMPLIED
provision IDREFS #IMPLIED
stereotype IDREFS #IMPLIED
elementReference IDREFS #IMPLIED
ModelElement.collaboration IDREFS #IMPLIED
behavior IDREFS #IMPLIED
partition IDREFS #IMPLIED
generalization IDREFS #IMPLIED
specialization IDREFS #IMPLIED
parameter IDREFS #IMPLIED
structuralFeature IDREFS #IMPLIED
specification IDREFS #IMPLIED
realization IDREFS #IMPLIED
associationEnd IDREFS #IMPLIED
participant IDREFS #IMPLIED
createAction IDREFS #IMPLIED
instance IDREFS #IMPLIED
Classifier.collaboration IDREFS #IMPLIED
classifierRole IDREFS #IMPLIED
classifierInState IDREFS #IMPLIED
associationEndRole IDREFS #IMPLIED
ClassifierRole.namespace IDREFS #IMPLIED
message2 IDREFS #IMPLIED
message IDREFS #IMPLIED
base IDREFS #IMPLIED
availableFeature IDREFS #IMPLIED
%XMI.element.att;
%XMI.link.att;
>
<!-- _______________________________________________________________ -->
<!-- -->
<!-- CLASS: Behavioral_Elements.Use_Cases.Actor -->
<!-- _______________________________________________________________ -->
<!ATTLIST UML:Actor
  name CDATA #IMPLIED
  visibility (public | protected | private) #IMPLIED
  isRoot (false | true) #IMPLIED
  isLeaf (false | true) #IMPLIED
  isAbstract (false | true) #IMPLIED
  binding IDREFS #IMPLIED
  template IDREFS #IMPLIED
  templateParameter IDREFS #IMPLIED
  implementation IDREFS #IMPLIED
  view IDREFS #IMPLIED
  presentation IDREFS #IMPLIED
  namespace IDREFS #IMPLIED
  constraint IDREFS #IMPLIED
  requirement IDREFS #IMPLIED
  provision IDREFS #IMPLIED
  stereotype IDREFS #IMPLIED
  elementReference IDREFS #IMPLIED
  ModelElement.collaboration IDREFS #IMPLIED
  behavior IDREFS #IMPLIED
  partition IDREFS #IMPLIED
  generalization IDREFS #IMPLIED
  specialization IDREFS #IMPLIED
  parameter IDREFS #IMPLIED
  structuralFeature IDREFS #IMPLIED
  specification IDREFS #IMPLIED
  realization IDREFS #IMPLIED
  associationEnd IDREFS #IMPLIED
  participant IDREFS #IMPLIED
  createAction IDREFS #IMPLIED
  instance IDREFS #IMPLIED
  Classifier.collaboration IDREFS #IMPLIED
  classifierRole IDREFS #IMPLIED
  classifierInState IDREFS #IMPLIED
  %XMI.element.att;
  %XMI.link.att;>

<!ATTLIST UML:UseCase
 name CDATA #IMPLIED
 visibility (public | protected | private) #IMPLIED
 isRoot (false | true) #IMPLIED
 isLeaf (false | true) #IMPLIED
 isAbstract (false | true) #IMPLIED
 extensionPoint CDATA #IMPLIED
 binding IDREFS #IMPLIED
 template IDREFS #IMPLIED
 templateParameter IDREFS #IMPLIED
 implementation IDREFS #IMPLIED
view IDREFS #IMPLIED
presentation IDREFS #IMPLIED
namespace IDREFS #IMPLIED
constraint IDREFS #IMPLIED
requirement IDREFS #IMPLIED
provision IDREFS #IMPLIED
stereotype IDREFS #IMPLIED
elementReference IDREFS #IMPLIED
ModelElement.collaboration IDREFS #IMPLIED
behavior IDREFS #IMPLIED
partition IDREFS #IMPLIED
generalization IDREFS #IMPLIED
specialization IDREFS #IMPLIED
parameter IDREFS #IMPLIED
structuralFeature IDREFS #IMPLIED
specification IDREFS #IMPLIED
realization IDREFS #IMPLIED
associationEnd IDREFS #IMPLIED
participant IDREFS #IMPLIED
createAction IDREFS #IMPLIED
instance IDREFS #IMPLIED
Classifier.collaboration IDREFS #IMPLIED
classifierRole IDREFS #IMPLIED
classifierInState IDREFS #IMPLIED
%XMI.element.att;
%XMI.link.att;

<!ELEMENT UML:UseCaseInstance (UML:ModelElement.name |
UML:ModelElement.visibility |
XMI.extension | UML:ModelElement.binding |
UML:ModelElement.template |
UML:ModelElement.templateParameter |
UML:ModelElement.implementation |
UML:ModelElement.view |
UML:ModelElement.presentation |
UML:ModelElement.namespace |
UML:ModelElement.constraint |
UML:ModelElement.requirement |
UML:ModelElement.provision |
UML:ModelElement.stereotype |
UML:ModelElement.elementReference |
UML:ModelElement.collaboration |
UML:ModelElement.behavior |
UML:ModelElement.partition |
UML:Instance.linkEnd |
UML:Instance.messageInstance3 |
UML:Instance.messageInstance2 |
UML:Instance.messageInstance |
UML:Instance.attributeLink |
<!ATTLIST UML:UseCaseInstance
    name CDATA #IMPLIED
    visibility (public | protected | private) #IMPLIED
    binding IDREFS #IMPLIED
    template IDREFS #IMPLIED
    templateParameter IDREFS #IMPLIED
    implementation IDREFS #IMPLIED
    view IDREFS #IMPLIED
    presentation IDREFS #IMPLIED
    namespace IDREFS #IMPLIED
    constraint IDREFS #IMPLIED
    requirement IDREFS #IMPLIED
    provision IDREFS #IMPLIED
    stereotype IDREFS #IMPLIED
    elementReference IDREFS #IMPLIED
    collaboration IDREFS #IMPLIED
    behavior IDREFS #IMPLIED
    partition IDREFS #IMPLIED
    linkEnd IDREFS #IMPLIED
    messageInstance3 IDREFS #IMPLIED
    messageInstance2 IDREFS #IMPLIED
    messageInstance IDREFS #IMPLIED
    attributeLink IDREFS #IMPLIED
    classifier IDREFS #IMPLIED
%
%XMI.element.att;
%XMI.link.att;
>
<!-- _______________________________________________________________ -->
<!-- -->
<!-- CLASS: Behavioral_Elements.State_Machines.StateMachine -->
<!-- _______________________________________________________________ -->
<!ELEMENT UML:StateMachine.context (UML:ModelElement)* >
<!ELEMENT UML:StateMachine.submachineState (UML:SubmachineState)* >
<!ELEMENT UML:StateMachine (UML:ModelElement.name |
<!ATTLIST UML:StateMachine
name CDATA #IMPLIED
visibility (public | protected | private) #IMPLIED
binding IDREFS #IMPLIED
template IDREFS #IMPLIED
templateParameter IDREFS #IMPLIED
implementation IDREFS #IMPLIED
view IDREFS #IMPLIED
presentation IDREFS #IMPLIED
namespace IDREFS #IMPLIED
constraint IDREFS #IMPLIED
requirement IDREFS #IMPLIED
provision IDREFS #IMPLIED
stereotype IDREFS #IMPLIED
elementReference IDREFS #IMPLIED
collaboration IDREFS #IMPLIED
behavior IDREFS #IMPLIED
partition IDREFS #IMPLIED
context IDREFS #IMPLIED
submachineState IDREFS #IMPLIED
%XMI.element.att;
%XMI.link.att;
>
<!ELEMENT UML:Guard.expression (UML:BooleanExpression) >
<!ELEMENT UML:Guard.transition (UML:Transition)* >
<!ATTLIST UML:Guard
  name CDATA #IMPLIED
  visibility (public | protected | private) #IMPLIED
  binding IDREFS #IMPLIED
  template IDREFS #IMPLIED
  templateParameter IDREFS #IMPLIED
  implementation IDREFS #IMPLIED
  view IDREFS #IMPLIED
  presentation IDREFS #IMPLIED
  namespace IDREFS #IMPLIED
  constraint IDREFS #IMPLIED
  requirement IDREFS #IMPLIED
  provision IDREFS #IMPLIED
  stereotype IDREFS #IMPLIED
  elementReference IDREFS #IMPLIED
  collaboration IDREFS #IMPLIED
  behavior IDREFS #IMPLIED
  partition IDREFS #IMPLIED
  transition IDREFS #IMPLIED
%XMI.element.att;
%XMI.link.att;
>

<!-- _______________________________________________________________ -->
<!ELEMENT UML:StateVertex.outgoing (UML:Transition)* >
<!ELEMENT UML:StateVertex.incoming (UML:Transition)* >
<!ELEMENT UML:StateVertex.parent (UML:CompositeState)* >
<!ELEMENT UML:StateVertex (UML:ModelElement.name |
  UML:ModelElement.visibility | XMI.extension |
  UML:ModelElement.binding |
  UML:ModelElement.template |
  UML:ModelElement.templateParameter |
  UML:ModelElement.implementation |
  UML:ModelElement.view |
  UML:ModelElement.presentation |
  UML:ModelElement.namespace |
  UML:ModelElement.constraint |
  UML:ModelElement.requirement |
  UML:ModelElement.provision |
  UML:ModelElement.stereotype |
  UML:ModelElement.elementReference |
  UML:ModelElement.collaboration |
  UML:ModelElement.behavior |
  UML:ModelElement.partition |
  UML:StateVertex.outgoing |
<!ATTLIST UML:StateVertex
  name CDATA #IMPLIED
  visibility (public | protected | private) #IMPLIED
  binding IDREFS #IMPLIED
  template IDREFS #IMPLIED
  templateParameter IDREFS #IMPLIED
  implementation IDREFS #IMPLIED
  view IDREFS #IMPLIED
  presentation IDREFS #IMPLIED
  namespace IDREFS #IMPLIED
  constraint IDREFS #IMPLIED
  requirement IDREFS #IMPLIED
  provision IDREFS #IMPLIED
  stereotype IDREFS #IMPLIED
  elementReference IDREFS #IMPLIED
  collaboration IDREFS #IMPLIED
  behavior IDREFS #IMPLIED
  partition IDREFS #IMPLIED
  outgoing IDREFS #IMPLIED
  incoming IDREFS #IMPLIED
  parent IDREFS #IMPLIED
%XMI.element.att;
%XMI.link.att;
>
<!ELEMENT UML:Transition.source (UML:StateVertex)* >
<!ELEMENT UML:Transition.target (UML:StateVertex)* >
<!ELEMENT UML:Transition.statemachine (UML:StateMachine)* >
<!ELEMENT UML:Transition.state (UML:State)* >
<!ELEMENT UML:Transition.trigger (UML:Event)* >
<!ELEMENT UML:Transition (UML:ModelElement.name | 
  UML:ModelElement.visibility | XMI.extension | 
  UML:ModelElement.binding | 
  UML:ModelElement.template | 
  UML:ModelElement.templateParameter | 
  UML:ModelElement.implementation | 
  UML:ModelElement.view | 
  UML:ModelElement.presentation | 
  UML:ModelElement.namespace | 
  UML:ModelElement.constraint | 
  UML:ModelElement.requirement | 
  UML:ModelElement.provision |

<!ATTLIST UML:Transition
  name CDATA #IMPLIED
  visibility (public | protected | private) #IMPLIED
  binding IDREFS #IMPLIED
  template IDREFS #IMPLIED
  templateParameter IDREFS #IMPLIED
  implementation IDREFS #IMPLIED
  view IDREFS #IMPLIED
  presentation IDREFS #IMPLIED
  namespace IDREFS #IMPLIED
  constraint IDREFS #IMPLIED
  requirement IDREFS #IMPLIED
  provision IDREFS #IMPLIED
  stereotype IDREFS #IMPLIED
  elementReference IDREFS #IMPLIED
  collaboration IDREFS #IMPLIED
  behavior IDREFS #IMPLIED
  partition IDREFS #IMPLIED
  source IDREFS #IMPLIED
  target IDREFS #IMPLIED
  statemachine IDREFS #IMPLIED
  state IDREFS #IMPLIED
  trigger IDREFS #IMPLIED
%XMI.element.att;
%XMI.link.att;
>

<!ELEMENT UML:PseudoState.kind EMPTY >
<!ATTLIST UML:PseudoState.kind
  xmi.value (initial | deepHistory | shallowHistory | join | fork | branch | final) #REQUIRED
>
<!ATTLIST UML:PseudoState
    name CDATA #IMPLIED
    visibility (public | protected | private) #IMPLIED
    kind (initial | deepHistory | shallowHistory | join | fork | branch | final) #IMPLIED
    binding IDREFS #IMPLIED
    template IDREFS #IMPLIED
    templateParameter IDREFS #IMPLIED
    implementation IDREFS #IMPLIED
    view IDREFS #IMPLIED
    presentation IDREFS #IMPLIED
    namespace IDREFS #IMPLIED
    constraint IDREFS #IMPLIED
    requirement IDREFS #IMPLIED
    provision IDREFS #IMPLIED
    stereotype IDREFS #IMPLIED
    elementReference IDREFS #IMPLIED
    collaboration IDREFS #IMPLIED
    behavior IDREFS #IMPLIED
    partition IDREFS #IMPLIED
    outgoing IDREFS #IMPLIED
    incoming IDREFS #IMPLIED
    parent IDREFS #IMPLIED
    %XMI.element.att;
    %XMI.link.att;
>
<!ATTLIST UML:State
  name CDATA #IMPLIED
  visibility (public | protected | private) #IMPLIED
  binding IDREFS #IMPLIED
  template IDREFS #IMPLIED
  templateParameter IDREFS #IMPLIED
  implementation IDREFS #IMPLIED
  view IDREFS #IMPLIED
  presentation IDREFS #IMPLIED
  namespace IDREFS #IMPLIED
  constraint IDREFS #IMPLIED
  requirement IDREFS #IMPLIED
  provision IDREFS #IMPLIED
  stereotype IDREFS #IMPLIED
  elementReference IDREFS #IMPLIED
  collaboration IDREFS #IMPLIED
  behavior IDREFS #IMPLIED
  partition IDREFS #IMPLIED
  outgoing IDREFS #IMPLIED
  incoming IDREFS #IMPLIED
  parent IDREFS #IMPLIED
  stateMachine IDREFS #IMPLIED
  deferredEvent IDREFS #IMPLIED
  classifierInState IDREFS #IMPLIED
  %XMI.element.att;
  %XMI.link.att;>

<!-- _______________________________________________________________ -->
<!-- -->
<!-- CLASS: Behavioral_Elements.State_Machines.CompositeState -->
<!-- _______________________________________________________________ -->
<!ELEMENT UML:CompositeState.isConcurrent EMPTY >
<!ATTLIST UML:CompositeState.isConcurrent
   xmi.value (false | true) #REQUIRED
>
<!ELEMENT UML:CompositeState (UML:ModelElement.name |
   UML:ModelElement.visibility |
   UML:CompositeState.isConcurrent |
   XML.extension | UML:ModelElement.binding |
   UML:ModelElement.template |
   UML:ModelElement.templateParameter |
   UML:ModelElement.implementation |
   UML:ModelElement.view |
   UML:ModelElement.presentation |
   UML:ModelElement.namespace |
   UML:ModelElement.constraint |
   UML:ModelElement.provision |
   UML:ModelElement.stereotype |
   UML:ModelElement.elementReference |
   UML:ModelElement.collaboration |
   UML:ModelElement.behavior |
   UML:ModelElement.partition |
   UML:StateVertex.outgoing |
   UML:StateVertex.incoming |
   UML:StateVertex.parent |
   UML:State.stateMachine |
   UML:State.deferredEvent |
   UML:State.classifierInState |
   UML:ModelElement.taggedValue |
   UML:State.internalTransition |
   UML:State.entry | UML:State.exit |
   UML:CompositeState.substate)* >

<!ATTLIST UML:CompositeState
   name CDATA #IMPLIED
   visibility (public | protected | private) #IMPLIED
   isConcurrent (false | true) #IMPLIED
   binding IDREFS #IMPLIED
   template IDREFS #IMPLIED
   templateParameter IDREFS #IMPLIED
   implementation IDREFS #IMPLIED
   view IDREFS #IMPLIED
   presentation IDREFS #IMPLIED
   namespace IDREFS #IMPLIED
   constraint IDREFS #IMPLIED
   requirement IDREFS #IMPLIED
   provision IDREFS #IMPLIED
   stereotype IDREFS #IMPLIED
   elementReference IDREFS #IMPLIED
   collaboration IDREFS #IMPLIED
   behavior IDREFS #IMPLIED
   partition IDREFS #IMPLIED
   outgoing IDREFS #IMPLIED
   incoming IDREFS #IMPLIED
   parent IDREFS #IMPLIED

<!ATTLIST UML:SimpleState
    name CDATA #IMPLIED
    visibility (public | protected | private) #IMPLIED
    binding IDREFS #IMPLIED
    template IDREFS #IMPLIED
    templateParameter IDREFS #IMPLIED
    implementation IDREFS #IMPLIED
    view IDREFS #IMPLIED
    presentation IDREFS #IMPLIED
    namespace IDREFS #IMPLIED
    constraint IDREFS #IMPLIED
    requirement IDREFS #IMPLIED
    provision IDREFS #IMPLIED
    stereotype IDREFS #IMPLIED
    elementReference IDREFS #IMPLIED
    collaboration IDREFS #IMPLIED
    behavior IDREFS #IMPLIED
partition IDREFS #IMPLIED
outgoing IDREFS #IMPLIED
incoming IDREFS #IMPLIED
parent IDREFS #IMPLIED
stateMachine IDREFS #IMPLIED
defferredEvent IDREFS #IMPLIED
classifierInState IDREFS #IMPLIED
%XMI.element.att;
%XMI.link.att;
>
<!-- _______________________________________________________________ -->
<!-- -->
<!-- CLASS: Behavioral_Elements.State_Machines.SubmachineState -->
<!-- _______________________________________________________________ -->
<!ELEMENT UML:SubmachineState.stateMachine (UML:StateMachine)* >
<!ELEMENT UML:SubmachineState (UML:ModelElement.name |
UML:ModelElement.visibility |
XML.extension | UML:ModelElement.binding |
UML:ModelElement.template |
UML:ModelElement.templateParameter |
UML:ModelElement.implementation |
UML:ModelElement.view |
UML:ModelElement.presentation |
UML:ModelElement.constraint |
UML:ModelElement.requirement |
UML:ModelElement.provision |
UML:ModelElement.stereotype |
UML:ModelElement.elementReference |
UML:ModelElement.collaboration |
UML:ModelElement.behavior |
UML:ModelElement.partition |
UML:StateVertex.outgoing |
UML:StateVertex.incoming |
UML:StateVertex.parent |
UML:State.stateMachine |
UML:State.deferredEvent |
UML:State.classifierInState |
UML:SubmachineState.stateMachine |
UML:ModelElement.taggedValue |
UML:State.internalTransition |
UML:State.entry | UML:State.exit)* >
<!ATTLIST UML:SubmachineState
name CDATA #IMPLIED
visibility (public | protected | private) #IMPLIED
binding IDREFS #IMPLIED
template IDREFS #IMPLIED
templateParameter IDREFS #IMPLIED
implementation IDREFS #IMPLIED
view IDREFS #IMPLIED
presentation IDREFS #IMPLIED
namespace IDREFS #IMPLIED
<!ELEMENT UML:Event.state (UML:State)* >
<!ELEMENT UML:Event.transition (UML:Transition)* >
<!ELEMENT UML:Event (UML:ModelElement.name |
  UML:ModelElement.visibility | XMI.extension |
  UML:ModelElement.binding |
  UML:ModelElement.template |
  UML:ModelElement.templateParameter |
  UML:ModelElement.implementation |
  UML:ModelElement.view |
  UML:ModelElement.presentation |
  UML:ModelElement.namespace |
  UML:ModelElement.constraint |
  UML:ModelElement.requirement |
  UML:ModelElement.provision |
  UML:ModelElement.stereotype |
  UML:ModelElement.elementReference |
  UML:ModelElement.partition | UML:Event.state |
  UML:Event.transition |
  UML:Event.taggedValue)* >
<!ATTLIST UML:Event
  name CDATA #IMPLIED
  visibility (public | protected | private) #IMPLIED
  binding IDREFS #IMPLIED
  template IDREFS #IMPLIED
  templateParameter IDREFS #IMPLIED
  implementation IDREFS #IMPLIED
  view IDREFS #IMPLIED
  constraint IDREFS #IMPLIED
  requirement IDREFS #IMPLIED
  provision IDREFS #IMPLIED
  stereotype IDREFS #IMPLIED
  elementReference IDREFS #IMPLIED
  collaboration IDREFS #IMPLIED
  behavior IDREFS #IMPLIED
  partition IDREFS #IMPLIED
  outgoing IDREFS #IMPLIED
  incoming IDREFS #IMPLIED
  parent IDREFS #IMPLIED
  State.stateMachine IDREFS #IMPLIED
  deferredEvent IDREFS #IMPLIED
  classifierInState IDREFS #IMPLIED
  SubmachineState.stateMachine IDREFS #IMPLIED
  %XMI.element.att;
  %XMI.link.att;
>
<!DOCTYPE UML:ModelElement [
  <!ATTLIST UML:ModelElement
    name CDATA #IMPLIED
    visibility (public | protected | private) #IMPLIED
    binding IDREFS #IMPLIED
    template IDREFS #IMPLIED
    templateParameter IDREFS #IMPLIED
    implementation IDREFS #IMPLIED
    view IDREFS #IMPLIED
    presentation IDREFS #IMPLIED
    namespace IDREFS #IMPLIED
    constraint IDREFS #IMPLIED
    requirement IDREFS #IMPLIED
    provision IDREFS #IMPLIED
]>

<!ENTITY %XMI.element.att; "presentation IDREFS #IMPLIED
namespace IDREFS #IMPLIED
constraint IDREFS #IMPLIED
requirement IDREFS #IMPLIED
provision IDREFS #IMPLIED
stereotype IDREFS #IMPLIED
elementReference IDREFS #IMPLIED
collaboration IDREFS #IMPLIED
behavior IDREFS #IMPLIED
partition IDREFS #IMPLIED
state IDREFS #IMPLIED
transition IDREFS #IMPLIED
%XMI.element.att;
%XMI.link.att;" >

<!-- _______________________________________________________________ -->
<!-- -->
<!-- CLASS: Behavioral_Elements.State_Machines.SignalEvent -->
<!-- _______________________________________________________________ -->

<!ELEMENT UML:SignalEvent.signal (UML:Signal)* >
<!ELEMENT UML:SignalEvent (UML:ModelElement.name |
  UML:ModelElement.visibility | XMI.extension |
  UML:ModelElement.bindin| UML:ModelElement.template |
  UML:ModelElement.templateParameter |
  UML:ModelElement.implementation |
  UML:ModelElement.view |
  UML:ModelElement.presentation |
  UML:ModelElement.namespace |
  UML:ModelElement.constraint |
  UML:ModelElement.requirement |
  UML:ModelElement.provision |
  UML:ModelElement.stereotype |
  UML:ModelElement.elementReference |
  UML:ModelElement.behavior |
  UML:ModelElement.partition | UML:Event.state |
  UML:Event.transition | UML:SignalEvent.signal |
  UML:ModelElement.taggedValue)* >
<!ATTLIST UML:SignalEvent
  name CDATA #IMPLIED
  visibility (public | protected | private) #IMPLIED
  binding IDREFS #IMPLIED
  template IDREFS #IMPLIED
  templateParameter IDREFS #IMPLIED
  implementation IDREFS #IMPLIED
  view IDREFS #IMPLIED
  presentation IDREFS #IMPLIED
  namespace IDREFS #IMPLIED
  constraint IDREFS #IMPLIED
  requirement IDREFS #IMPLIED
  provision IDREFS #IMPLIED
>
stereotype IDREFS #IMPLIED
elementReference IDREFS #IMPLIED
collaboration IDREFS #IMPLIED
behavior IDREFS #IMPLIED
partition IDREFS #IMPLIED
state IDREFS #IMPLIED
transition IDREFS #IMPLIED
signal IDREFS #IMPLIED
%XMI.element.att;
%XMI.link.att;

<!-- _______________________________________________________________ -->
<!-- -->
<!-- CLASS: Behavioral_Elements.State_Machines.CallEvent -->
<!-- _______________________________________________________________ -->

<!ELEMENT UML:CallEvent.operation (UML:Operation)* >
<!ELEMENT UML:CallEvent (UML:ModelElement.name |

UML:ModelElement.visibility | XMI.extension |
UML:ModelElement.binding |
UML:ModelElement.template |
UML:ModelElement.templateParameter |
UML:ModelElement.implementation |
UML:ModelElement.view |
UML:ModelElement.presentation |
UML:ModelElement.namespace |
UML:ModelElement.constraint |
UML:ModelElement.requirement |
UML:ModelElement.provision |
UML:ModelElement.stereotype |
UML:ModelElement.elementReference |
UML:ModelElement.collaboration |
UML:ModelElement.behavior |
UML:ModelElement.partition | UML:Event.state |
UML:Event.transition | UML:CallEvent.operation |
UML:ModelElement.taggedValue)* >
<!ATTLIST UML:CallEvent

name CDATA #IMPLIED
visibility (public | protected | private) #IMPLIED
binding IDREFS #IMPLIED
template IDREFS #IMPLIED
templateParameter IDREFS #IMPLIED
implementation IDREFS #IMPLIED
view IDREFS #IMPLIED
presentation IDREFS #IMPLIED
namespace IDREFS #IMPLIED
constraint IDREFS #IMPLIED
requirement IDREFS #IMPLIED
provision IDREFS #IMPLIED
stereotype IDREFS #IMPLIED
elementReference IDREFS #IMPLIED
collaboration IDREFS #IMPLIED
behavior IDREFS #IMPLIED
<!ELEMENT UML:TimeEvent.duration (UML:TimeExpression) >

<!ELEMENT UML:TimeEvent (UML:ModelElement.name |
  UML:ModelElement.visibility |
  UML:ModelElement.duration | XMI.extension |
  UML:ModelElement.binding |
  UML:ModelElement.template |
  UML:ModelElement.templateParameter |
  UML:ModelElement.implementation |
  UML:ModelElement.view |
  UML:ModelElement.presentation |
  UML:ModelElement.namespace |
  UML:ModelElement.constraint |
  UML:ModelElement.requirement |
  UML:ModelElement.provision |
  UML:ModelElement.stereotype |
  UML:ModelElement.elementReference |
  UML:ModelElement.collaboration |
  UML:ModelElement.behavior |
  UML:ModelElement.partition | UML:Event.state |
  UML:Event.transition |
  UML:ModelElement.taggedValue)* >

<!ATTLIST UML:TimeEvent
  name CDATA #IMPLIED
  visibility (public | protected | private) #IMPLIED
  binding IDREFS #IMPLIED
  template IDREFS #IMPLIED
  templateParameter IDREFS #IMPLIED
  implementation IDREFS #IMPLIED
  view IDREFS #IMPLIED
  presentation IDREFS #IMPLIED
  namespace IDREFS #IMPLIED
  constraint IDREFS #IMPLIED
  requirement IDREFS #IMPLIED
  provision IDREFS #IMPLIED
  stereotype IDREFS #IMPLIED
  elementReference IDREFS #IMPLIED
  collaboration IDREFS #IMPLIED
  behavior IDREFS #IMPLIED
  partition IDREFS #IMPLIED
  state IDREFS #IMPLIED
  transition IDREFS #IMPLIED
<!ELEMENT UML:ChangeEvent.changeExpression (UML:BooleanExpression) >
<!ELEMENT UML:ChangeEvent (UML:ModelElement.name |
    UML:ChangeEvent.changeExpression |
    XMI.extension | UML:ModelElement.binding |
    UML:ModelElement.template |
    UML:ModelElement.templateParameter |
    UML:ModelElement.implementation |
    UML:ModelElement.view |
    UML:ModelElement.presentation |
    UML:ModelElement.namespace |
    UML:ModelElement.constraint |
    UML:ModelElement.requirement |
    UML:ModelElement.provision |
    UML:ModelElement.stereotype |
    UML:ModelElement.elementReference |
    UML:ModelElement.collaboration |
    UML:ModelElement.behavior |
    UML:Event.state |
    UML:Event.transition |
    UML:ModelElement.taggedValue)* >
<!ATTLIST UML:ChangeEvent
    name CDATA #IMPLIED
    visibility (public | protected | private) #IMPLIED
    binding IDREFS #IMPLIED
    template IDREFS #IMPLIED
    templateParameter IDREFS #IMPLIED
    implementation IDREFS #IMPLIED
    view IDREFS #IMPLIED
    presentation IDREFS #IMPLIED
    namespace IDREFS #IMPLIED
    constraint IDREFS #IMPLIED
    requirement IDREFS #IMPLIED
    provision IDREFS #IMPLIED
    stereotype IDREFS #IMPLIED
    elementReference IDREFS #IMPLIED
    collaboration IDREFS #IMPLIED
    behavior IDREFS #IMPLIED
    partition IDREFS #IMPLIED
    state IDREFS #IMPLIED
    transition IDREFS #IMPLIED
%XMI.element.att;
%XMI.link.att;
<!ELEMENT UML:ActivityModel (UML:ModelElement.name |
  UML:ModelElement.visibility | XMI.extension |
  UML:ModelElement.binding |
  UML:ModelElement.template |
  UML:ModelElement.templateParameter |
  UML:ModelElement.implementation |
  UML:ModelElement.view |
  UML:ModelElement.presentation |
  UML:ModelElement.namespace |
  UML:ModelElement.constraint |
  UML:ModelElement.requirement |
  UML:ModelElement.provision |
  UML:ModelElement.stereotype |
  UML:ModelElement.elementReference |
  UML:ModelElement.behavior |
  UML:ModelElement.partition |
  UML:StateMachine.context |
  UML:StateMachine.submachineState |
  UML:ModelElement.taggedValue |
  UML:StateMachine.top |
  UML:StateMachine.transitions |
  UML:ActivityModel.partition)* >

<!ATTLIST UML:ActivityModel
  name CDATA #IMPLIED
  visibility (public | protected | private) #IMPLIED
  binding IDREFS #IMPLIED
  template IDREFS #IMPLIED
  templateParameter IDREFS #IMPLIED
  implementation IDREFS #IMPLIED
  view IDREFS #IMPLIED
  presentation IDREFS #IMPLIED
  namespace IDREFS #IMPLIED
  constraint IDREFS #IMPLIED
  requirement IDREFS #IMPLIED
  provision IDREFS #IMPLIED
  stereotype IDREFS #IMPLIED
  elementReference IDREFS #IMPLIED
  collaboration IDREFS #IMPLIED
  behavior IDREFS #IMPLIED
  partition IDREFS #IMPLIED
  context IDREFS #IMPLIED
  submachineState IDREFS #IMPLIED
  %XMI.element.att;
  %XMI.link.att;>
<!ELEMENT UML:Partition.contents (UML:ModelElement)* >

<!ELEMENT UML:Partition.activityModel (UML:ActivityModel)* >

<!ELEMENT UML:Partition (UML:ModelElement.name |
  UML:ModelElement.visibility | XMI.extension |
  UML:ModelElement.binding |
  UML:ModelElement.template |
  UML:ModelElement.templateParameter |
  UML:ModelElement.implementation |
  UML:ModelElement.view |
  UML:ModelElement.presentation |
  UML:ModelElement.namespace |
  UML:ModelElement.constraint |
  UML:ModelElement.requirement |
  UML:ModelElement.provision |
  UML:ModelElement.stereotype |
  UML:ModelElement.elementReference |
  UML:ModelElement.collaboration |
  UML:ModelElement.behavior |
  UML:ModelElement.partition |
  UML:Partition.contents |
  UML:Partition.activityModel |
  UML:ModelElement.taggedValue)* >

<!ATTLIST UML:Partition
  name CDATA #IMPLIED
  visibility (public | protected | private) #IMPLIED
  binding IDREFS #IMPLIED
  template IDREFS #IMPLIED
  templateParameter IDREFS #IMPLIED
  implementation IDREFS #IMPLIED
  view IDREFS #IMPLIED
  presentation IDREFS #IMPLIED
  namespace IDREFS #IMPLIED
  constraint IDREFS #IMPLIED
  requirement IDREFS #IMPLIED
  provision IDREFS #IMPLIED
  stereotype IDREFS #IMPLIED
  elementReference IDREFS #IMPLIED
  collaboration IDREFS #IMPLIED
  behavior IDREFS #IMPLIED
  partition IDREFS #IMPLIED
  contents IDREFS #IMPLIED
  activityModel IDREFS #IMPLIED
  XMI.element.att;
  XMI.link.att;>
<!-- CLASS: Behavioral_Elements.State_Machines.ObjectFlowState -->
<!ELEMENT UML:ObjectFlowState.typeState (UML:ClassifierInState)* >
<!ATTLIST UML:ObjectFlowState
name CDATA #IMPLIED
visibility (public | protected | private) #IMPLIED
binding IDREFS #IMPLIED
template IDREFS #IMPLIED
templateParameter IDREFS #IMPLIED
implementation IDREFS #IMPLIED
view IDREFS #IMPLIED
presentation IDREFS #IMPLIED
namespace IDREFS #IMPLIED
constraint IDREFS #IMPLIED
requirement IDREFS #IMPLIED
provision IDREFS #IMPLIED
stereotype IDREFS #IMPLIED
elementReference IDREFS #IMPLIED
collaboration IDREFS #IMPLIED
behavior IDREFS #IMPLIED
partition IDREFS #IMPLIED
outgoing IDREFS #IMPLIED
incoming IDREFS #IMPLIED
parent IDREFS #IMPLIED
stateMachine IDREFS #IMPLIED
deferredEvent IDREFS #IMPLIED
classifierInState IDREFS #IMPLIED
typeState IDREFS #IMPLIED
%XMI.element.att;
%XMI.link.att;

<!ELEMENT UML:ClassifierInState.objectFlowState (UML:ObjectFlowState)* >
<!ELEMENT UML:ClassifierInState.inState (UML:State)* >
<!ELEMENT UML:ClassifierInState.type (UML:Classifier)* >
<!ELEMENT UML:ClassifierInState (UML:ModelElement.name |
  UML:ModelElement.visibility | 
  UML:GeneralizableElement.isRoot | 
  UML:GeneralizableElement.isLeaf | 
  UML:GeneralizableElement.isAbstract | 
  XMI.extension | 
  UML:ModelElement.binding | 
  UML:ModelElement.template | 
  UML:ModelElement.templateParameter | 
  UML:ModelElement.implementation | 
  UML:ModelElement.view | 
  UML:ModelElement.presentation | 
  UML:ModelElement.namespace | 
  UML:ModelElement.constraint | 
  UML:ModelElement.requirement | 
  UML:ModelElement.provision | 
  UML:ModelElement.stereotype | 
  UML:GeneralizableElement.generalization | 
  UML:GeneralizableElement.specialization | 
  UML:Classifier.parameter | 
  UML:Classifier.structuralFeature | 
  UML:Classifier.specification | 
  UML:Classifier.realization | 
  UML:Classifier.associationEnd | 
  UML:Classifier.participant | 
  UML:Classifier.createAction | 
  UML:Classifier.instance | 
  UML:Classifier.collaboration | 
  UML:Classifier.classifierRole | 
  UML:ClassifierInState.objectFlowState | 
  UML:ClassifierInState.inState | 
  UML:ClassifierInState.type |
<!ATTLIST UML:ClassifierInState
  name CDATA #IMPLIED
  visibility (public | protected | private) #IMPLIED
  isRoot (false | true) #IMPLIED
  isLeaf (false | true) #IMPLIED
  isAbstract (false | true) #IMPLIED
  binding IDREFS #IMPLIED
  template IDREFS #IMPLIED
  templateParameter IDREFS #IMPLIED
  implementation IDREFS #IMPLIED
  view IDREFS #IMPLIED
  presentation IDREFS #IMPLIED
  namespace IDREFS #IMPLIED
  constraint IDREFS #IMPLIED
  requirement IDREFS #IMPLIED
  provision IDREFS #IMPLIED
  stereotype IDREFS #IMPLIED
  elementReference IDREFS #IMPLIED
  ModelElement.collaboration IDREFS #IMPLIED
  behavior IDREFS #IMPLIED
  partition IDREFS #IMPLIED
  generalization IDREFS #IMPLIED
  specialization IDREFS #IMPLIED
  parameter IDREFS #IMPLIED
  structuralFeature IDREFS #IMPLIED
  specification IDREFS #IMPLIED
  realization IDREFS #IMPLIED
  associationEnd IDREFS #IMPLIED
  participant IDREFS #IMPLIED
  createAction IDREFS #IMPLIED
  instance IDREFS #IMPLIED
  Classifier.collaboration IDREFS #IMPLIED
  classifierRole IDREFS #IMPLIED
  classifierInState IDREFS #IMPLIED
  objectFlowState IDREFS #IMPLIED
  inState IDREFS #IMPLIED
  type IDREFS #IMPLIED
  %XMI.element.att;
  %XMI.link.att;
>
<!-- _______________________________________________________________ -->
<!-- CLASS: Behavioral_Elements.State_Machines.ActivityState -->
<!-- _______________________________________________________________ -->
<!ELEMENT UML:ActivityState (UML:ModelElement.name |
  UML:ModelElement.visibility | XMI.extension |
  UML:ModelElement.binding | UML:ModelElement.template |
  UML:ModelElement.templateParameter | UML:ModelElement.implementation |
<!ATTLIST UML:ActivityState
  name CDATA #IMPLIED
  visibility (public | protected | private) #IMPLIED
  binding IDREFS #IMPLIED
  template IDREFS #IMPLIED
  templateParameter IDREFS #IMPLIED
  implementation IDREFS #IMPLIED
  view IDREFS #IMPLIED
  presentation IDREFS #IMPLIED
  namespace IDREFS #IMPLIED
  constraint IDREFS #IMPLIED
  requirement IDREFS #IMPLIED
  provision IDREFS #IMPLIED
  stereotype IDREFS #IMPLIED
  elementReference IDREFS #IMPLIED
  collaboration IDREFS #IMPLIED
  behavior IDREFS #IMPLIED
  partition IDREFS #IMPLIED
  outgoing IDREFS #IMPLIED
  incoming IDREFS #IMPLIED
  parent IDREFS #IMPLIED
  State.stateMachine IDREFS #IMPLIED
  deferredEvent IDREFS #IMPLIED
  classifierInState IDREFS #IMPLIED
  SubmachineState.stateMachine IDREFS #IMPLIED
%XMI.element.att;
%XMI.link.att;
>
<!ELEMENT UML:Foundation (UML:Auxiliary_Elements | UML:Core |
  UML:Extension_Mechanisms | UML:Data_Types)* >
<!ATTLIST UML:Foundation
  %XMI.element.att;
  %XMI.link.att;
<!ATTLIST UML:Auxiliary_Elements %XMI.element.att; %XMI.link.att;>

<!ATTLIST UML:Core %XMI.element.att; %XMI.link.att;>

<!ELEMENT UML:Extension_Mechanisms (UML:TaggedValue | UML:Stereotype)* >
<!ATTLIST UML:Extension_Mechanisms %XMI.element.att; %XMI.link.att;>

<!ATTLIST UML:Data_Types %XMI.element.att; %XMI.link.att;>

<!ATTLIST UML:Model_Management %XMI.element.att; %XMI.link.att;>

<!ATTLIST UML:Behavioral_Elements %XMI.element.att;>

<!ATTLIST UML:Common_Behavior
%XMI.element.att;
%XMI.link.att;
>


<!ATTLIST UML:Collaborations
%XMI.element.att;
%XMI.link.att;
>

<!ELEMENT UML:Use_Cases (UML:Actor | UML:UseCase | UML:UseCaseInstance)* >

<!ATTLIST UML:Use_Cases
%XMI.element.att;
%XMI.link.att;
>


<!ATTLIST UML:State_Machines
%XMI.element.att;
%XMI.link.att;
>
B.1 Introduction

The MOF 1.1 XMI 1.1 DTD included here uses Rule Set 1 rather than Rule Set 3, which was used for the previous DTD. To use the Corba XML elements that were included in the previous DTD, you will need to add them to the fixed DTD part of this DTD.

Appendix B contains a DTD generated by programmatically applying Rule Set 3 of the DTD Production Rules to a MOF model of itself. This model represents the MOF model that is described in Section 3 MOF Model and Interfaces of the MOF 1.1 specification. The MOF model is used for describing OMG compliant metamodels, but for the purposes of transferring metamodels, the MOF model itself is treated as a metamodel and therefore has a corresponding DTD. This DTD generally follows the specification of the above section on representing metamodel information. By examining this DTD, you can gain a better understanding of the types of metamodel information that can be represented in an XML DTD, and the information that cannot be specified.

The structure of the DTD closely corresponds to the document “MOF 1.1 Specification, 1 September 1997.” Each XML element corresponding to an MOF class has a comment indicating which section and pages of that document describe the class. You can verify the accuracy of the DTD against the document by reading the pages of the document in the comments and verifying that the encoding for them is correct.

B.2 MOF DTD

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!-- Generated by: XMI Framework 1.0 -->
<!-- Date: Mon Oct 25 17:35:30 PDT 1999 -->

<!-- _______________________________________________________________ -->
```
<!-- This section of the DTD contains XML declarations required by XMI. -->
<!ELEMENT XMI (XMI.header?, XMI.content?, XMI.difference*, XMI.extensions*)>
<!ATTLIST XMI
  xmi.version CDATA #FIXED "1.1"
  timestamp CDATA #IMPLIED
  verified (true | false) #IMPLIED>

<!ELEMENT XMI.header (XMI.documentation?, XMI.model*, XMI.metamodel*, XMI.metametamodel*, XMI.import*)>

<!ELEMENT XMI.documentation (#PCDATA | XMI.owner | XMI.contact | XMI.longDescription | XMI.shortDescription | XMI.exporter | XMI.exporterVersion | XMI.notice)*>
<!ELEMENT XMI.owner ANY>
<!ELEMENT XMI.contact ANY>
<!ELEMENT XMI.longDescription ANY>
<!ELEMENT XMI.shortDescription ANY>
<!ELEMENT XMI.exporter ANY>
<!ELEMENT XMI.exporterVersion ANY>
<!ELEMENT XMI.exporterID ANY>
<!ELEMENT XMI.notice ANY>

<!ELEMENT XMI.element.att>

<!ENTITY % XMI.element.att
<!-- XMI.link.att defines the attributes that each XML element that corresponds to a metamodel class must have to enable it to function as a simple XLink as well as refer to model constructs within the same XMI file. -->

<!ENTITY % XMI.link.att
    'href CDATA #IMPLIED xmi.idref IDREF #IMPLIED xml:link CDATA #IMPLIED xlink:actuate (true | false) #IMPLIED xlink:content-role CDATA #IMPLIED xlink:title CDATA #IMPLIED xlink:show (embed | replace | new) #IMPLIED xlink:behavior CDATA #IMPLIED'>

<!-- XMI.model identifies the model(s) being transferred -->

<!ELEMENT XMI.model ANY >
<!ATTLIST XMI.model
    %XMI.link.att;
    xmi.name CDATA #REQUIRED
    xmi.version CDATA #IMPLIED
>

<!-- XMI.metamodel identifies the metamodel(s) for the transferred data -->

<!ELEMENT XMI.metamodel ANY >
<!ATTLIST XMI.metamodel
    %XMI.link.att;
    xmi.name CDATA #REQUIRED
    xmi.version CDATA #IMPLIED
>

<!-- XMI.metametamodel identifies the metametamodel(s) for the transferred data -->

<!ELEMENT XMI.metametamodel ANY >
<!ATTLIST XMI.metametamodel
    %XMI.link.att;
    xmi.name CDATA #REQUIRED
>
xml.version CDATA #IMPLIED
>
<!-- _______________________________________________________________ -->
<!-- XMI.import identifies other files associated with the transferred data -->
<!-- _______________________________________________________________ -->
<!ELEMENT XMI.import ANY >
<!ATTLIST XMI.import
%XMI.link.att;
xml.name CDATA #REQUIRED
xml.version CDATA #IMPLIED
>
<!-- _______________________________________________________________ -->
<!-- XMI.content is the actual data being transferred -->
<!-- _______________________________________________________________ -->
<!ELEMENT XMI.content ANY >

<!-- _______________________________________________________________ -->
<!-- XMI.extensions contains data to transfer that does not conform to the metamodel(s) in the header -->
<!-- _______________________________________________________________ -->
<!ELEMENT XMI.extensions ANY >
<!ATTLIST XMI.extensions
xmi.extender CDATA #REQUIRED
>
<!-- _______________________________________________________________ -->
<!-- extension contains information related to a specific model construct that is not defined in the metamodel(s) in the header -->
<!-- _______________________________________________________________ -->
<!ELEMENT XMI.extension ANY >
<!ATTLIST XMI.extension
%XMI.element.att;
%XMI.link.att;
xmi.extender CDATA #REQUIRED
xmi.extenderID CDATA #IMPLIED
>
<!-- _______________________________________________________________ -->
<!-- XMI.difference holds XML elements representing differences to a base model -->
<!-- _______________________________________________________________ -->
<!ELEMENT XMI.difference (XMI.difference | XMI.delete | XMI.add |
<!ATTLIST XMI.difference
  %XMI.element.att;
  %XMI.link.att;
>
<!ATTLIST XMI.delete
  %XMI.element.att;
  %XMI.link.att;
>
<!ATTLIST XMI.add
  %XMI.element.att;
  %XMI.link.att;
xmi.position CDATA "-1"
>
<!ATTLIST XMI.replace
  %XMI.element.att;
  %XMI.link.att;
xmi.position CDATA "-1"
>
<!ATTLIST XMI.reference
  %XMI.link.att;
>
<!ATTLIST XMI
  xmlns:MOF CDATA #IMPLIED
<!ELEMENT XMI.any ANY >
<!ATTLIST XMI.any
  %XMI.link.att;
  xml.type CDATA #IMPLIED
  xml.name CDATA #IMPLIED
>

<!-- _______________________________________________________________ -->
<!-- -->
<!-- CLASS: Mof.Model.Feature -->
<!-- _______________________________________________________________ -->

<!ELEMENT MOF:Feature.visibility EMPTY >
<!ATTLIST MOF:Feature.visibility
  xml.value (public_vis | protected_vis | private_vis) #REQUIRED
>
<!ELEMENT MOF:Feature.scope EMPTY >
<!ATTLIST MOF:Feature.scope
  xml.value (instance_level | classifier_level) #REQUIRED
>

<!ATTLIST MOF:Feature
  name CDATA #IMPLIED
  annotation CDATA #IMPLIED
  visibility (public_vis | protected_vis | private_vis) #IMPLIED
  scope (instance_level | classifier_level) #IMPLIED
  container IDREFS #IMPLIED
  constraints IDREFS #IMPLIED
  %XMI.element.att;
  %XMI.link.att;
>

<!ATTLIST MOF:Classifier
    name CDATA #IMPLIED
    annotation CDATA #IMPLIED
    visibility (public_vis | protected_vis | private_vis) #IMPLIED
    isAbstract (true | false) #IMPLIED
    isRoot (yes | no | dont_care) #IMPLIED
    isLeaf (yes | no | dont_care) #IMPLIED
    container IDREFS #IMPLIED
    constraints IDREFS #IMPLIED
    supertypes IDREFS #IMPLIED
    %XMI.element.att;
    %XMI.link.att;
>

<!ATTLIST MOF:Namespace
    name CDATA #IMPLIED
    annotation CDATA #IMPLIED
    container IDREFS #IMPLIED
    constraints IDREFS #IMPLIED
    %XMI.element.att;
    %XMI.link.att;
>
<!ELEMENT MOF:StructuralFeature.multiplicity (MOF:Multiplicity) >
<!ELEMENT MOF:StructuralFeature.isChangeable EMPTY >
<!ATTLIST MOF:StructuralFeature.isChangeable
  xmi.value (true | false) #REQUIRED
>
<!ELEMENT MOF:StructuralFeature (MOF:ModelElement.name |
  MOF:ModelElement.annotation |
  MOF:Feature.visibility |
  MOF:Feature.scope |
  MOF:StructuralFeature.multiplicity |
  MOF:StructuralFeature.isChangeable |
  XMI.extension |
  MOF:ModelElement.container |
  MOF:ModelElement.constraints |
  MOF:TypedElement.type)* >
<!ATTLIST MOF:StructuralFeature
  name CDATA #IMPLIED
  annotation CDATA #IMPLIED
  visibility (public_vis | protected_vis | private_vis) #IMPLIED
  scope (instance_level | classifier_level) #IMPLIED
  isChangeable (true | false) #IMPLIED
  container IDREFS #IMPLIED
  constraints IDREFS #IMPLIED
  type IDREFS #IMPLIED
  %XMI.element.att;
  %XMI.link.att;
>
<!ELEMENT MOF:MofAttribute.isDerived EMPTY >
<!ATTLIST MOF:MofAttribute.isDerived
  xmi.value (true | false) #REQUIRED
>
<!ELEMENT MOF:MofAttribute (MOF:ModelElement.name |
  MOF:ModelElement.annotation |
  MOF:Feature.visibility | MOF:Feature.scope |
  MOF:StructuralFeature.multiplicity |
  MOF:StructuralFeature.isChangeable |
  MOF:MofAttribute.isDerived | XMI.extension |
  MOF:ModelElement.container |
  MOF:ModelElement.constraints |
  MOF:TypedElement.type)* >
<!ATTLIST MOF:MofAttribute
  name CDATA #IMPLIED
  annotation CDATA #IMPLIED
  visibility (public_vis | protected_vis | private_vis) #IMPLIED
  scope (instance_level | classifier_level) #IMPLIED
  isChangeable (true | false) #IMPLIED
  isDerived (true | false) #IMPLIED
  container IDREFS #IMPLIED
<!ELEMENT MOF:Operation.isQuery EMPTY >
<!ATTLIST MOF:Operation.isQuery
  xmi.value (true | false) #REQUIRED >

<!ELEMENT MOF:Operation.exceptions (MOF:MofException)* >

<!ELEMENT MOF:Operation (MOF:ModelElement.name |
  MOF:ModelElement.annotation |
  MOF:Feature.visibility | MOF:Feature.scope |
  MOF:Operation.isQuery | XMI.extension |
  MOF:ModelElement.container |
  MOF:ModelElement.constraints |
  MOF:Operation.exceptions |
  MOF:Namespace.contents)* >

<!ATTLIST MOF:Operation
  name CDATA #IMPLIED
  annotation CDATA #IMPLIED
  visibility (public_vis | protected_vis | private_vis) #IMPLIED
  scope (instance_level | classifier_level) #IMPLIED
  isQuery (true | false) #IMPLIED
  container IDREFS #IMPLIED
  constraints IDREFS #IMPLIED
  exceptions IDREFS #IMPLIED
  %XMI.element.att;
  %XMI.link.att; >

<!ELEMENT MOF:Class.isSingleton EMPTY >
<!ATTLIST MOF:Class.isSingleton
  xmi.value (true | false) #REQUIRED >

<!ELEMENT MOF:Class (MOF:ModelElement.name |
  MOF:ModelElement.annotation |
  MOF:GeneralizableElement.visibility |
  MOF:GeneralizableElement.isAbstract |
  MOF:GeneralizableElement.isRoot |
  MOF:GeneralizableElement.isLeaf |
B

\[\text{MOF:Class.isSingleton | XMI.extension |} \]
\[\text{MOF:ModelElement.container |} \]
\[\text{MOF:ModelElement.constraints |} \]
\[\text{MOF:GeneralizableElement.supertypes |} \]
\[\text{MOF:Namespace.contents)* >} \]

<!ATTLIST MOF:Class

name CDATA #IMPLIED
annotation CDATA #IMPLIED
visibility (public_vis | protected_vis | private_vis) #IMPLIED
isAbstract (true | false) #IMPLIED
isLeaf (yes | no | dont_care) #IMPLIED
isSingleton (true | false) #IMPLIED
container IDREFS #IMPLIED
constraints IDREFS #IMPLIED
supertypes IDREFS #IMPLIED
%XMI.element.att;
%XMI.link.att;

>

<!-- _______________________________________________________________ -->
<!-- -->
<!-- CLASS: Mof.Model.ModelElement -->
<!-- _______________________________________________________________ -->

<!ELEMENT MOF:ModelElement.name (#PCDATA | XMI.reference)* >

<!ELEMENT MOF:ModelElement.annotation (#PCDATA | XMI.reference)* >

<!ELEMENT MOF:ModelElement.container (MOF:Namespace)* >

<!ELEMENT MOF:ModelElement.constraints (MOF:Constraint)* >

<!ELEMENT MOF:ModelElement (MOF:ModelElement.name |

MOF:ModelElement.annotation | XMI.extension |

MOF:ModelElement.container |

MOF:ModelElement.constraints)* >

<!ATTLIST MOF:ModelElement

name CDATA #IMPLIED
annotation CDATA #IMPLIED
container IDREFS #IMPLIED
constraints IDREFS #IMPLIED
%XMI.element.att;
%XMI.link.att;

>

<!-- _______________________________________________________________ -->
<!-- -->
<!-- CLASS: Mof.Model.Package -->
<!-- _______________________________________________________________ -->

<!ELEMENT MOF:Package (MOF:ModelElement.name |

MOF:ModelElement.annotation |

MOF:GeneralizableElement.visibility |

MOF:GeneralizableElement.isAbstract |
<!ATTLIST MOF:Parameter
    name CDATA #IMPLIED
    annotation CDATA #IMPLIED
    direction (in_dir | out_dir | inout_dir | return_dir) #IMPLIED
    container IDREFS #IMPLIED
    constraints IDREFS #IMPLIED
    type IDREFS #IMPLIED
    %XMI.element.att;
    %XMI.link.att;
>
<!-- _______________________________________________________________ -->
<!-- -->
<!-- CLASS: Mof.Model.BehavioralFeature -->
<!-- -->
<!ELEMENT MOF:BehavioralFeature (MOF:ModelElement.name |
    MOF:ModelElement.annotation |
    MOF:Feature.visibility |
    MOF:Feature.scope | XMI.extension |
    MOF:ModelElement.container |
    MOF:ModelElement.constraints |
    MOF:Namespace.contents)* >

<!ATTLIST MOF:BehavioralFeature
    name CDATA #IMPLIED
    annotation CDATA #IMPLIED
    visibility (public_vis | protected_vis | private_vis) #IMPLIED
    scope (instance_level | classifier_level) #IMPLIED
    container IDREFS #IMPLIED
    constraints IDREFS #IMPLIED
    %XMI.element.att;
    %XMI.link.att;
>
<!-- _______________________________________________________________ -->
<!-- -->
<!-- CLASS: Mof.Model.Import -->
<!-- -->
<!ELEMENT MOF:Import.visibility EMPTY >
<!ATTLIST MOF:Import.visibility
    xmi.value (public_vis | protected_vis | private_vis) #REQUIRED
>
<!ELEMENT MOF:Import.importedNamespace (MOF:Namespace)* >
<!ELEMENT MOF:Import (MOF:ModelElement.name |
    MOF:ModelElement.annotation |
    MOF:Import.visibility | XML.extension |
    MOF:ModelElement.container |
    MOF:ModelElement.constraints |
    MOF:Import.importedNamespace)* >

<!ATTLIST MOF:Import
    name CDATA #IMPLIED
<!ELEMENT MOF:GeneralizableElement.visibility EMPTY>
<!ATTLIST MOF:GeneralizableElement.visibility
  xmi.value (public_vis | protected_vis | private_vis) #REQUIRED>

<!ELEMENT MOF:GeneralizableElement.isAbstract EMPTY>
<!ATTLIST MOF:GeneralizableElement.isAbstract
  xmi.value (true | false) #REQUIRED>

<!ELEMENT MOF:GeneralizableElement.isRoot EMPTY>
<!ATTLIST MOF:GeneralizableElement.isRoot
  xmi.value (yes | no | dont_care) #REQUIRED>

<!ELEMENT MOF:GeneralizableElement.isLeaf EMPTY>
<!ATTLIST MOF:GeneralizableElement.isLeaf
  xmi.value (yes | no | dont_care) #REQUIRED>

<!ELEMENT MOF:GeneralizableElement.supertypes (MOF:GeneralizableElement)*>


<!ATTLIST MOF:GeneralizableElement
  name CDATA #IMPLIED
  annotation CDATA #IMPLIED
  visibility (public_vis | protected_vis | private_vis) #IMPLIED
  isAbstract (true | false) #IMPLIED
  isRoot (yes | no | dont_care) #IMPLIED
  isLeaf (yes | no | dont_care) #IMPLIED
<!ATTLIST MOF:ModelElement
name CDATA #IMPLIED
annotation CDATA #IMPLIED
container IDREFS #IMPLIED
constraints IDREFS #IMPLIED
type IDREFS #IMPLIED
%XMI.element.att;
%XMI.link.att;
>
<!ELEMENT MOF:ModelElement.referencedEnd (MOF:AssociationEnd)* >
<!ATTLIST MOF:Reference
name CDATA #IMPLIED
annotation CDATA #IMPLIED
visibility (public_vis | protected_vis | private_vis) #IMPLIED
scope (instance_level | classifier_level) #IMPLIED
isChangeable (true | false) #IMPLIED
container IDREFS #IMPLIED
constraints IDREFS #IMPLIED
type IDREFS #IMPLIED
<!ATTLIST MOF:MofException
  name CDATA #IMPLIED
  annotation CDATA #IMPLIED
  visibility (public_vis | protected_vis | private_vis) #IMPLIED
  scope (instance_level | classifier_level) #IMPLIED
  container IDREFS #IMPLIED
  constraints IDREFS #IMPLIED
  %XMI.element.att;
  %XMI.link.att;>

<!ELEMENT MOF:DataType.typeCode ANY >
<!ELEMENT MOF:DataType (MOF:ModelElement.name | MOF:ModelElement.annotation | MOF:GeneralizableElement.visibility | MOF:GeneralizableElement.isAbstract | MOF:GeneralizableElement.isRoot | MOF:GeneralizableElement.isLeaf | MOF:DataType.typeCode | XMI.extension | MOF:ModelElement.container | MOF:ModelElement.constraints | MOF:GeneralizableElement.supertypes | MOF:Namespace.contents)* >
<!ATTLIST MOF:DataType
  name CDATA #IMPLIED
  annotation CDATA #IMPLIED
  visibility (public_vis | protected_vis | private_vis) #IMPLIED
  isAbstract (true | false) #IMPLIED
  isRoot (yes | no | dont_care) #IMPLIED
  isLeaf (yes | no | dont_care) #IMPLIED
  typeCode CDATA #IMPLIED
  container IDREFS #IMPLIED
  constraints IDREFS #IMPLIED
  %XMI.element.att;
  %XMI.link.att;>
supertypes IDREFS #IMPLIED
%XMI.element.att;
%XMI.link.att;

<!-- _______________________________________________________________ -->
<!-- -->
<!-- CLASS: Mof.Model.AssociationEnd -->
<!-- _______________________________________________________________ -->

<!ELEMENT MOF:AssociationEnd.multiplicity (MOF:Multiplicity) >
<!ELEMENT MOF:AssociationEnd.aggregation EMPTY >
<!ATTLIST MOF:AssociationEnd.aggregation
  xmi.value (none | shared | composite) #REQUIRED
>
<!ELEMENT MOF:AssociationEnd.isNavigable EMPTY >
<!ATTLIST MOF:AssociationEnd.isNavigable
  xmi.value (true | false) #REQUIRED
>
<!ELEMENT MOF:AssociationEnd.isChangeable EMPTY >
<!ATTLIST MOF:AssociationEnd.isChangeable
  xmi.value (true | false) #REQUIRED
>
<!ELEMENT MOF:AssociationEnd.otherEnd (MOF:AssociationEnd) >

<!ELEMENT MOF:AssociationEnd (MOF:ModelElement.name |
  MOF:ModelElement.annotation |
  MOF:AssociationEnd.multiplicity |
  MOF:AssociationEnd.aggregation |
  MOF:AssociationEnd.isNavigable |
  MOF:AssociationEnd.isChangeable |
  MOF:AssociationEnd.otherEnd |
  XMI.extension | MOF:ModelElement.container |
  MOF:ModelElement.constraints |
  MOF:TypedElement.type)* >

<!ATTLIST MOF:AssociationEnd
  name CDATA #IMPLIED
  annotation CDATA #IMPLIED
  aggregation (none | shared | composite) #IMPLIED
  isNavigable (true | false) #IMPLIED
  isChangeable (true | false) #IMPLIED
  container IDREFS #IMPLIED
  constraints IDREFS #IMPLIED
  type IDREFS #IMPLIED
  %XMI.element.att;
  %XMI.link.att;

<!-- _______________________________________________________________ -->
<!-- -->
<!-- CLASS: Mof.Model.TypedElement -->
<!-- _______________________________________________________________ -->
<!ELEMENT MOF:TypeAlias.multiplicity (MOF:Multiplicity) >
<!ELEMENT MOF:TypeAlias (MOF:ModelElement.name |
     MOF:ModelElement.annotation |
     MOF:TypeAlias.multiplicity | XMI.extension |
     MOF:ModelElement.container |
     MOF:ModelElement.constraints |
     MOF:TypedElement.type)* >
<!ATTLIST MOF:TypeAlias
     name CDATA #IMPLIED
     annotation CDATA #IMPLIED
     container IDREFS #IMPLIED
     constraints IDREFS #IMPLIED
     type IDREFS #IMPLIED
     %XMI.element.att;
     %XMI.link.att;
>
<!-- _______________________________________________________________ -->
<!-- -->
<!-- CLASS: Mof.Model.Tag -->
<!-- _______________________________________________________________ -->
<!ELEMENT MOF:Tag.tagId (#PCDATA | XMI.reference)* >
<!ELEMENT MOF:Tag.values (XMI.any) >
<!ELEMENT MOF:Tag.elements (MOF:ModelElement)* >
<!ELEMENT MOF:Tag (MOF:ModelElement.name | MOF:ModelElement.annotation |
     MOF:Tag.tagId | MOF:Tag.values | XMI.extension |
     MOF:ModelElement.container |
     MOF:ModelElement.constraints | MOF:Tag.elements)* >
<!ATTLIST MOF:Tag
     name CDATA #IMPLIED
     annotation CDATA #IMPLIED
     tagId CDATA #IMPLIED
     container IDREFS #IMPLIED
     constraints IDREFS #IMPLIED
     elements IDREFS #IMPLIED
     %XMI.element.att;
     %XMI.link.att;
>
<!ELEMENT MOF:Mof (MOF:Model)* >
<!ATTLIST MOF:Mof
     %XMI.element.att;
     %XMI.link.att;
>
<!ELEMENT MOF:Model (MOF:Feature | MOF:Classifier | MOF:Namespace |
     MOF:StructuralFeature | MOF:MofAttribute |
     MOF:Operation | MOF:Class | MOF:ModelElement |
     MOF:Package | MOF:Constraint | MOF:Association |
<!ATTLIST MOF:Model
  %XMI.element.att;
  %XMI.link.att;
>

MOF:Parameter | MOF:BehavioralFeature | MOF:Import |
MOF:GeneralizableElement | MOF:Constant |
MOF:Reference | MOF:MofException | MOF:DataType |
MOF:AssociationEnd | MOF:TypedElement |
MOF:Multiplicity | MOF:TypeAlias | MOF:Tag)* >
Example Model Encodings

C

C.1 Introduction

This appendix provides the following example UML models and their resultant XML encoding.

• An example of a department at a university.
• An example of a model of a schedule, using the department and a campus model.
• An example of instances of a schedule.
• An example using an IDL datatype metamodel for “complete” datatype metamodels.

C.2 Example 1 - Department

This is an example of a model of a department, using the UML DTD for UML 1.3.
The Department class has a relationship called instructor to Instructors, a superclass for Professor, Postdoc, Lecturer, and Teaching Assistant.
In the above example, the namespace of UML is used in the namespace declaration in the XMI element as well as the XMI.metamodel tag. The model’s name is Department, which will also serve as its namespace in example 3. Each class is listed in the content, including superclass references. For simplicity in the example, the ids were chosen to be the same as the name. The association between Instructor and Department is also shown.

C.3 Example 2 - Campus and Schedule

This example contains three models, the department above, a campus, and a schedule which uses both models.

The Campus model has a Campus associated with Facilities, which have subclasses of Hall and Classroom. The Campus and Department models are not related, but the Schedule model uses both.
File="Campus.xml" Namespace="Campus":
<XMI version="1.1" xmlns:UML="org.omg/UML1.3">
  <XMI.header>
    <XMI.model xmi.name="Campus" href="Campus.xml"/>
    <XMI.metamodel xmi.name="UML" href="UML.xml"/>
  </XMI.header>
  <XMI.content>
    <UML:Class name="Campus" xmi.id="Campus"/>
    <UML:Class name="Facility" xmi.id="Facility"/>
    <UML:Class name="Classroom" xmi.id="Classroom" generalization="Facility"/>
    <UML:Class name="Hall" xmi.id="Hall" generalization="Facility"/>
    <UML:Association>
      <UML:AssociationEnd name="classrooms" type="Facility"/>
      <UML:AssociationEnd name="school" type="Campus"/>
    </UML:Association>
  </XMI.content>
</XMI>
The schedule model has a Schedule based on Courses, where each Course is associated with a Facility from the Campus model and an Instructor from the Department model.

File="Schedule.xml" Namespace="Schedule":
<XMI version="1.1" xmlns:UML="org.omg/UML1.3">
  <XMI.header>
    <XMI.model xmi.name="Schedule" href="Schedule.xml"/>
    <XMI.metamodel xmi.name="UML" href="UML.xml"/>
    <XMI.import name="Department" href="Department.xml"/>
    <XMI.import name="Campus" href="Campus.xml"/>
  </XMI.header>
  <XMI.content>
    <UML:Class name="Schedule" xmi.id="Schedule"/>
    <UML:Class name="Course" xmi.id="Course"/>
    <UML:Association>
      <UML:Association.connection>
        <UML:AssociationEnd name="courses" type="Course"/>
        <UML:AssociationEnd name="schedule" type="Schedule"/>
      </UML:Association.connection>
    </UML:Association>
    <UML:Association>
      <UML:Association.connection>
        <UML:AssociationEnd name="instructors">
          <UML:AssociationEnd.type>
            <UML:Classifier href="Department.xml#Instructor"/>
          </UML:AssociationEnd.type>
        </UML:AssociationEnd>
        <UML:AssociationEnd name="courses" type="Course"/>
      </UML:Association.connection>
    </UML:Association>
    <UML:Association>
      <UML:Association.connection>
        <UML:AssociationEnd name="location">
          <UML:AssociationEnd.type>
            <UML:Classifier href="Campus.xml#Facility"/>
          </UML:AssociationEnd.type>
        </UML:AssociationEnd>
        <UML:AssociationEnd name="courses" type="Course"/>
      </UML:Association.connection>
    </UML:Association>
  </XMI.content>
</XMI>

Since the Campus and Department classes are in different files, XMI.import statements are generated as well as cross-document linking proxies for the AssociationEnd’s type.

### C.4 Example 3 - Instances of Schedule

In this example, each of the models are instantiated. The models are referred to using the XMI.metamodel tags and the namespace declarations are used in the XMI element.

Chemistry is an instance of department:
The Department is considered the metamodel for Chemistry. The Department namespace is used and is listed in XML.metamodel. This example shows Chemistry is a Department with four instances of Instructor’s subclasses, Professor Smith, Postdoc Jones, Lecturer Visitor, and TeachingAssistant Fred.

UCLA is an instance of the Campus model. The Campus namespace is used and is listed in XML.metamodel. UCLA is a Campus with three instances of Facility’s subclasses, Classroom 101, Classroom 202, and Hall Branner Hall.

Spring2000 is an instance of Schedule that uses UCLA, an instance of Campus, and Chemistry, an instance of Department. The Schedule namespace is used and is listed in XML.metamodel. Also, Schedule has imported Campus and Department as declared in the XMI.import elements. Schedule and Campus are also used as metamodels for Spring2000, and are also declared in the XMI element and in the XML.metamodel elements. Spring2000 uses UCLA and Chemistry, as listed in the XMI.import elements.
Spring2000 is an instance of Schedule which has a Course instance Chemistry 1 with instructors Professor Smith and TeachingAssistant Fred from the Department instance in Chemistry.xml and location Hall Branner Hall from the Campus instance in UCLA.xml.

C.5 Example 4 - “Complete” datatype metamodel.

As an example, suppose we have a model called Mail of letters and envelopes. The model consists of two classes called Letter and Envelope, where Envelope contains Letter. Envelope has two attributes, toAddress and fromAddress, both of type Address. Address is an IDL struct with fields name, streetNumber, street, city, state, and zip, all of IDL type string.
As an example of the standard IDL metamodel, we will use the Base IDL metamodel from the Corba Components Model (CCM) specification, orbos/99-07-02, Volume 2, Chapter 2.4.1. The Base IDL metamodel should subclass MOF:DataType. The following is the declaration of StructDef directly from the submission.

```
<!ELEMENT BaseIDL.StructDef.members (BaseIDL.Field)>  
<!ENTITY % BaseIDL.StructDefProperties 
('%BaseIDL.TypedefDefProperties;,(BaseIDL.StructDef.members)*')>  
<!ENTITY % BaseIDL.StructDefAssociations 
('%BaseIDL.TypedefDefAssociations;')>  
<!ELEMENT BaseIDL.StructDef ( %BaseIDL.StructDefProperties;,(XMI.extension*, %BaseIDL.StructDefAssociations; ) )*?>  
<!ATTLIST BaseIDL.StructDef %XMI.element.att; %XMI.link.att; >
```

Since Address is a definition of a struct, it is an instance of the IDL metaclass StructDef.

The XMI document for the Mail model uses the UML DTD for Letter and Envelope and the IDL DTD for Address:
The DTD that would be generated for the Mail model is:
An XMI document instance of the Mail model is:

```xml
<xmi version="1.1">
  <xmi.header>
    <xmi.model xmi.name="myMail" href="myMail.xml"/>
    <xmi.metamodel xmi.name="Mail" href="mail.xml"/>
  </xmi.header>
  <xmi.content>
    <envelope xmi.id="myEnvelope" letter="myLetter">
      <envelope.toAddress name="Sridhar" streetNumber="25725"
       street="Jeronimo" city="Mission Viejo" state="CA" zip="92691"/>
      <envelope.fromAddress name="Steve" streetNumber="555"
       street="Bailey" city="San Jose" state="CA" zip="95141"/>
    </envelope>
    <letter xmi.id="myLetter" envelope="myEnvelope"/>
  </xmi.content>
</xmi>
```
References


/XML] XML, a technical recommendation standard of the W3C. http://www.w3.org/TR/REC-xml

[NAMESP] Namespaces, a working draft of the W3C. http://www.w3.org/TR/WD-xml-names

[XLINK] XLinks, a working draft of the W3C. http://www.w3.org/TR/WD-xlink and
http://www.w3.org/TR/NOTE-xlink-principles

[XPointer] XPointer, working draft of the W3C. http://www.w3.org/TR/WD-xptr

[RDF] RDF, a working draft of the W3C. http://w3c.org/RDF/

[RDFSCEM] RDF-Schema, a working draft of the W3C. http://www.w3.org/TR/WD-rdf-schema

[XMLDATA] XML-Data, a note for discussion purposes to the W3C. http://www.w3.org/TR/1998/NOTE-XML-
data. DCD supersedes XML-Data.


[XSL] XSL, a working draft of the W3C. http://www.w3.org/Style/XSL/

[DOM] DOM, a working draft of the W3C. http://www.w3.org/DOM/


[UML] UML, an adopted standard of the OMG. http://www.omg.org

[MOF] MOF, an adopted standard of the OMG. http://www.omg.org

[XMLJAVA] XML for Java, a free, complete, commercial XML parser written in Java by IBM.
http://www.alphaworks.ibm.com/formula/xml
The following is the XML specification's reference to its character set standards:


The following is the XML specification’s reference to its character set standards:


The following is the Open Group DCE standard on UUIDs.

[UUID] CAE Specification
DCE 1.1: Remote Procedure Call
Document Number: C706
http://www.opengroup.org/onlinepubs/9629399/toc.htm
http://www.opengroup.org/onlinepubs/9629399/apdxa.htm (Definition/creation of UUIDs).
Compatibility and Conformance

E.1 Compatibility with Other Standards

The XMI specification addresses the metadata interchange requirement of the OMG repository architecture which is described in the OMG MOF specification (ad/97-10-02, Section 1.3) and corresponds to the 'Data Interchange' component of the architecture. The XMI specification conforms to the following standards:

- **XML**, the Extensible Markup Language, is a new data format for electronic interchange designed to bring structured information to the web. XML is an open technology standard of the W3C (www.w3c.org), the standards group responsible for maintaining and advancing HTML. XML is used as the concrete syntax and transfer format for OMG MOF compliant metadata.

There are several benefits of basing metamodel interchange on XML. XML is an open standard, platform and vendor independent. XML supports the international character set standards of extended ISO Unicode. XML is metamodel-neutral and can represent metamodels compliant with OMG’s meta-metamodel, the MOF. XML is programming language-neutral and API-neutral. XML APIs are provided in additional standards, giving the user an open choice of several access methods to create, view, and integrate XML information. Leading XML APIs include DOM, SAX, and WEB-DAV.

- **MOF**, the Meta Object Facility is an OMG (www.omg.org) metadata interface standard that can be used to define and manipulate a set of interoperable metamodels and their instances (models). The MOF also defines a simple meta-metamodel (based on the OMG UML - Unified Modeling Language) with sufficient semantics to describe metamodels in various domains starting with the domain of object analysis and design. The XMI specification uses MOF as the meta-metamodel to ensure transfer of any MOF compliant metamodel (such as UML) and instances of these metamodels - the models themselves.
UML, the Unified Modeling Language is an OMG (www.omg.org) standard modeling language for specification, construction, visualization and documentation of the artifacts of a software system. The XMI can be used to exchange UML models between tools and between tools and repositories.

The CORBA interfaces specified in the MOF (ad/97-10-02, ad/97-10-03) can be used to internalize and externalize XML streams of MOF based metamodels. (See the interface MOF::Package in ad/97-10-02) for more details. In this sense, the XMI together with the MOF conforms to the OMA and can be used as the foundation for developing web based distributed development environments.

In summary the XMI supports W3C XML, OMG MOF, UML and OMA standards. There are no dependencies on any other standards.

### E.1.1 XMI and W3C DCD

IBM and Microsoft have collaborated and proposed a new W3C proposal based on XML - Document Content Definition (DCD). DCD is richer than DTD and has better data structuring and data typing capabilities, thus making it an attractive target for XMI in the future. The XMI submitters anticipate that when the DCD specification solidifies, mappings from XMI to DCD will be produced as an evolution of XMI. Another W3C initiative that could influence future XMI direction is XML-Schema work that is getting underway.

### E.1.2 XMI and CDIF

EIA CDIF (Electronic Industry Associates - Case Data Interchange Format) was proposed as one of the initial submissions to the SMIF RFP. The CDIF submitters have collaborated with the XMI submitters to incorporate key aspects of CDIF such as the use of unique IDs into the XMI specification. Preliminary work on providing a migration path from CDIF to XMI has begun and technical feasibility has been assessed.

It is anticipated that additional work is required to provide a migration path from existing metadata interchange standards (such as EIA CDIF - Electronics Industry Associates Case Data Interchange Format) to XMI should such a market requirement exist. The submitters believe that such a migration path is possible based on

1. Implementation experience on CDIF
2. The MOF meta-metamodel has all the modeling concepts needed to represent the CDIF meta-metamodel and provide appropriate transformation algorithms from CDIF to MOF and vice-versa. This analysis has been made by CDIF and MOF experts between the times of initial and final submission. The experience of CDIF designers in metamodeling architectures as well as the experience in the unique ID for meta information has been worked into the current XMI specification. The work on a migration path from CDIF Transfer Format to XMI is expected to be done in the by the OMG Object Analysis and Design Task Force.

The co-submitters and supporters of the CDIF proposal have helped improve the XMI specification and are now part of the final submission team for this XMI specification.
E.2 Conformance Issues

E.2.1 Introduction

This section describes the required and optional points of compliance with the XMI specification. The term “XML recommendation” refers to technical recommendations by the W3C for XML version 1.0 and later [XML reference] [W3C reference].

E.2.2 Required Compliance

**XMI DTD Compliance**

XMI DTDs are required to conform to the following points:

- The XMI DTD(s), both internal and external, must be “valid” and “well-formed” as defined by the XML recommendation. [XMI reference]
- The determination of compliance on a DTD is made in the “expanded form” where all entity information is expanded out. Many variations of entity declarations result in the same “expanded form” DTD, each variation having identical compliance.
- The expanded form of an XMI DTD must follow the processing and fixed element declarations of Section 3.2.2, “Requirements for XMI DTDs,” on page 3-3, Section 3.4, “XMI DTD and Document Structure,” on page 3-5, and Section 3.5, “Necessary XMI DTD Declarations,” on page 3-6.
- An expanded form XMI DTD must have the “same” set of elements as those which are created in expanded form using the rule set from Chapter 3. The definition of “same” for two DTDs is that there is an exact one to one correspondence between the elements in each DTD, each correspondence identical in terms of element name, element attributes (name, type, and default actions), element content specification, content grammar, and content multiplicities.

**XMI Document Compliance**

XMI Documents are required to conform to the following points:

- The XMI document must be “valid” and “well-formed” as defined by the XML recommendation [XMI reference], whether used with or without the document’s corresponding XMI DTD(s). Although it is optional not to transmit and/or validate a document with its XMI DTD(s), the document must still conform as if the check had been made.
- The XMI document must contain the XML declarations and processing instructions as defined in Section 3.4, “XMI DTD and Document Structure,” on page 3-5.
• The XMI document must contain one or more XMI root elements that together contain all other XMI information within the document as defined in Section 3.5, “Necessary XMI DTD Declarations,” on page 3-6.

• The XMI document must be the “same” as a document following the document production rules of Chapter 4. The definition of “same” for two documents is that there is an exact one to one correspondence between the elements in each document, each correspondence identical in terms of element name, element attributes (name and value), and contained elements. Elements declared within the XMI.documentation, XMI.extension, and XMI.extensions elements are excepted.

**Usage Compliance**

The XMI documents must be used under the following conditions:

• The XML parsers, browsers, or other tools used to input and/or output XMI information must conform to the XML recommendation [XMI reference]. Note that early releases of many tools are not fully XML version 1.0 compliant.

**E.2.3 Optional Compliance Points**

**XMI MOF Subset**

XMI support for MOF meta-models (at the M2 level only) beyond the following subset is optional:

• Data types not contained explicitly within the metamodel.
• Metamodels having different names for MOF reference ends as association ends.
• Metamodels having association ends without references.
• Metamodels containing static attributes.
• Metamodels with nested classes.

**XMI DTD Compliance**

XMI DTDs optionally conform to the following points:

• The definition of XML entities within DTDs are suggested to follow the design rules in Chapter 3 and Chapter 4.

• Incomplete model DTD generation rules may be used to support transmission of incomplete models. Either all incomplete rules or no incomplete rules should be supported. The incomplete model DTD is a different DTD than the complete model DTD. Support for incomplete models is an optional addition to the mandatory support for complete models.

• Contained elements may optionally have a role for their container with lower bound multiplicity of zero.
XMI Document Compliance

XMI Documents optionally conform to the following points:

- The guidelines for using the XMI.extension and XMI.extensions elements are suggested in Chapter 3. Tools should place their extended information within the designated extension areas, declare the nature of the extension using the standard XMI elements where applicable, and preserve the extensions of other tools where appropriate.

- Processing of XMI differencing elements is an optional compliance point. Either all differencing elements are produced and processed, or no differencing elements are produced and processed.

- Documents may support the incomplete model DTD or the complete model DTD.

- Contained elements may optionally have a role for their container with lower bound multiplicity of zero.

Usage Compliance

The XMI documents are optionally used under the following conditions:

- The XML parsers, browsers, or other tools used to input and/or output XMI information should conform to standard APIs for the XML recommendation [XMI reference]. These APIs include, but are not limited to, DOM [DOM reference], SAX [SAX reference], and Web-DAV [Web-DAV reference].

- Note that the early releases of many tools are not fully XML version 1.0 compliant. Check for updated versions of the tools or use the references as a guide for locating compliant tools.
Index

P
Package 1-4, 5-8
    import 1-4
    inheritance 1-4
    nested 1-4
Parameter 1-4
    production rules 6-1

Q
quotation marks 4-2

R
RDF 1-12
    Reference model viii
    reuse 1-4
    Rule Set 1
        Simple DTD 4-4
    Rule Set 1 4-4
    Rule Set 10 4-11
    Rule Set 11 4-12
    Rule Set 12 4-12
    Rule Set 2 4-5
    Rule Set 3 4-6
    Rule Set 4 4-6
    Rule Set 5 4-7
    Rule Set 6 4-8
    Rule Set 7 4-9
    Rule Set 8 4-10
    Rule Set 9 4-10

S
SGML 1-7
    Standard elements 5-13
    syntax
        abstract 1-3

T
tag 1-9
    balanced pairs of 1-9
    name 1-9
    Tag-Value 3-18
    timestamp 3-9
    Token class 5-7
    Transmitting Incomplete Metadata 3-20
        Interchange of modern fragments 3-20
        XMI encoding 3-20
    Transmitting Metadata Differences 3-24
        Definitions 3-25
        Differences 3-25
        XMI encoding 3-25

U
qualified name 4-2
    UML 1-6
    uniqueness 1-4
    URI 1-10
    UUID 3-6
    UUIDrefs 3-22

V
Value 3-18
verified 3-9

W
W3C, the 1-1, 1-7
white space 4-2

X
XLink
    XPointer 1-12
    XLinks 3-22
    XMI 1-1, 3-9
        Common Elements 3-8
            document production rules 1-1
            DTD production rules 1-1
            use for data interchange 1-11
    xmi
        extender 3-10
        extenderID 3-11
        id 3-6
        idref 3-8
        label 3-6
        name 3-11, 3-12, 3-13
        position 3-13, 3-14
        uuid 3-6
        uuidref 3-8
        version 3-9, 3-11, 3-12, 3-13
    XMI DTD
        Declarations 3-6
        Document Structure 3-5
    XMI DTDs
        Requirements 3-3
        XMI element 3-9
        XMI.add 3-13
        XMLcontent 3-10
        XMI.delete 3-13
        XMI.difference 3-13
        XMI.documentation 3-11
        XMI.element.att 3-6
        XML.extension 3-10
        XML.extensions 3-10
        XML.header 3-10
        XML.import 3-12
        XML.link.att 3-7
        XML.metametamodel 3-12
        XML.metamodel 3-12
        XML.model 3-11
        XML.reference 3-14
        XML.replace 3-14
        XMLDataType 3-18
        XMIEnumSet 3-18
        XML 1-1, 1-7
            APIs 1-8, 1-12
            benefits of using 1-7
            display of 1-7
            links 1-12
            low cost of entry 1-8
            namespaces 1-12
            overview of 1-8
            uptake of 1-8
        XML Declarations 3-3
        XML document
Index

semantics of 1-11
valid 1-10
well-formed 1-10
XML DTDs 3-2

XML Validation of XMI documents 3-3
XML-Data 1-12
XPointer 3-22
XSL 1-7, 1-12