XMI for FAMIX

Implementation of a prototype for generating XMI documents based on the FAMIX meta-model

Informatikprojekt

von
Michael Freidig
freidig@iam.unibe.ch

Supervised by Oscar Nierstrasz
Software Composition Group

Institut für Informatik und angewandte Mathematik
Universität Bern, Neubrückstr. 10 Bern

Abstract

In the context of the FAMOOS project it is necessary to transfer object-oriented models of software systems between different analysis tools. These models conform to the meta-model FAMIX, which is a model of the source code of a software system. In this project the OMG standard XMI (XML Metadata Interchange) is used to map the FAMIX meta-model to an XML DTD and generate XML files based on the model that is transferred. Because XMI is based on MOF (Meta Object Facility) meta-models, FAMIX is defined as a MOF meta-model. Based on these concepts, a prototype in Java is implemented that reads data from the Java reflective-interface and uses the XMI standard to generate an XML document. Because any model that has a MOF compliant meta-model can be exchanged with XMI, the prototype uses a generic approach by implementing the MOF interfaces and instantiating the FAMIX meta-model from the MOF model. This architecture can be reused for systems that use meta-models different from FAMIX. In order to test the correctness of the model-data after the transfer, a test program is implemented, that verifies the syntax and the content of generated XMI documents.

Table of Contents

1 Introduction

1.1 Motivation and Goal of the Project

The Extensible Markup Language (XML) is likely to become the defacto standard for transferring information between applications. Next to that the Meta-Object Facility (MOF) [MOF99] is likely to become the defacto standard to describe meta-models such as the models behind UML and CORBA. The third standard in the works is XMI [XMI99], a standardized way to exchange models based on the MOF and using XML.
Within the FAMOOS project it is necessary to transfer information that is described in the information model FAMIX between software analysis tools. Currently this information is transferred in the CDIF format. Because XMI is the emerging standard for exchanging meta-data the steps required for using XMI together with FAMIX should be examined.

The goal of the project is to build a prototype of a system that is capable of generating XMI documents ([XMI98] XML Generation Principles) for the meta-model FAMIX that is described as MOF meta-model ([MOF99] 2.4 Meta-models and mapping). Since FAMIX is a source code model, that means it defines a data model for object-oriented concepts – Class, Method, Attribute and InheritanceDefinition- plus Invocation and Access, it is necessary to read source code information from a datasource and transform it according to the naming convention that is defined by FAMIX. The system should then generate an XML document and its DTD according to the XMI standard. Furthermore a test program should then test the correctness of the syntax and the content of the generated XMI document. Therefore the following goals are defined:

1. FAMIX should be defined as a MOF meta-model by instantiating MOF classes so that it can be used as source of meta-data in the generation process of XMI documents. In order to query the meta-data that is defined as MOF meta-model, XMI requires the implementation of the MOF interfaces.
2. Generators for creating XMI documents and a DTD for FAMIX are implemented using the rules according to the XMI specification.
3. Java is the target language for transferring information and also the implementation language. Source code information is extracted using Java reflective interface [SUN99].
4. A testing tool should be implemented to test the correctness of the generation process.

1.2 Overview of XMI for FAMIX

The main purpose of XMI is to enable easy interchange of metadata between modeling tools and MOF based metadata repositories in distributed heterogeneous environments.
XMI integrates two industry standards:

- XML - extensible Markup Language, a W3C standard
- MOF - Meta Object Facility, an OMG meta-modeling and metadata repository standard

The MOF standard defines a 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.

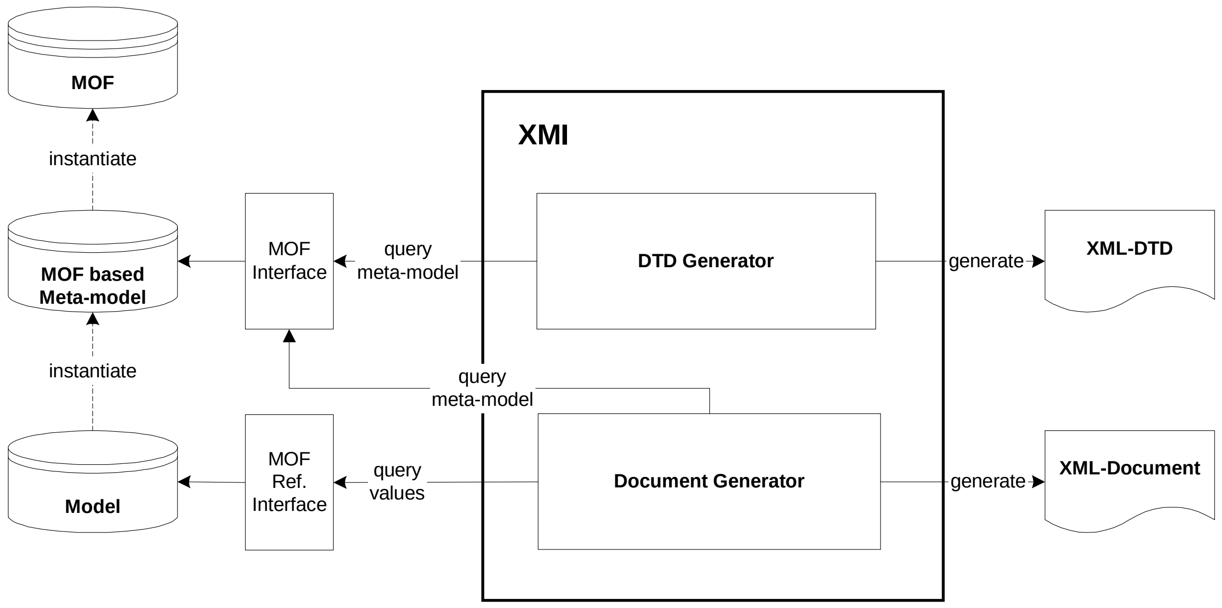


Figure 1: Architectural Overview of XMI

The MOF is an object-oriented meta-meta-model that defines a class model that can be instantiated to define different meta-models. A MOF based meta-model is a meta-model that is described by instances of classes from the MOF. This meta-model is the meta-data for a model that is exchanged. The principle of XMI is that it maps a MOF meta-model to an XML-DTD and a model to an XML-Document. XMI consists of two parts

- A set of XML Document Type Definition (DTD) production rules for transforming MOF based meta-models into XML DTDs
- A set of XML Document production rules for encoding and decoding MOF based metadata

The DTD Generator is a component that implements the DTD production rules and uses the MOF Interfaces to query information from the MOF meta-model. The MOF Interfaces allows a client to query the structure of the meta-model such as selecting all classes of a package, all attributes of a class, inheritance definition etc.
The Document Generator is a component that implements the document production rules for creating an XML document based on objects that are instance of classes of the meta-model. In order to query values the MOF Reflective interface.

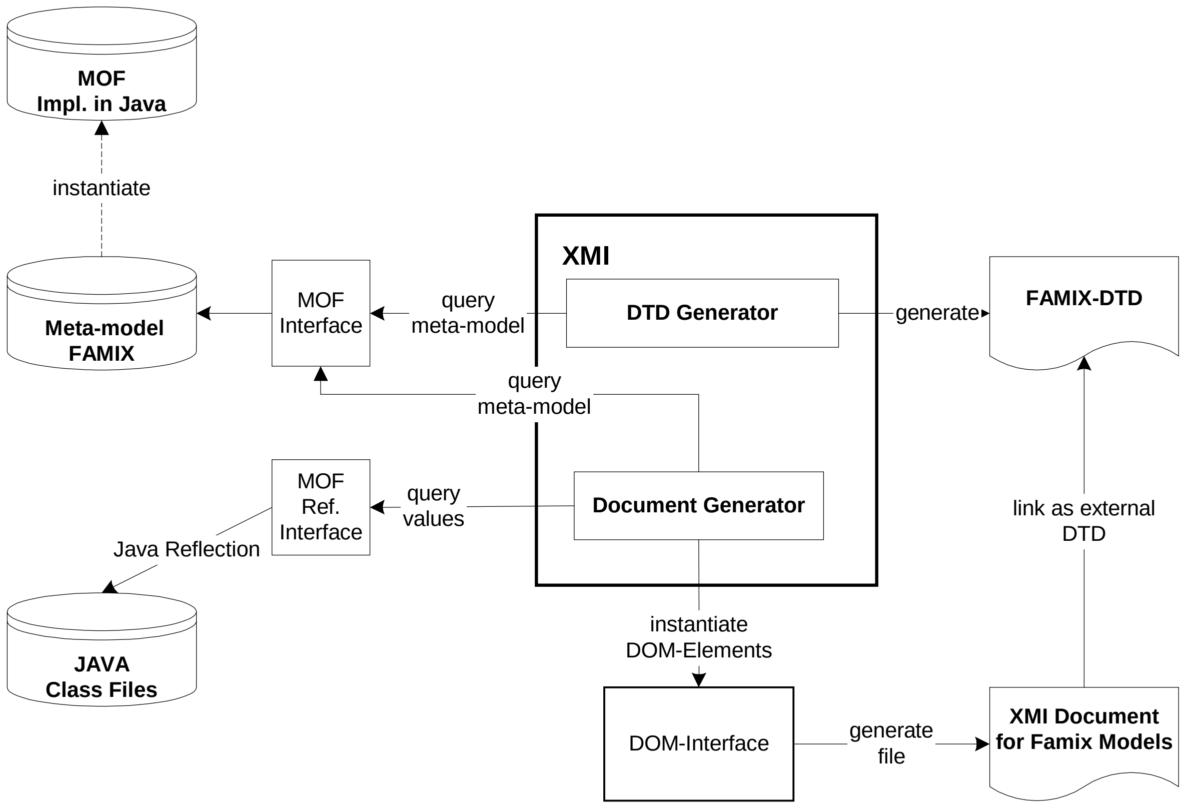


Figure 2: Overview of XMI for FAMIX

In context of XMI for FAMIX the architecture above is used to implement a tool that generates an XML-DTD for the meta-mode FAMIX and XMI documents for sourcecode models extracted from Java class files. The MOF is implemented in Java and the meta-model FAMIX is a set of instances of Java classes. The MOF Interfaces are described as a set of Java interfaces. The DTD Generator then generates a DTD for the meta-model FAMIX. The model data is extracted from Java class files using Java reflection. The Document Generator queries data values and uses them to create XML-elements via the DOM API. The previously generated DTD is linked as external DTD in the generated XML document.

1.3 Structure of this Document

FAMIX
Gives in brief terms a description of the role and the structure of the FAMIX model.
Meta-Object-Facility (MOF)
Explains the basic concepts of the MOF and its position in the OMG meta data architecture.
XML Overview
Gives an introduction to XML as far it is relevant for XMI
XMI Generation Principles
This chapter gives an overview of the basic principles used to generate XMI DTDs and documents. The concepts are explained using a simple example from FAMIX
Java Reflection
Gives background information about the structure and the usage of the Java reflective interface.
Design of XMI for FAMIX
The design of XMI for FAMIX is described using UML diagrams. The package diagram shows the dependency among the different packages. The class diagrams gives overview of the most important classes interfaces associations and inheritance hierarchies.
Implementation of XMI for FAMIX
This chapter covers various implementation aspects of XMI for FAMIX such as the instantiation of FAMIX as a MOF meta-model, the implementation of the DTD- and the Document Generator and the test program.
Usage Scenarios
This section describes some of the problems that XMI helps to solve.
Conclusion
In this last chapter the experience made throughout the project is summarized and an outlook for further work in this area is given.

1.4 Standards

The following standards are used for XMI for FAMIX

Standard	Organization	Version
MOF (Meta-Object-Facility)	OMG	1.1
XMI (XML based Meta-model Interchange)	OMG	1.0
XML	W3C	1.1
DOM	W3C	1.0
JDK	SUN	1.1.8

1.5 Conventions

`XML, Java code and class names appear using this font.`

2 Background

2.1 FAMIX

In the context of the FAMOOS project various partners developed a set of tool prototypes that supports object-oriented reengineering. The FAMOOS partners have built a number of tool prototypes for conducting various experiments within the area of object oriented reengineering. However, the source code available for case studies is written in different implementation languages (C++, Ada and to a lesser extent Java and Smalltalk). To avoid equipping all the tool prototypes with parsing technology for all of the implementation languages, a common information exchange model with language specific extensions is specified. This model has been named FAMIX, standing for FAMOOS Information Exchange Model.

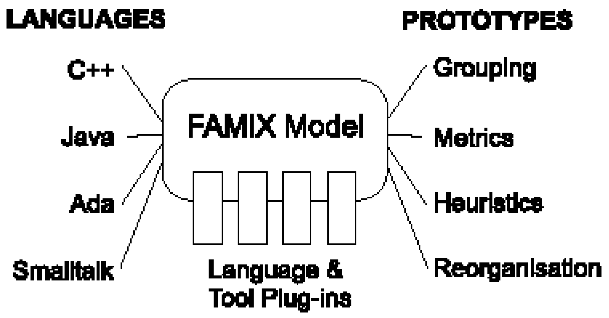


Figure 3: Conception of the FAMIX Model

2.1.1 Description of FAMIX

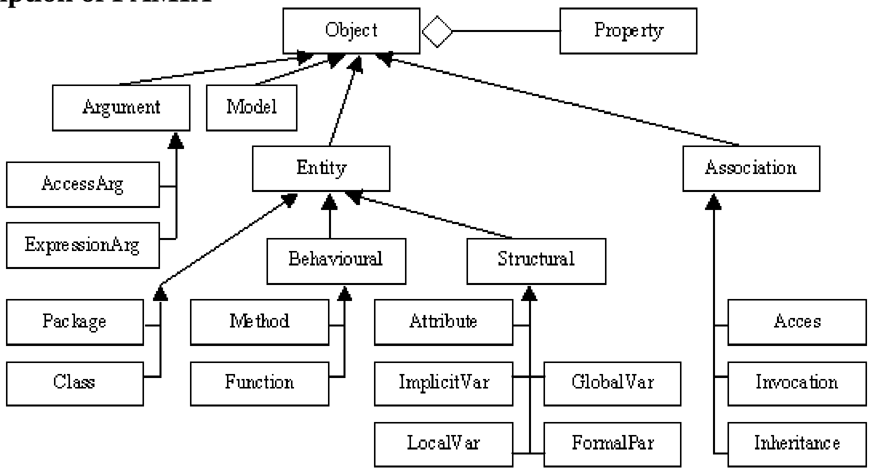


Figure 4: The Complete FAMIX Meta Model

- Object is an abstract class without a superclass. Association and Entity are both abstract classes inheriting from Object. Property is a concrete class without a superclass.
- A Model represents information concerning the particular system being modeled.

- A Package represents a named sub-unit of a source code model, for example namespaces in C++, and packages in Java. What exactly constitutes such a sub-unit is a language dependent issue. Packages and other entities can only belong to one Package. Package is a concrete class inheriting from Entity.
- A Class represents the definition of a class in source code. What exactly constitutes such a definition is a language dependent issue. Class is a concrete class inheriting from Entity.
- A BehaviouralEntity represents the definition in source code of a behavioral abstraction, i.e. an abstraction that denotes an action rather than a part of the state. Subclasses of this class represent different mechanisms for defining such an entity. A Method represents the definition in source code of an aspect of the behavior of a class. What exactly constitutes such a definition is a language dependent issue.
- A StructuralEntity represents the definition in source code of a structural entity; i.e. it denotes an aspect of the state of a system. The different kinds of structural entities mainly differ in lifetime: some have the same lifetime as the entity they belong to, e.g. an attribute and a class, some have a lifetime that is the same as the whole system, e.g. a global variable. Subclasses of this class represent different mechanisms for defining such an entity.
- An Attribute represents the definition in source code of an aspect of the state of a class. What exactly constitutes such a definition is a language dependent issue.
- An ImplicitVariable represents the definition in source code of context dependent reference to a memory location (i.e., ’this’ in C++ and Java, ’self’ and ’super’ in Smalltalk). What exactly constitutes such a definition is a language dependent issue.
- A LocalVariable represents the definition in source code of a variable defined locally to a behavioural entity. What exactly constitutes such a definition is a language dependent issue.
- A FormalParameter represents the definition in source code of a formal parameter, i.e. the declaration of what a behavioural entity expects as an argument. What exactly constitutes such a definition is a language dependent issue.
- An InheritanceDefinition represents the definition in source code of an inheritance association between two classes. One class then plays the role of the superclass, the other plays the role of the subclass. What exactly constitutes such a definition is a language dependent issue.
- An Access represents the definition in source code of a BehaviouralEntity accessing a StructuralEntity. Depending on the level of extraction, that StructuralEntity may be an attribute, a local variable, an argument, and a global variable. What exactly constitutes such a definition is a language dependent issue.
- An Invocation represents the definition in source code of a BehaviouralEntity invoking another BehaviouralEntity. What exactly constitutes such a definition is a language dependent issue. However, when the same behavioural entity is invoked more than once in a method body, then parsers should generate a separate invocation-association for each occurrence.
- An Argument represents the passing of an argument when invoking a BehaviouralEntity. What exactly constitutes such a definition is a language dependent issue. The model distinguishes between two kinds of arguments, an ExpressionArgument or an AccessArgument. The former means that some complex expression is passed, in that case the contents of the expression are not further specified. The latter means that a reference to a StructuralEntity is passed, thus involving an Access to the corresponding structural entity, hence a reference to the corresponding Access is stored within the AccessArgument.

2.1.2 Java Language Extensions

In order to handle specific aspects of the Java language, FAMIX is extended in three different ways [Tich99]

- A class `TypeCast` is added to the common exchange model to model type cast like `(MyClass)variable`.
- New attributes are added to existing classes of the basic FAMIX model. These attributes model language specific modifiers of classes (isInterface, isPublic, isFinal), attributes (isFinal, isTransient, isVolatile) and methods (isFinal, isSynchronized, isNative)
- The definitions of attributes of existing classes are modified or are made more specific in context of the language. The most important examples are the interpretation `InheritanceDefinition` and the meaning of attributes `declaredType` and `declaredClass` of `StructuralEntity.`

2.2 MOF

2.2.1 Meta-Data Architecture

The Meta Object Facility (MOF) is an OMG standard that is used for defining any kind of object-oriented meta-models. A meta-model is explained in the context of the four-layer meta-data architecture.

Meta-level	MOF terms	Examples
M3	Meta-meta-model	MOF model
M2	Meta-model	UML Meta-model, FAMIX
M1	Model	UML Models, Java classes
M0	Data	Records in a database, instances of M1 classes

Table 1 OMG Metadata Architecture

- The meta-meta-model layer is comprised of the description of the structure and semantics of meta-meta-data. In other words, it is the "language" for defining different kinds of meta-data.
- The meta-model layer is comprised of the descriptions (i.e. meta-meta-data) that define the structure and semantics of meta-data. Meta-meta-data is informally aggregated as meta-models. A meta-model can also be thought of as a "language" for describing different kinds of data.
- The model layer is comprised of the meta-data that describes information. Meta-data is informally aggregated as models.
- The M0 layer consists of the information that we wish to describe. This information is typically referred to as "data".

2.2.2 The MOF Model

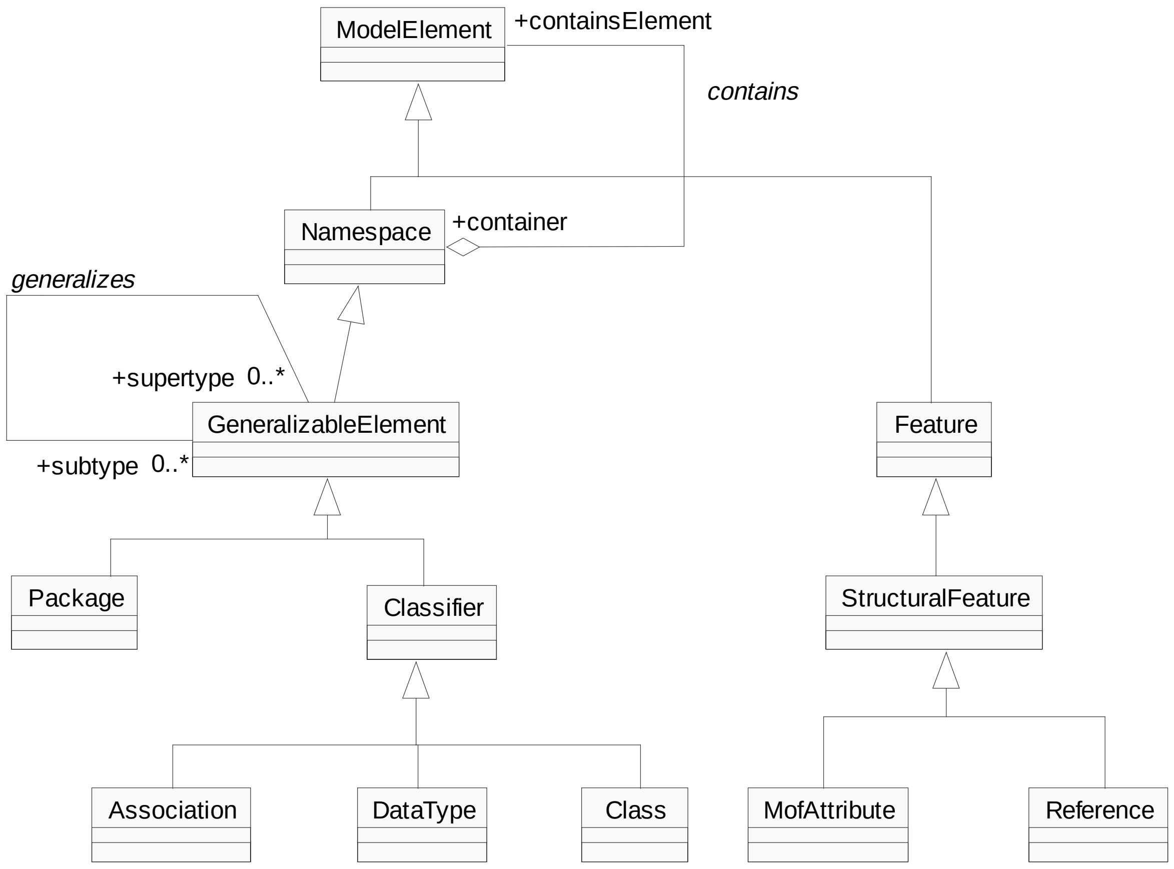


Figure 5 Overview of the Structure of the MOF model

The core structure of the MOF model is shown in the class diagram above. This diagram shows the key abstract classes in the MOF model and the main metadata constructs `Class`, `Association`, `Package`, `MofAttribute`. For a complete overview of the MOF model refer to Appendix C.

The key abstract classes in the MOF model are as follows:

- `ModelElement` - this is the common base `Class` of all M3-level classes in the MOF model. Every `ModelElement` has a "name".

- `Namespace` - this is the base class for all M3-level classes that need to act as containers in the MOF model.
- `GeneralizableElement` - this is the base class for all M3-level classes that support “generalization”; i.e. inheritance.
- `TypedElement` - this is the base class for M3-level classes such as `Attribute`, `Parameter` and `Constant` whose definition requires a type specification.
- `Classifier` - this is the base class for all M3-level classes that (notionally) define types. Examples of `Classifier` include `Class` and `DataType`.

Classes, Association and Packages are the three main metadata-modeling constructs that are provided by the MOF.

- 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 meta-model 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.

MOF provides a set of interfaces that can be used to query objects at the M2 Level. Examples of using these interfaces are a selection of all M2-Attributes of an instance of a MOF-Class (M3) or the selection of all supertypes of a class. In the MOF specification these interfaces are described as a set of IDL interfaces but they can be mapped easily to interfaces in another programming language such as Java.

Furthermore the MOF provides a reflective interface ([MOF99] 6.2 The Reflective Interfaces) that is used for XMI generation. There are two main interfaces the `RefObject` and the `RefBaseObject` interface. The `RefObject` interface is used to query attribute values of Objects at the M1 level. This interface is used during the generation of XML data where it is necessary to query a value given the name of an M2-Level attribute. The `RefBaseObject` provides an interface to access the meta Object of a class and to test whether an object is an instance of a specific meta-model class.

2.3 Overview of XML

This section provides a simple overview of XML technology.

2.3.1 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, which 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 elements content. The following example shows a simple element:

```
<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.

2.3.2 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.)

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

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

2.3.3 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:

<Class xmi.label=”c1”> </Class>

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.

2.3.4 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”.`

2.3.5 Document Object Model

The Document Object Model [DOM98] is a W3C standard and offers an API to query and manipulate XML files. With the DOM, programmers can build documents, navigate their structure, and add, modify, or delete elements and content. As an object model, the DOM identifies:

- the interfaces and objects used to represent and manipulate a document
- the semantics of these interfaces and objects - including both behavior and attributes
- the relationships and collaborations among these interfaces and objects.

Using the DOM, XML documents are represented as trees. Every XML tag is a node of the tree and the content of a tag is a subtree that has the tag-node as its subnode.

2.4 XMI Generation Principles

XMI defines a standard to map object-oriented metadata and data to an XML DTD and an XML document. Metadata is used to define a syntax for an XML document and the document itself consists of a streamed representation of object data.

The XML-based Metadata Interchange (XMI) proposal has two major components:

- The **XML DTD Production Rules** for producing XML Document Type Definitions (DTD) 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.

- 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 production rules can be applied to any layer in the four-layer metadata architecture. In our project the FAMIX-model (M2) is mapped to an XML-DTD, the model of a software-system is mapped to an XML-document.

Meta-Level	Metadata	XMI DTD	XMI documents
M3	The MOF Model	MOF DTD	
M2	UML meta-model, FAMIX	FAMIX DTD	MOF Meta-model
M1	UML model, FAMIX models		Models
M0	Instances of M1 classes		

Table 2: XMI and the OMG Metadata Architecture

The XMI DTD production rules define the mapping of a MOF Meta-model to an XML DTD, which serves as a syntax specification for the XMI document. Although the use of a DTD is optional for creating an XMI document it is advantageous to use it so that the syntax can be validated using an XML-validating parser while reading the document.
XMI Document production rules define a standard of how to generate an XML document from instances of a given MOF meta-model. The content produced by these rules must conform to the DTD generated from the meta-model.
In context of the FAMIX this means that the FAMIX (as instance of the MOF model) is mapped to an XML DTD and the data read from the reflective interface is mapped to an XML Document that conforms to the syntax specified by the FAMIX DTD.

2.4.1 DTD Generation

The DTD Generation from a MOF – meta-model follows the following basic principles

- Each meta-model class is represented in the DTD by an XML element declaration whose name is the class name. The element declaration lists the attributes of the class, references to association ends relating to the class and aggregated classes.
- Every attribute of a meta-model class is represented in the DTD by an XML element declaration whose name is the attribute name. The attributes are listed in the element tag for the class in the order they are defined. Attribute multiplicities in the meta-model are mapped to XML multiplicities.
- Each association (with or without containment) between meta-model classes is represented by two XML elements that represent the roles of the association ends. The multiplicities of the association ends are translated to the XML multiplicities for the definition of the content for an XML element.
- If a meta-model Class is a subclass of another meta-model Class every attribute in the superclass is copied in the element definition of the subclass

In order to represent data of a meta-model it is necessary to convert multiplicities that occur in the meta-model to XML multiplicites. Because this mapping can be made according to the table below, meta-model multiplicities can be enforced using an XML validating parser.

Meta-model Multiplicity	XML Multiplicity
0 or 1	?
1 exactly	,‘
0 or more	*
1 or more	+
Other	*

Table 3: Mapping of Multiplicities

2.4.2 Example of an XMI DTD

Look at the following situation from the FAMIX meta-model.

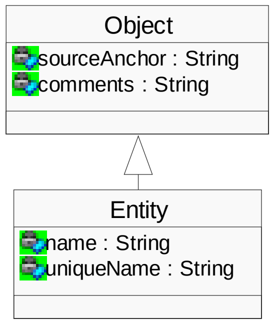


Figure 6: Fragment of the FAMIX meta-model

For this meta-model the generated XMI DTD looks as follows:

```
<!-- ______________________________________________________________ -->
<!--                                                                -->
<!-- META-MODEL CLASS:  Object                                       -->
<!-- ______________________________________________________________ -->


<!ELEMENT FAMIX.Object.sourceAnchor (#PCDATA | XMI.reference)* >

<!ELEMENT FAMIX.Object.comments (#PCDATA | XMI.reference)* >

<!ELEMENT FAMIX.Object (FAMIX.Object.sourceAnchor?,
                        FAMIX.Object.comments?, XMI.extension*)? >

<!-- ______________________________________________________________ -->
<!--                                                                -->
<!-- META-MODEL CLASS:  Entity                                       -->
<!-- ______________________________________________________________ -->


<!ELEMENT FAMIX.Entity.name (#PCDATA | XMI.reference)* >

<!ELEMENT FAMIX.Entity.uniqueName (#PCDATA | XMI.reference)* >

<!ELEMENT FAMIX.Entity (FAMIX.Object.sourceAnchor?,
                        FAMIX.Object.comments?, FAMIX.Entity.name,
                        FAMIX.Entity.uniqueName, XMI.extension*)? >
```

- For both classes Object and Entity an XML element declaration is generated. The element name is composed of the meta-model name and the classname.
- For every attribute an element declaration with the qualified name of the attribute is generated. #PCData means that in the document the attribute element contains data that needs not to be interpreted.
- Every element of a meta-model class attribute that can be a child element of the class element in the document is listed in the model group of the element declaration. For the element declaration of the class Entity every inherited attribute from class Object is in the model group. Optional attributes such as sourceAnchor have the XML - multiplicity zero or one and are listed with the ‚?‘.

A special case is the declaration of elements for boolean or enum attributes

```
<!ELEMENT FAMIX.Class.isPublic EMPTY >
<!ATTLIST FAMIX.Class.isPublic
            xmi.value ( true | false | null ) #REQUIRED
>
```

The element is declared with the keyword `EMPTY` which means that the element has no content and has no end tag. The attribute value is given by the element attribute xmi.value. The XML keyword `#REQUIRED` means that a value for `xmi.value` must be inserted. This syntax allows a parser to test whether the given enum value is valid.

2.4.3 XML Document Generation

An XMI document starts with an XMI header that contains meta data about the document. This metadata contains the name of the meta-model of which data is transferred, the XMI Version number and the name of the document generator.

```
<XMI.header>
    <XMI.documentation>
      <XMI.exporter>XMI MASTER</XMI.exporter>
      <XMI.exporterVersion>1.0</XMI.exporterVersion>
    </XMI.documentation>
    <XMI.meta-model xmi.name="FAMIX" xmi.version="2.0"/>
  </XMI.header>
```

The header followed by the XMI content enclosed in the tag `<XMI.content>`, which consists of the data that is to be transferred.
For each object that is to be transferred an XML Element according to the element declaration of the objects class is generated. The attribute xmi.id provides a unique identifier with the document for this element.

```
<FAMIX.Class xmi.id="_2">
```

Next in generation process attribute values are queried to generate XML content. The name of the attribute element is defined by the qualified name it has in the meta-model.

```
<FAMIX.Entity.name>testInterfaceA</FAMIX.Entity.name>
```

`Boolean` and `Enum` attributes have the syntax shown below

```
<FAMIX.Class.isAbstract xmi.value="true"/>
```

2.5 Java Reflection

The Java Core Reflection API provides a small, type-safe, and secure API that supports introspection about the classes and objects in the current Java Virtual Machine. If permitted by security policy, the API can be used to:

- construct new class instances and new arrays
- access and modify fields of objects and classes
- invoke methods on objects and classes
- access and modify elements of arrays

The Core Reflection API defines classes and methods, as follows:

- Three classes-`Field`, `Method`, and `Constructor`-that reflect class and interface members and constructors. These classes provide:
- Reflective information about the underlying member or constructor
- Methods of class `Class` that provide for the construction of new instances of the `Field`, `Method`, and `Constructor` classes.
- A class-`Array`-that provides methods to dynamically construct and access Java arrays.
- A utility class-`Modifier`-that helps decode Java language modifier information about classes and their members.

2.5.1 Reflection Model

The three classes `Field`, `Method`, and `Constructor` are `final`. Only the Java Virtual Machine may create instances of these classes; these objects are used to manipulate the underlying objects; that is, to:

- get reflective information about the underlying member or constructor
- get and set field values
- invoke methods on objects or classes
- create new instances of classes

The `final` uninstantiable class `Array` provides `static` methods that permit creating new arrays, and getting and setting the elements of arrays.

The classes `Field`, `Method` and `Constructor` implement the `Member` interface. The methods of `Member` are used to query a reflected member for basic identifying information. Identifying information consists of the class or interface that declared the member, the name of the member itself, and the Java language modifiers (such as `public`, `protected`, `abstract`, `synchronized`, and so on) for the member.

A `Field` object represents a reflected field. The underlying field may be a class variable (a `static` field) or an instance variable (a non-`static` field). Methods of class `Field` are used to obtain the type of the underlying field, and to get and set the underlying field's value on objects.

A `Method` object represents a reflected method. The underlying method may be an abstract method, an instance method, or a class (`static`) method.

Methods of class `Method` are used to obtain the formal parameter types, the return type, and the checked exception types of the underlying method. In addition, the `invoke` method of class `Method` is used to invoke the underlying method on target objects. Instance and abstract method invocation uses dynamic method resolution based on the target object's run-time class and the reflected method's declaring class, name, and formal parameter types. (Thus, it is permissible to invoke a reflected interface method on an object that is an instance of a class that implements the interface.) Static method invocation uses the underlying static method of the method's declaring class.

A `Constructor` object represents a reflected constructor. Methods of class `Constructor` are used to obtain the formal parameter types and the checked exception types of the underlying constructor. In addition, the `newInstance` method of class `Constructor` is used to create and initialize a new instance of the class that declares the constructor, provided the class is instantiable.

The `Array` class is an uninstantiable class that exports class methods to create Java arrays with primitive or class component types. Methods of class `Array` are also used to get and set array component values.

The `Modifier` class is an uninstantiable class that exports class methods to decode Java language modifiers for classes and members. The language modifiers are encoded in an integer, and use the encoding constants defined by The Java Virtual Machine Specification.

3 Design of XMI for FAMIX

3.1 Package Design

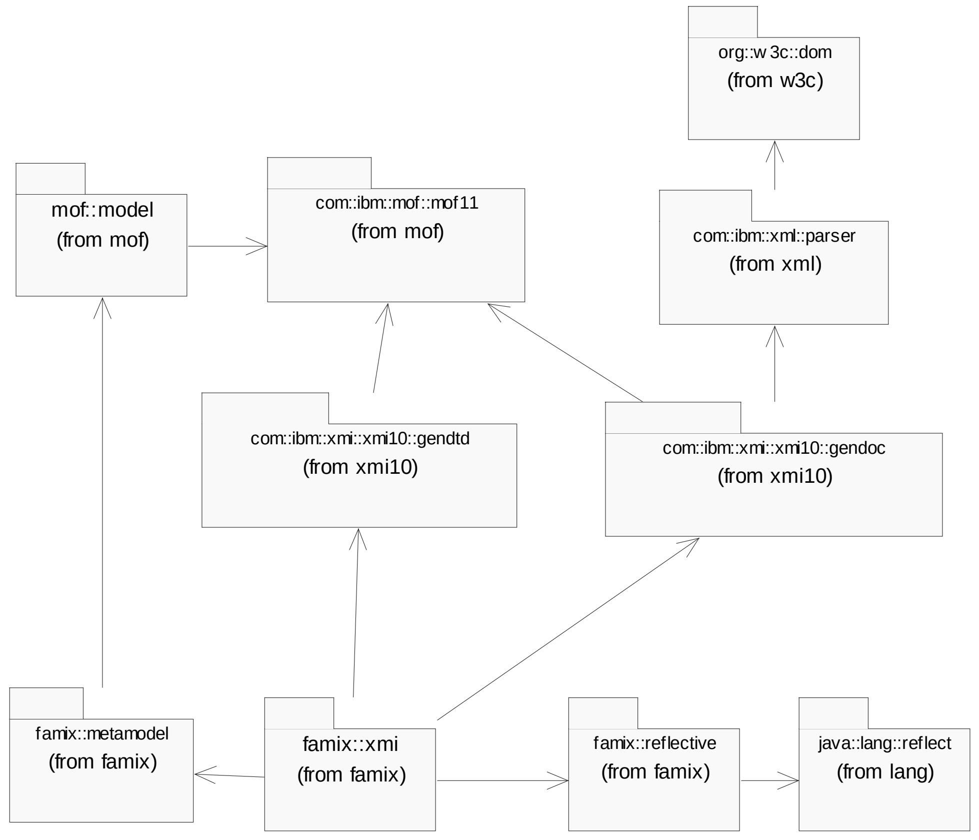


Figure 7: Package Diagram of XMI for FAMIX

The packages `com::ibm::xmi::xmi10::gendtd`, `com::ibm::xmi::xmi10::gendoc` and `com::ibm::mof::mof11` are part of the XMI Application Framework from IBM Alphaworks. The package `com::ibm::mof::mof11` contains the MOF interfaces implemented as Java interfaces. This package contains also the interface `RefObject` and `RefBaseObject` from the MOF reflective interfaces as far as they are needed for the document generation. The package `com::ibm::xmi::xmi10::gendtd` contains a generic DTD Generator that is capable of generating a DTD for any MOF meta-model. It is solely dependent on the MOF interfaces that means from the package `com::ibm::mof::mof11`. A generic document generator that reads attribute values through the MOF reflective interface `RefObject` is implemented in the package `com::ibm::xmi::xmi10::gendoc`. This document generator is dependent on the package `com::ibm::xml::parser` because it instantiates objects that implement the DOM interface `Element` to create nodes in the XML tree. The content of `mof::model` is an implementation of the MOF interfaces in `com::ibm::mof::mof11`. The meta-model FAMIX is instantiated in `famix::meta-model` using the MOF implementation in `mof::model`. Finally, the package `famix::xmi` contains the implementation of the FAMIX specific parts that are neccessary for XMI. This includes reading from the reflective interface and glueing the components together. Furthermore a test program is included in the subpackage `famix::xmi::doctest` that helps to test the correctness of the system in case of a change. The functionality that reads data from the Java reflective interface is implemented in `famix::reflective`.

3.2 DTD Generator

The DTD generator maps a MOF meta-model at the M2 level in the four-layer OMG meta-model architecture to a XML DTD. The DTD Generator is implemented in the classes shown in the diagram below. The class

XMIDTDBuilder and the interfaces `Package, MOFAttribute` and `Class` originate from the XMI framework.

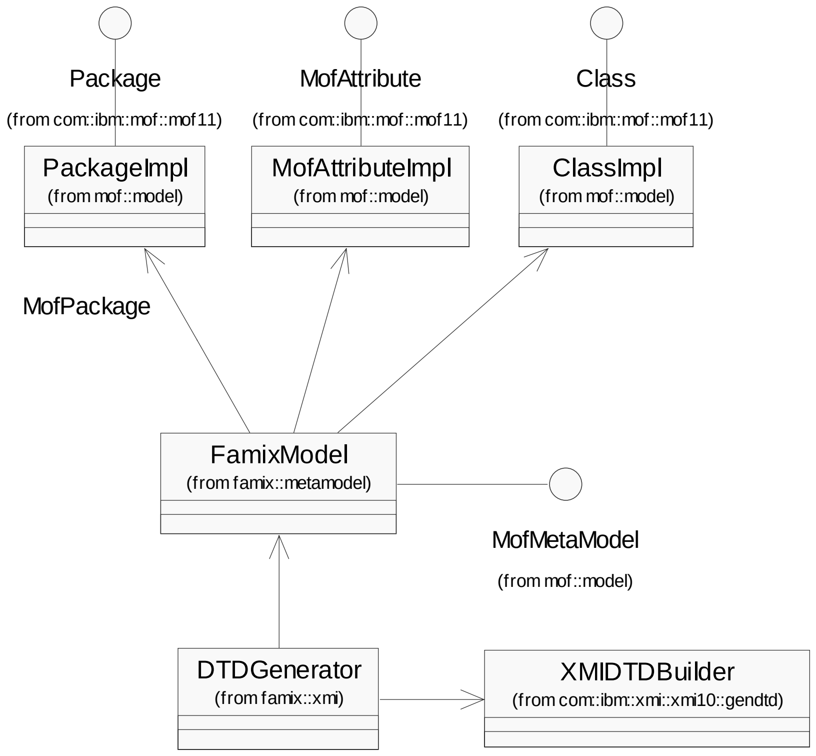


Figure 8: Design of the DTD Generator

- `Package`, `MofAttribute` and `Class` are MOF interfaces according to the definition in [MOF99] "MOF Model Classes".
- `PackageImpl`, `MofAttributeImpl` and `ClassImpl` are implementation classes of the corresponding interfaces.
- The interface `MOFMeta-model` specifies an interface to query the top level MOF package of a MOF meta-model.
- The class `FamixModel` instantiates the meta-model FAMIX by instantiating the classes `ClassImpl` and `MofAttributeImpl`. The MOF requires that every meta-model class is contained within a MOF package. For this reason a MOF package with the name "Famix" is created. `FamixModel` implements the interface `MofMeta-model` so that the top level MOF packages can be returned and passed to `DTDBuilder`.
- `DTDGenerator` has the responsibility to instantiate the meta-model and a `DTDBuilder`. Because DTD generation is generic for every MOF meta-model the `DTDGenerator` is capable of loading different meta-models. This is done by specifiying the name of a class, which instantiates a MOF meta-model, as a commandline parameter for the call to the DTD generator , so that the class can be dynamically loaded by the class loader. Every class that instantiates a meta-model from the MOF must implement the method `getTopLevelPackage()` which returns the top level MOF package.
- `XMIDTDBuilder` is a framework class that implements the DTD generation rules according to [XMI99] XML DTD Production.

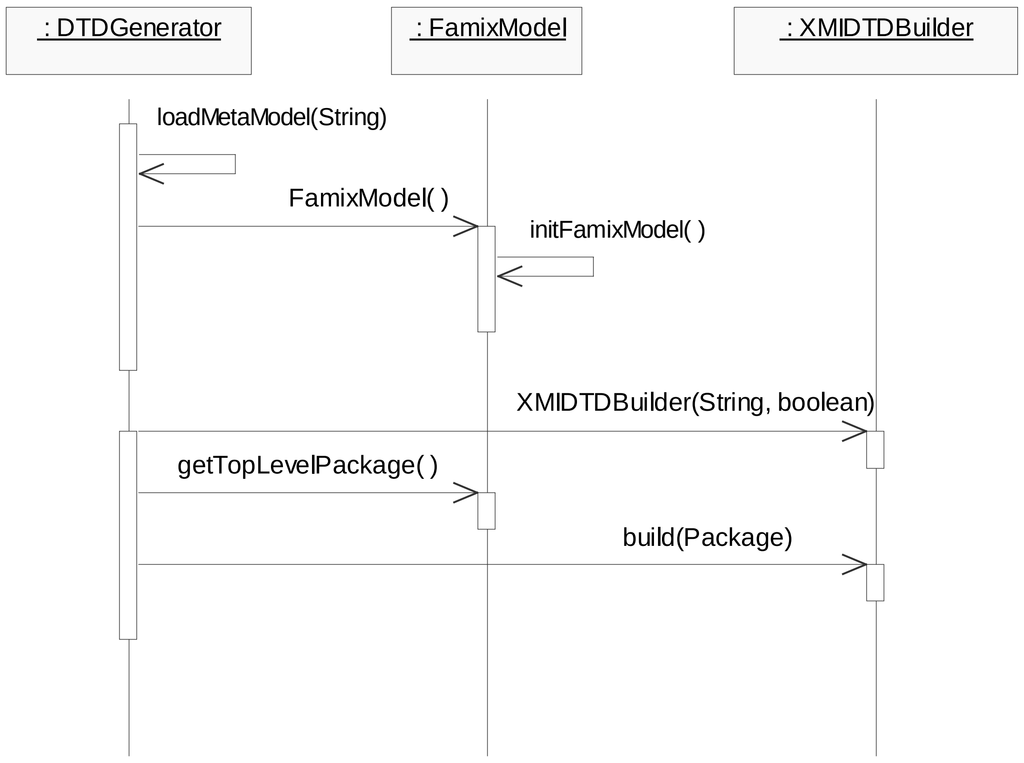


Figure 9: Generating a DTD

The DTD Generator dynamically loads a class through the class loader, specified by the first argument to the method `loadMetaModel()`. For FAMIX this is the class `FamixModel`. In its constructor the method `initFamixModel()`, where the meta-model is instantiated from the MOF, is called. Next `XMIDTDBuilder` is instantiated. Using the method `getTopLevelPackage()`, the top level MOF package is queried and passed as argument to the build method of `XMIDTDBuilder`. `XMIDTDBuilder` then generates a file containing the DTD for the loaded meta-model.

3.3 XMI Document Generator

The XMI Document Generator uses the class `JavaReflectiveReader` to extract source code information from a Java via the Java reflective interface. This data is transformed according to the FAMIX naming convention using the class `FamixNamingConvention`. With help of the framework class XMIDocBuilder an XMI document is generated.

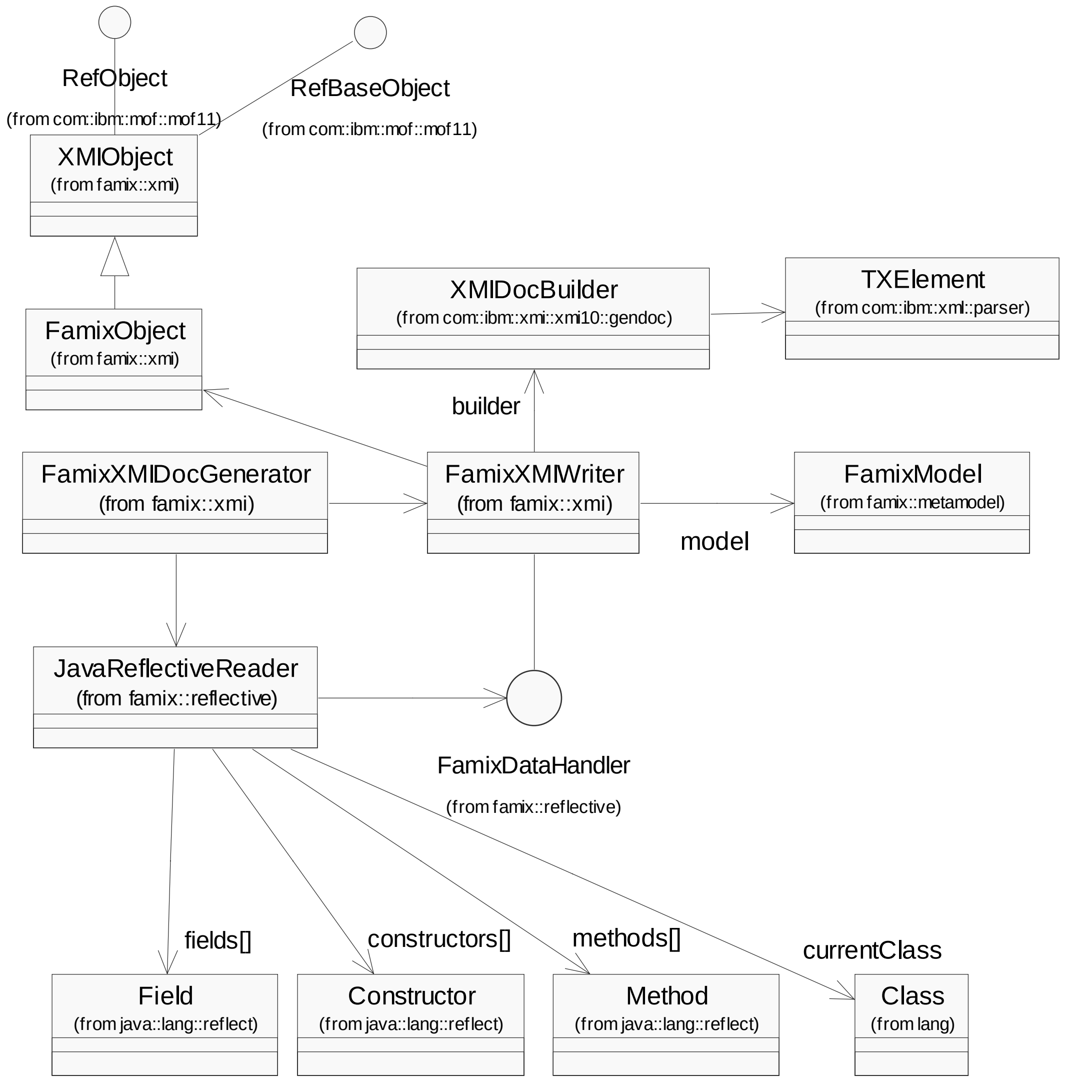


Figure 10: Design of the XMI Document Generator

- The class `JavaReflectiveReader` extracts data from Java class files using the Java reflective API and transforms it to the FAMIX naming convention. Currently level 2 extraction for FAMIX is supported. In addition to the standard FAMIX model version 2.0, JavaReflectiveReader supports the Java language extensions [Tich99] for FAMIX. The data is then passed to an object of a class that implements the interface `FamixDataHandler`. The method `getClassesOfPackage()` queries the file system for class files that are contained in the specified package, because Java has no built in package reflection.
- In order to enforce the correct naming convention a class `FamixNamingConvention` was created. Its main purpose is to give the correct naming format for unique names of entities and for method signatures.
- The interface `FamixDataHandler` implements handler functions for each FAMIX class that can be extracted through the reflective interface. Using this interface `JavaReflectiveReader` is independent of a specific handler class, and therefore more reusable.
- `Field`, `Constructor`, `Method` and `Class` are classes from the package `java.lang.reflect` and `java.lang` that implement the access to the reflective interface.
- `FamixXMIWriter` implements `FamixDataHandler` and instantiates a `FamixObject` for the data received from the reflective reader. `FamixXMIWriter` then passes the instance of `FamixObject` to `XMIDocBuilder`.
- `XMIObject` is a class that encapsulates a `Hashtable`. This `Hashtable` is used to associate an attribute name and its value, so that it can be queried using `Reflective::RefObject::value(attribute_name). RefObject::value` returns an object that encodes values according to [MOF99] „The alternate value encoding pattern“. Basically this means that non-mandatory attribute and attributes with a meta-model multiplicity greater than one are

encoded as lists. Furthermore `XMIObject` links an object to its meta object by referencing the corresponding instance of the MOF class `Class`. The meta object can be queried using the method `metaObject()` from the MOF interface `Reflective::RefBaseObject`.

- `FamixObject` inherits from `XMIObject` and implements a method `getStringValue(String key)`. This method is used in the test program, where the test of equality of two object is based on string comparison.
- `FamixXMIDocGenerator` contains the main method and instantiates `JavaReflectiveReader` and `FamixXMIWriter`. `FamixXMIWriter` is passed as `FamixDataHandler` to the instance of `JavaReflectiveReader`. The generation process is then started by calling the `transferInfo()` method of `JavaReflectiveReader`.
- `XMIDocBuilder` is a framework class that implements the document generation rules according to [XMI99] „XML generation principles“. `XMIDocBuilder` instantiates `TXElement` class, that implements the DOM interface `Element`, in order to create the XML tree.

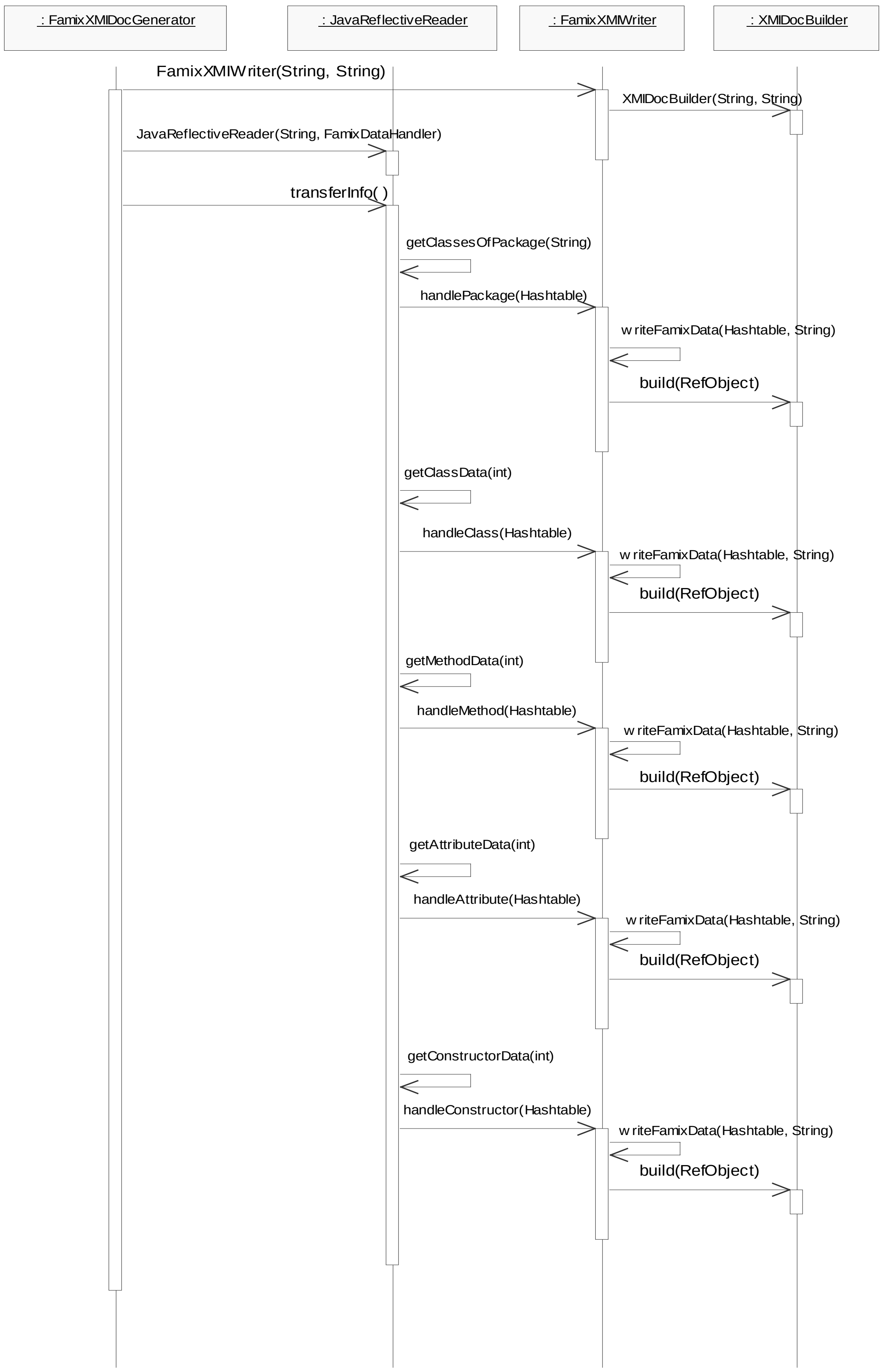

: FamixXMIDocGenerator
: JavaReflectiveReader
: FamixXMIWriter
: XMIDocBuilder
FamixXMIWriter(String, String)
XMIDocBuilder(String, String)
JavaReflectiveReader(String, FamixDataHandler)
transferInfo( )
getClassesOfPackage(String)
handlePackage(Hashtable)
writeFamixData(Hashtable, String)
build(RefObject)
getClassData(int)
handleClass(Hashtable)
writeFamixData(Hashtable, String)
build(RefObject)
getMethodData(int)
handleMethod(Hashtable)
writeFamixData(Hashtable, String)
build(RefObject)
getAttributeData(int)
handleAttribute(Hashtable)
writeFamixData(Hashtable, String)
build(RefObject)
getConstructorData(int)
handleConstructor(Hashtable)
writeFamixData(Hashtable, String)
build(RefObject)

Figure 11: Generating an XMI Document for FAMIX

As we see in the sequence chart `FamixXMIDocGenerator` instantiates first a `FamixXMIWriter` wich in turn instantiates a `XMIDocBuilder`. Then a `JavaReflectiveReader` is instantiated passing a name of a Java package and a `FamixDataHandler` in the constructor. The previously created instance of `FamixXMIWriter` is registered as `FamixDataHandler` within the instance of `JavaReflectiveReader`. The generation process is started calling the method `transferInfo()`. Then iteratively data is extracted from the Java reflective interface and passed to the corresponding handler function in `FamixXMIWriter`. `FamixXMIWriter` instantiates a `FamixObject` an passes it as argument to the build method of `XMIDocBuilder`.

3.4 Test Program

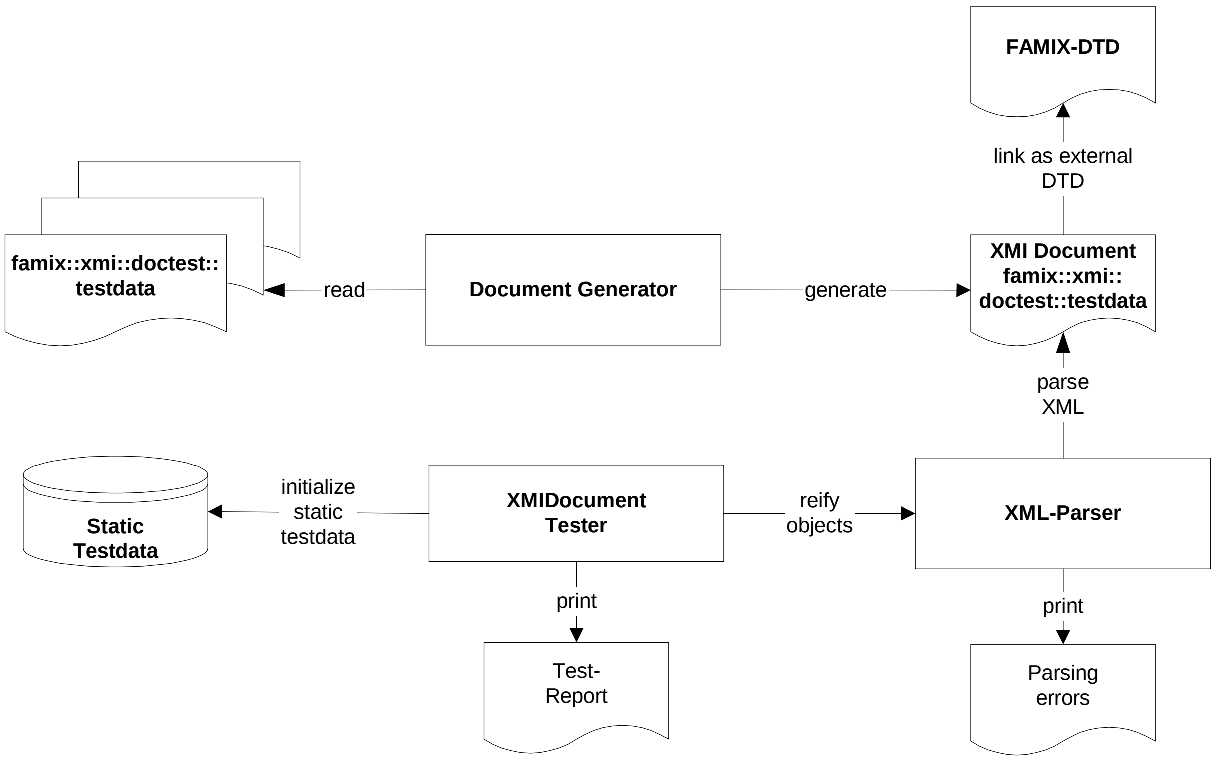


Figure 12: Concept of the Test Program

The purpose of the test program is to implement a test case for the DTD generator and the document generator and then test if the output produced is correct. It uses a fixed set of test data contained in the package famix::xmi::doctest::testdata that should cover every attribute in FAMIX as far as the reflective reader supports it. Based on the test package an XMI document is generated. The content of this document is then compared to hardcoded test values that represent the expected attribute values of the test data from the package. In order to pass the test attribute values read from the document must match those that are hardcoded.
Other aspects of correctness such as the correct syntax according to the DTD and the correct XML-tag syntax are verified using a validating XML-parser. This parser is also used to parse the file so that object can be reified from the document using the DOM. The class XMIDocumentTest program reifies every object from the XMI document and compares attribute values to the ones that are hardcoded. Errors are reported to the standard output so that their source can be identified.

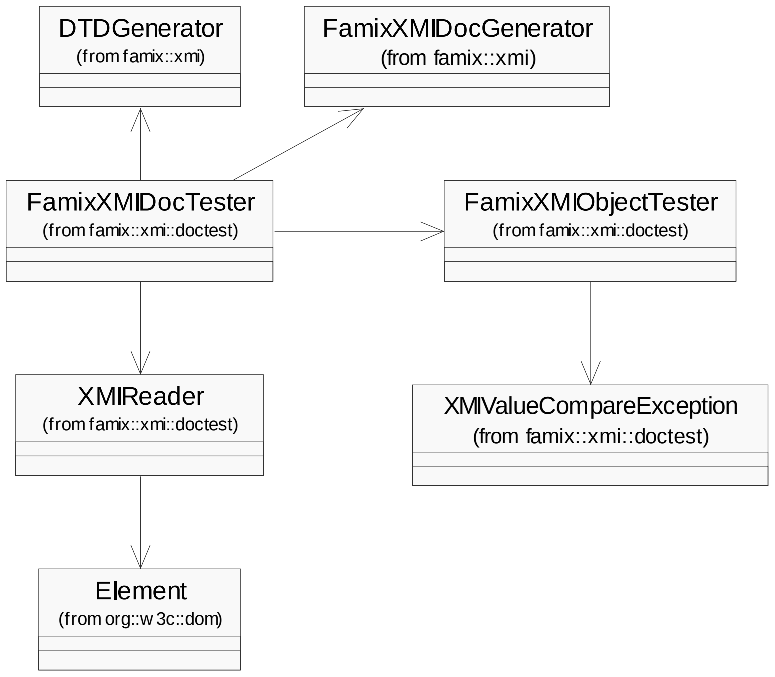


Figure 13: Design of the Test Program

- `FamixXMIDocTest program` instantiates a `DTDGenerator` and a `FamixXMIDocGenerator` in order to generate a DTD and a document based on the testdata.
- The class `XMIReader` is used to reify objects from document. The reification is done by an inverse application of the generation rules for the document. `XMIReader` uses the `Element` interface of the DOM to extract data from the document. The reified object can be retrieved via the method `getObject()`. This class implements a generic `XMIReader` and can therefore be reused.
- Each reified object is then tested against the hardcoded data using the functionality of class `FamixXMIObjectTest`.

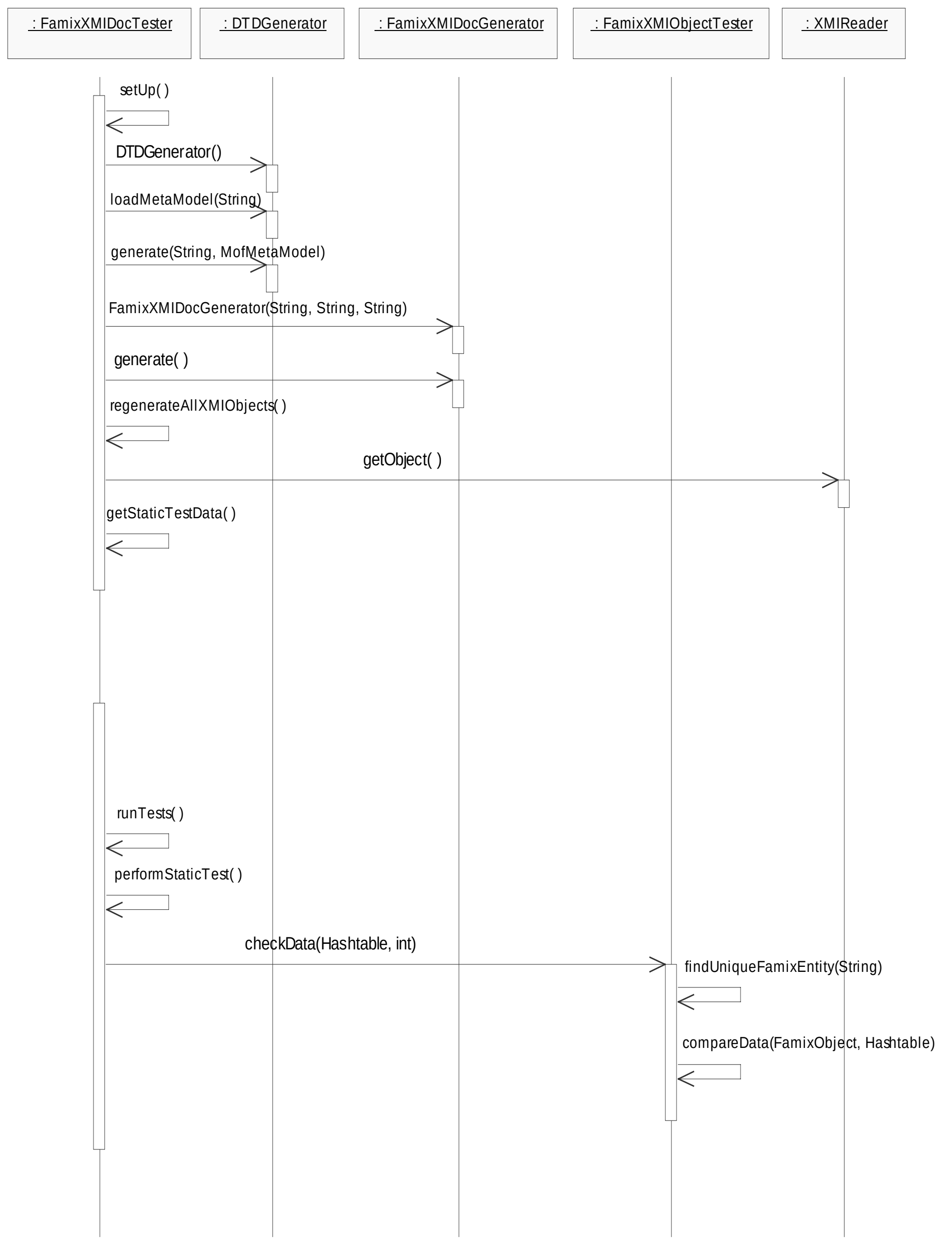


Figure 14: Running a Test

The test consists of the three phases `setUp()`, `runTest()` and `tearDown()`. During the setup phase the static data is initialized, a DTD and the document are generated. Next the test is executed in runTest. Finally everything is cleaned up in `tearDown()`.
First DTD and the document are generated equivalent to the description in the previous chapters. Then objects are reified in `regenerateXMIObjects()` and stored in a list. The static data is initialized in `getStaticTestData()`. Then `checkData()` compares two objects that should contain the same values one of it is a reified object the other one contains the hardcoded values. Attribute values are compared in `compareData().`

4 Implementation of XMI for FAMIX

4.1 Instantiation of FAMIX as MOF meta-model

The following section gives an overview of how the instantiation of FAMIX as a MOF meta-model (see Figure 8: Design of the DTD Generator) is realized at the source code level. This information can be used to generate the source code for a MOF meta-model instantiation from existing datasources like a file from a modeling tool. The MOF interfaces are implemented as Java interfaces according to the MOF IDL specification and are placed in the package `com::ibm::mof::mof11`. The corresponding implementation classes are placed in the package `mof::model` and their names end with the character sequence „Impl" such as „ModelElementImpl".

4.1.1 Class

Creating a class at the M2 level means creating an instance of the MOF class Class (see Figure 5 Overview of the Structure of the MOF model) and specifiying the attribute values according to the MOF specification. The MOF class `Class` is instantiated by calling the static method `create_class` which returns a new object of type `ClassImpl`.

Syntax
```
public static ClassImpl create_class(String name, String annotation,
                              boolean is_root, boolean is_leaf,
                               boolean is_abstract,
                               boolean is_singleton)
```

Parameter	Description
Name	Name of the meta-model class
Annotation	Not used
is_root	True if it is the top level class in the inheritance hierachy an can have no superclasses, false otherwise
is_leaf	True if the class cannot be subclassed
is_abstract	True if the class is abstract
is_singleton	True if only a single instance can be created (at the M1 lev

Table 4 MOF Class Constructor Parameters

Example
```
ClassImpl Model = ClassImpl.create_class("Model", "", true, false,  false,
false);
```

4.1.2 Inheritance

In order to define inheritance at the M2 level the method `add_supertypes` of the MOF class `GeneralizableElement(`Figure 5 Overview of the Structure of the MOF model`)` is used. This method is called for the object that represents the subclass and takes a superclass as parameter

Syntax
```
public void add_supertypes(GeneralizableElement new_element)
```

Parameter	Description
New_element	An object of type ClassImpl

Table 5 GeneralizableElement add_supertypes

Example
```
Class.add_supertypes(Entity);
```

4.1.3 Attributes

An attribute of a FAMIX class is described as an instance of the class `MOFAttribute`. For FAMIX, three different datatypes are used: `MOF.String`, `MOF.Boolean` and `MOF.Integer`. `MOF.String` is used for attributes with datatype `Name` or `Qualifier`, `MOF.Boolean` is used for boolean attributes and `MOF.Integer` is for attributes of type `Position`.
In a MOF meta-model every attribute has a multiplicity which is described as an instance of class `MOF.Multiplicity`. For FAMIX every optional attribute has multiplicity 0..1 (or 0..n for multivalued attributes) every mandatory attribute has multiplicity 1 (or 1..n for multivalued attributes).

Every attribute has a scope, which describes weather it is a class level attribute or an instance level attribute on the M2 level. This is achieved by instantiating the classes `ScopeTypeInstanceLevel` and `ScopeTypeClassifierLevel` and setting the scope attribute of `MofAttribute` by passing one of these instances as an argument in the constructor.

Syntax
```
public MofAttributeImpl(String name, String annotation, ScopeType scope,
                      MultiplicityType multiplicity, Classifier type,
                      boolean is_changeable, boolean is_derived)
```

Parameter	Description
name	Name of the meta-model attribute
annotation	Not used
scope	Object of Type ScopeTypeClassifierLevel or ScopeTypeInstanceLevel
multiplicity	Multiplictiy of the attribute
is_changeable	true if value can be set at the M1 level
is_derived	Always false for FAMIX

Table 6 MofAttributeImpl Constructor Parameters

Example
```
MofAttributeImpl attr = new MofAttributeImpl(name, "", scopeInstance,
                              multiplicity_optional,
                              MOFType.STRING, true, false);
```

4.1.4 Multiplicities

`MultiplicityType` is a type that is used to specify the multiplicity properties of an Attribute, Parameter, Reference or AssociationEnd.

Syntax
```
public MultiplicityTypeImpl(long lower, long upper, boolean isOrdered,
                        boolean isUnique)
```

Parameter	Description
lower	lower multiplicity bound
upper	upper multiyplicity bound
isOrdered	values of multivalued attributes are ordered, always false for FAMIX
isUnique	value of multivalued attributes are unique, always false for FAMIX

Table 7 MulitplicityImpl Constructor Parameters

Example
```
multiplicity_one = new MultiplicityTypeImpl(1, 1, false, false)
```

4.1.5 Scope

Attributes can have two different scopes, instance or class scope. In order to assign a specific scope to an instance of `MofAttribute` one passes either an instance of `ScopeTypeInstanceLevelImpl` or `ScopeTypeClassifierLevel` as an argument in the constructor of `MofAttribute`.

Syntax
```
public ScopeTypeInstanceLevelImpl()
```
or
```
public ScopeTypeClassifierLevel()
```

Example
```
scopeInstance = new ScopeTypeInstanceLevelImpl()
```

4.1.6 Assignment of Attributes to Classes

In the MOF model `Class` is a subtype of `Namespace`. `Namespace` serves as container because instances of `Namespace` can contain instances `ModelElements`. `ModelElement` classifies the elementary, atomic constructs of models. `ModelElement` is the root `Class` within the MOF model.

Syntax

```
public void add_contents(ModelElement new_element)
```

Example

Adding an attribute with name „`hasClassScope`“ to class method.

```
Method.add_contents(new MofAttributeImpl("hasClassScope", "",
                                        scopeInstance,
                                        multiplicity_optional,
                                        MOFType.BOOLEAN, true,
                                        false));
```

4.2 JavaReflectiveReader

The class `JavaReflectiveReader` extracts data from the Java reflective API, transforms it to the FAMIX naming convention and passes a hashtable to the handler interface. In the following section we show what FAMIX information is supported using the Java reflective interface. According to the „Levels of extraction“ level 2 is supported. This means that FAMIX entities of type Class, InheritanceDefinition, Method and Attribute can be extracted.
In order to form the correct naming convention a class `FamixNamingConvention` was created. Its main purpose is to give the correct naming format for unique names of entities and for method signatures. In addition to the standard FAMIX attributes, the reflective reader supports the Java language extensions [Tich99] for FAMIX.

Reading from the reflective interface means querying attributes of classes in the package `java.lang.reflect` and the class Class in the package `java.lang`.

The following classes from the package `java.lang.reflect` are used

- `Constructor`, for extraction name and signature of constructors of a class
- `Field`, to extract name and type of attributes
- `Method`, to extract name, signature and return type of methods
- `Modifier`, used to determine the language modifier (i.e. public, abstract,..) of a class, method or field
- `Class` from package `java.lang` to extract name, super types and implemented interfaces.

4.2.1 Reading Classes

The class `JavaReflectiveReader` takes package path as an argument in its constructor and then queries the filesystem for existing classfiles. For each found classfile, the class loader reifies a class object.

Source for FAMIX attribute values:

FAMIX Attribute	Reflective Interface Source	Description
name	currentClass.getName()	Is transformed to the FAMIX naming convention by the method getSimpleNameFromQualified
isAbstract	Modifier.isAbstract(modifier)	The value for modifier is obtained by calling the method getModifier for the current Class object
uniqueName	currentClass.getName()	Is transformed to the FAMIX naming convention by calling the method qualifiedClassName
isInterface	Modifier.isInterface(modifier)	
isPublic	Modifier.isPublic(modifier)	
isFinal	Modifier.isFinal(modifier)	

Table 8 Attributes for Class

4.2.2 Reading Methods

For each class the list of methods is obtained by calling `getDeclaredMethods` of the corresponding object. Constructors are retrieved by calling `getDeclaredConstructors` and they are treated as instances of FAMIX methods for the document generation.

Source for FAMIX attribute values:

FAMIX Attribute	Reflective Interface Source	Description
name	method.getName()	

uniqueName	method.getName()
	method.getParameterTypes()
		Unique name is build by using the method methodNameWithSignature of class FamixNamingConvention

accessControlQualifier	Modifier.isPublic(modifier)
	Modifier.isPrivate(modifier)
	Modifier.isProtected(modifier)
		The text for this is evaluated in the method getAccessControlQualifier(modifier) of the JavaReflectiveReader

signature	method.getParameterTypes()	
declaredReturnType	method.getReturnType()	
declaredReturnClass	method.getReturnType()	empty if type is primitive or if it is a constructor
belongsToClass	currentClass.getName()	
hasClassScope	Modifier.isStatic(modifier)	always false for constructor
isAbstract	Modifier.isAbstract(modifier)	always false for constructor
isConstructor	Check if type is Method or Constructor	false for method true for constructor
isNative	Modifier.isNative(modifier)	always false for constructor
isSynchronized	Modifier.isSynchronized(modifier)	always false for constructor
isFinal	Modifier.isFinal(modifier)	always false for constructor
isPureAccessor	not supported	value null because it is unknown

Table 9 Attributes for Methods

4.2.3 Reading Attributes

For each class the list of attributes is obtained by calling `getDeclaredFields`.

Source for FAMIX attribute values:

FAMIX Attribute	Reflective Interface Source	Description
name	field.getName()	

uniqueName	currentClass.getName()
	field.getName()
		according to FAMIX naming convention

accessControlQualifier	Modifier.isPublic(modifier)
	Modifier.isPrivate(modifier)
	Modifier.isProtected(modifier)
		The text for this is evaluated in the method getAccessControlQualifier(modifier) of the JavaReflectiveReader

declaredType	type.getName()	
declaredClass	type.getName()	empty if type is primitive
belongsToClass	currentClass.getName()	
hasClassScope	Modifier.isStatic(modifier)	
isFinal	Modifier.isFinal(modifier)	
isTransient	Modifier.isTransient(modifier)	
isVolatile	Modifier.isVolatile(modifier)	

Table 10 Attributes for Attribute

4.2.4 Reading Inheritance

In Java there are three different situations where an inheritance is generated

- a subclass extends a superclass
- an interface extends another interface
- a class implements an interface

Implemented interfaces of a class are queried by the `getInterfaces()` method, the superclass is evaluated by calling `getSuperClass()`.

Source for FAMIX attribute values:

FAMIX Attribute	Reflective Interface Source	Description
subclass	currentClass.getName()	unique class name according to FAMIX naming convention
superclass	superClass.getName()	unique class name according to FAMIX naming convention
index	not supported	The index is always "null" as Java has single inheritance and therefore name collisions cannot appear.

Table 11 Attributes for Inheritance

4.3 Class DTDBuilder

The class DTDBuilder creates an XML DTD according to the rules described in the chapter XMI generation principles. It takes a MOF package and then generates the DTD from its content. The rules that are important for FAMIX are given in pseudo-code notation below.

To generate a DTD:

```
Generate the FixedContent XMI definitions.

For each Package in the Meta-model not contained by another Package Do
      Generate a PackageDTD(#2).
End
```

Package refers here to a MOF package. For FAMIX we have one MOF package with name Famix.

To Generate a PackageDTD:

```
For Each Class of the Package Do
      Generate the ClassDTD (#3) for the Class
End
```

To Generate a Class DTD:

```
For Each Attribute of the Class Do
      If isDerived is false Then
            If scope is instanceLevel then
                  Generate the AttributeElementDef (#4) for the Attribute
            End
      End
End
Generate the ClassElementDef (#6) for the Class
```

To Generate an AttributeElementDef:

```
Set AttribName := the qualified name of the Attribute.
If the type reference refers to a DataType Then
If DataType.typeCode is tk_Boolean or tk_enum Then
      Set AttribContents := ’EMPTY’
      Set AttribAttList := ’value (’ + the enumerated values, separated by
"|" +’)’ + ’#REQUIRED"
else
      Set AttribContents := ’(#PCDATA | XMI.reference)*
End
Generate the !ELEMENT and !ATTLIST definitions using AttribName,
AttribContents, and
AttribAttList
```

To Generate a ClassElementDef:

```
Set ClassName := the qualified name of the Class
Set comps2 := GetContainedClasses(this Class, ‘’)
Set ClassContents to match the pattern:
atts, refs, comps1, comps2
Remove dangling commas due to empty terms from ClassContents
Set ClassContents := ‘(‘ + ClassContents + ‘)’ + ’?’
Set ClassAttlistItems := ’%XMI.element.att; %XMI.link.att;’
Generate the !ELEMENT and !ATTLIST definitions using ClassName,
ClassContents
```
.

4.4 Class FamixObject

The class `FamixObject` is a subclass of XMIObject which encapsulates XMI data so that the object can be used by the class `XMIDocBuilder`. Every class that is used to transfer objects via the `XMIDocBuilder` must satisfy the following requirements

- It must implement the method `refMetaObject` from the MOF reflective interfaces. This method returns the meta object as instance of MOF class `Class`.
- It must implement the method `refValue` from the MOF reflective interface `RefObject`. This method returns the value from a given attribute name.

The class `XMIObject` uses a hashtable to associate a name and a value. The instance variable `metaObject` is set in the constructor and can be accessed through the method `metaObject`. Values can be queried through the method `getValue`. The internal hashtable of `XMIObject` is initialized in the constructor whith hashtable contains data return from `JavaReflectiveReader`. Values in `XMIObject` are encoded using „Standard value encoding patterns“ ([MOF99] 6.2.1 The Standard Value Encoding Pattern) of the MOF reflective interface. This means that every optional attribute (this means multiplicity 0.1) is encoded as a list. This rule is implemented by creating an array for every optional attribute with length 0 if the value is null and with length 1 otherwise.

4.5 Class XMIDocBuilder

Objects are written to an XML document by using the class XMIDocBuilder. It implements a method build which takes an instance of `FamixObject` as its single parameter. By accessing the meta object the name of the meta-model class and the names of attributes are evaluated. For each attribute name, its value is queried using the accessors of `XMIObject`. Class name and attribute values are then mapped to objects of class Element from the DOM, so that finally the XML document can be generated. The following production rules given as syntax in EBNF ([XMI98] 9.4 Production Rules) are applied in order to produce XML content of an instance of `XMIObject`:

Rule 9.4.3.1:
```
ObjectAsElement ::= < class-name xmi.id=" IdOfObject ">ObjectContents </
class-name >
```

An object is represented as an element by producing an element start tag, then the object (and any objects it contains) as the contents of the element, followed by the element end tag. The operation produces the name of the element from the Class instance in the meta-model defining the class of this object. The fully qualified name, using a dot notation, is used to avoid any ambiguity in naming the Class. The element start tag also includes an identifier of the object, which is required to be a locally-unique identifier. The ObjectContents operation provides the contents of the element.

Example
```
<FAMIX.Class xmi.id="_7">
</FAMIX.Class>
```

Rule 9.4.3.2
```
ObjectContents ::= AttributeAsElement*
```

The ObjectContents operation produces XML to represent the contents of an object –its state (attributes and references. The attribute values of the object are included in the contents of this element. From the object’s class, all the Attributes are obtained, including inherited attributes. Among those, the non-derived, instance-level attributes are selected. Over that sequence, string representations of the values are produced, using the AttributeAsElement operation. The value operation, from the MOF’s RefObject interface, provides the attribute value or values.

Example
```
<FAMIX.Class xmi.id="_7">
      <FAMIX.Entity.name>XMITestClass</FAMIX.Entity.name>
      <FAMIX.Entity.uniqueName>xmi::doctest::testdata::XMITestClass</FAMIX.
Entity.uniqueName>
      <FAMIX.Class.isAbstract xmi.value="true"/>
      ....
</FAMIX.Class>
```

Rule 9.4.4.1
```
AttributeAsElement ::= (SvAttributeContents | MvAttributeContents)
```
Each object attribute value is represented in XML in enclosing start and end tags which identify the attribute. If the attribute is multi-valued (holding a more than one instance of an object or datatype) then all those values may be represented within the contents of the single element representing the attribute. Because of the differences between single- and multi-valued references, they are handled in separate operations. If the Attribute’s multiplicity has an upper bound greater than one, including unbounded, or a lower bound of zero, the attribute value or values are returned in the MOF as a possibly-empty collection.

Rule 9.4.4.3
```
SvAttributeContents ::= (<attribute-name> EmbeddedObject    </attribute-
name>) |EnumAttribute | (<attribute-name> ObjRefOrDataValue </attribute-
name>)
```
The contents of a single valued attribute is either an object or a data value. If it is a datavalue of type boolean, or of an enum type, special production rules are used. Enum and boolean values are represented in the XML element’s attribute, while all other values are represented in the Attribute element’s contents.

Rule 9.4.4.4
```
MvAttributeContents ::= < attribute-name> (EmbeddedObject+ | EnumAttribute+
| ObjRefOrDataValue+) </attribute-name>
```
For a multivalued attribute’s contents, multiple attribute values may be present.Because data values do not have enclosing element tags, each value is delimited with a sequence item tag.

Rule 9.4.4.5
```
EnumAttribute ::= < attribute-name xmi.value="( BooleanAsString |
EnumAsString )"/>
```
An attribute of an enum of boolean type is treated special. The value is represented in an element attribute, rather than the element contents.

Example
```
<FAMIX.Class.isAbstract xmi.value="true"/>
```

Rule 9.4.7.1
```
ObjRefOrDataValue ::= ( StructValue | SequenceValue | UnionValue
|StringValue | CharacterValue | EnumAsElement |IntegralValue | RealValue |
TypeCodeValue | AnyValue |ObjectReference )
```
Based upon the supplied type representation, the appropriate operation is used to
represent the value. For FAMIX the operation StringValue and EnumAsElement return string representations of the data by using the toString method of the value object.

4.6 Testing the Syntax of XMI Documents

By parsing the file the parser validates the syntax of the document using the specified external DTD and prints out messages in case of an error. Using an XMI DTD the parser checks if required child elements (mandatory attributes) are present, if element names are correct according to their specification and element attribute values are elements of the set of specified values.

4.6.1 Example of erroneous XML content

We see here a fragment of the FAMIX DTD together with some XML content that has syntax errors according to the element declaration

```
<!ELEMENT FAMIX.Class.isFinal EMPTY >
<!ATTLIST FAMIX.Class.isFinal
            xmi.value ( true | false | null ) #REQUIRED
>

<!ELEMENT FAMIX.Class (FAMIX.Object.sourceAnchor?,
                       FAMIX.Object.comments?, FAMIX.Entity.name,
                       FAMIX.Entity.uniqueName,
                       FAMIX.Class.belongsToPackage?,
                       FAMIX.Class.isAbstract?,
                       FAMIX.Class.isInterface?, FAMIX.Class.isPublic?,
                       FAMIX.Class.isFinal?, XMI.extension*)? >


<FAMIX.Class xmi.id="_2">
FAMIX.Entity.name>testInterfaceA</FAMIX.Entity.name>
<FAMIX.Class.belonsToPackage>xmi::doctest::testdata
</FAMIX.Class.belongsToPackage>
      <FAMIX.Class.isAbstract xmi.value="true"/>
      <FAMIX.Class.isInterface xmi.value="true"/>
      <FAMIX.Class.isPublic xmi.value="true"/>
      <FAMIX.Class.isFinal xmi.value="tru"/>
 </FAMIX.Class>
```

- According to the DTD every element with name `FAMIX.Class` should contain a child element with name `FAMIX.Entity.uniqueName`. Because the element is not there an error is generated.
- The attribute `xmi.value` of element `FAMIX.Class.isFinal` can have the values `true,false` or `null`. The parsers recognizes that the attribute has a wrong value ‚`tru`‘
- Misspelling of attribute names are also recognized as for `FAMIX.Class.belonsToPackage` instead of `FAMIX.Class.belon`**`g`**`sToPackage`

Furthermore the parser checks the correctness of the tag syntax, but this check is independent of the DTD.

4.7 Reifying XMI objects

After testing the syntax XMI objects are reified by traversing the DOM tree and reading element and attribute values. This functionality is implemented in the class `XMIReader`. The tree traversing is performed according to syntax specification of the generation rules, this means that the syntax is parsed by recursive descent. For each found object in the document an object of class `FamixObject` is instantiated and attribute values are read from the DOM.

There are two ways in which values are read
- The content of a non-empty element is read from a text element that is a child element.
- The content of an empty element, defined with the keyword EMPTY in the DTD (this is for boolean attributes) is read by extracting the value from the element attribute `xmi.value`

The following code sequence that is extracted from `XMIReader` and shows how to read a text element. The variable element is the parent element, which eventually contains content.

```
if (element.hasChildNodes()) {
      NodeList el = element.getChildNodes();
      int      size = el.getLength();

      for (int i = 0; i < size; i++) {
            if (el.item(i) instanceof TXText) {
                  TXText te = (TXText) el.item(i);
                  if (!te.getIsIgnorableWhitespace()) {
                        val = te.getNodeValue();
                  }
            }
       }
}
```

First the list of child nodes are evaluated. This list is searched for elements of type `TXText`. `TXText` is the name of the implementation class for the `TEXT` interface of the `DOM`. This implementation class is specific for the used parser. If the text is not a white space, that means a return or tab character in the document, the string value is read with the method `getNodeValue`.

Example of reading a text value from the element
`<FAMIX.Entity.name>XMITestClass</FAMIX.Entity.name>`:
`<FAMIX.Entity.name></FAMIX.Entity.name>` is the element represented by the variable element in the code above, the string `XMITestClass` is the value of a child element with type `TXText`.

The following shows how to read a value from an element attribute in case of boolean types:
`element.getAttribute("xmi.value")`

Example
For the element
`<FAMIX.Method.isAbstract xmi.value="true"/>`
the statement
`element.getAttribute("xmi.value")`
returns the string "true".

4.8 Testing XMI Values

The static test data is hardcoded in the class `XMIDocTest` and contains the document data that should be produced whenever the content of the package `xmi.doctest.testdata` is transferred. The static testdata was chosen in a way so that every attribute has a value that is read from the reflective interface. Below we see Java source of this package.

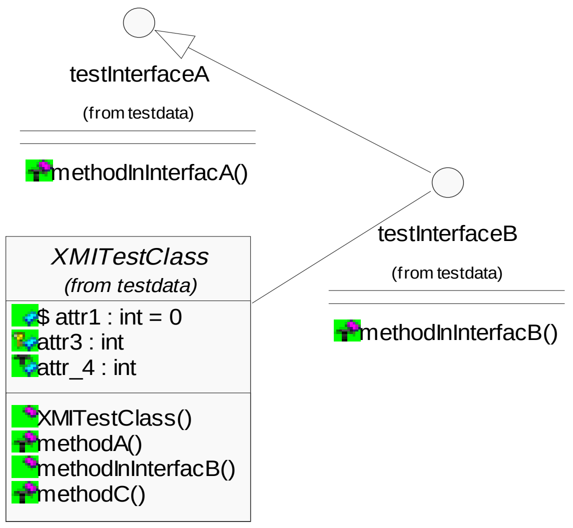


Figure 15: Package famix::xmi::doctest::testdata

```
package FAMIX.xmi.doctest.testdata;

public interface testInterfaceA {

void methodInInterfacA();
}

public interface testInterfaceB extends testInterfaceA {

String methodInInterfacB(int p1, String p2);
}

public abstract class XMITestClass implements testInterfaceB {
    public static final int attr1 = 0;
    volatile String        attr2;
    protected transient int attr3;
    int                    attr_4;

public XMITestClass() {;}

abstract synchronized void methodA();

public final String methodInInterfacB(int p1, String p2) {
      return "";
}

static void methodC() {}

}
```

For each entity or inheritance that is encoded as static test data, the test program compares the data to the correspondig object reified from the XMI document. To do this the object from the document must be identified first. Entities are identified by their unique name, inheritance objects are identified by the subclass and superclass. If an object can not be found among the reified objects a runtime error is generated. Searching the objects is done in the method checkData of class `XMIDoctest`. Comparison of attribute values is done in the method `compareData`. Each attribute value read from the reflective reader is compared to the corresponding attribute value of the `XMIObject` read from the document by comparing its string representation.

Two errors can occur during content testing.

- The object or attribute is not found in the list of reified objects from the `XMIDocument`
- Attribute values are not equal

For the first error an Exception with a message that the object was not found is thrown. In the second case an `XMIValueCompareException` is thrown and the detail message contains information about the attribute name and the incorrect values. In both error cases the test is interrupted immediately.

Example of an error output if an object is not found
```
Object xmi::doctest::testdata::testInterfaceA not found
```

Example of an error output if attribute values do not match. The first line identifies values and attributes. The second line identifies the unique name of the object that produced the error.
```
XMI Value ’false’ doesn’t equal true for element:isAbstract
Error in object xmi::doctest::testdata::testInterfaceA
```

5 Usage Scenarios

5.1 XMI for Large Java Packages

In order to test the behavior of the document generator for large Java packages, documents where generated for the packages java.lang and java.awt.
The generation of an XMI document for java.awt results in a document with the size of approximately 2MB consisting of more than 2000 FAMIX objects. During the execution of the document generator on a Windows NT workstation one could observe an increased memory consumption of about 40MB as maximum value. Out of

memory errors can be avoided by increasing the heap size for the Java interpreter. For java.awt a heap size of 50 MB is sufficient.

```
java -mx50m famix.xmi.FamixXMIDocGenerator java.awt
+FamixTransfer/testTransfer.xml -famix.dtd
```

5.2 Tool Interoperability

In order to implement complex software solutions a combination of tools from different vendors is often 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 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.

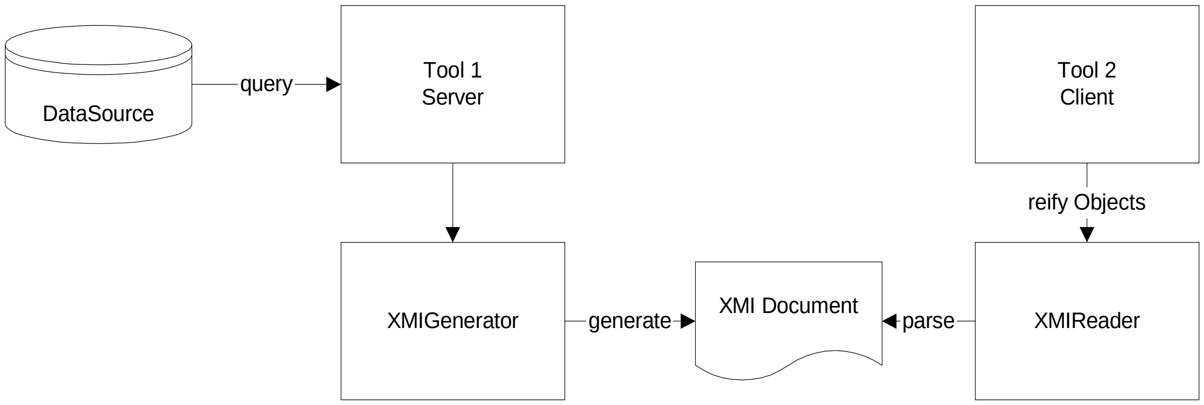


Figure 16: Tool-Interoperability with XMI

In the context of exchanging FAMIX models one could think of a scenario where one tool extracts source code information and generates an XMI file and another tool, for example a metrics tool imports the source code model for further analysis. This scenario requires two components one to generate a file and another to read it in. The generator part is implemented in the class `XMIDocBuilder` an can be reused because it simply implements the XMI document generation rules using the MOF reflective interfaces and without having tool specific interfaces. The class `XMIReader`, used for the test program implements, a generic reader for XMI documents. Thus, a metrics tool could reuse it, so that an import of the model is possible with almost no coding effort. This scenario of a generator and a reader is already implemented for the class `FamixXMIDocGenerator` which generates an XMI document for any Java package and `FamixXMIDocTest` as a client that checks whether the document contains the correct data of a test package.

5.3 Common Meta-model Definition

If two tools want to exchange models they need to agree on a common meta-model. This requires to define a structure and a syntax for encoding the meta data of a model. Because XMI builds on the OMG Meta Object Facility that provides a standard way to define meta-models, the MOF is the best candidate for a meta-meta-model used to instantiate a meta-model. Although it is possible not to use the MOF as a meta-meta-model and generate the DTD manually, but conforming to the XMI specification, this approach implies a very high likelihood that there are errors in the DTD. Possible errors are missing or misspelled element declaration and an improper mapping of inheritance and composition in the meta-model. Using XMI, the XML DTDs for a meta-model are obtained by defining the meta-model in MOF and then applying the XMI generation rules. The generation approach ensures that a given meta-model will always map to the same set of XML DTDs regardless of which vendor implemented the MOF and the XMI DTD generation rules.
This scenario is implemented in the class `DTDGenerator` specifying `FamixModel` as the class that contains a MOF meta-model. The generated DTD for FAMIX is attached in Appendix A.

5.4 Exchanging Meta-Models

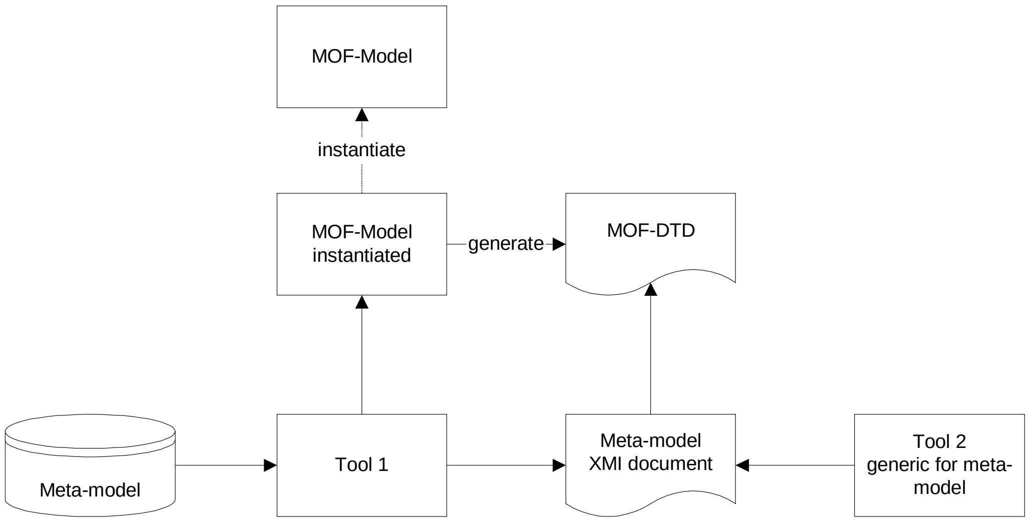


Figure 17: Exchanging Meta-model with XMI

The fact that the MOF model (the description of the MOF itself), can be defined as an instance of the MOF itself means that XMI can also be used to transfer meta-model definitions from one tool to another. From the definition of the MOF in itself a DTD can be generated. Every MOF meta-model then conforms to the syntax specified by this DTD. If two tools need to exchange information an the client doesn't know the meta-model of the model that is to be transferred, the meta-model itself can be transferred first with XMI, so that the model can be understood by the client. This scenario is applicable if the client is a generic application such as browser or query tool for XMI files and should work together with different meta-models.

6 Conclusion

Implementing XMI for FAMIX required the integration of different technologies and standards such as MOF, FAMIX, XML, XMI and Java Reflection. Putting all these parts together requires a good understanding of every part and the interfaces between them. Using existing packages from the XMI Framework and the XML parser from IBM Alphaworks helped to accomplish this task, because they are highly reusable and rely on the OMG standards XMI and MOF and on the W3C standard DOM.
Using XMI as an interchange format for FAMIX causes no problems because as an object-oriented meta-model FAMIX can easily be instantiated from the MOF and therefore an integration with the DTD- and the document generator is straightforward. The downside of it is, that the MOF model itself must first be implemented. A full-featured MOF implementation is complex and should be done once and then be provided as a standard component. Because Sun has submitted a JSR (Java specification request) for an implementation of the MOF model in Java it is quite likely that a standard Java package for the MOF model will be available in the near future.
FAMIX is a model with a small complexity. Despite that the DTD becomes large and is very hard to create and maintain without tool support. This shows us that non-trivial meta-models require a tool like the DTD generator demonstrated in this project. The necessity of tool support becomes even clearer if one considers that the DTD for UML has the size of 200 pages. Another problem that arises when transferring large model is the size of the document itself. Every attribute value is represented as XML-elements and therefore a node in the DOM tree that is returned by the parser. Parsing an XMI file that contains the model of a complex software system consisting of several hundred classes has an increased memory and time complexity and is possibly beyond the resources provided by the hardware, so that other strategies such as splitting the file have to be taken in mind.

6.1 Further Work

Because XMI for FAMIX interfaces various technological areas such as meta-data architectures, data interchange formats, XML, source code models and reflection there are many interesting starting points to do further work.

In the area of interchange formats it is possible to integrate a CDIF writer and develop a converter from CDIF to XMI and vice versa. The CDIF writer can be implemented as a class that implements the `FamixDataHandler` interface and generates output in the CDIF format. A conversion from XMI to CDIF requires and XMI Reader that reifies objects from an XMI document and a CDIF writer. The XMI reader is already implemented in the class `XMIReader` so that the only development effort needed is to code the writer. For a CDIF to XMI conversion it would be necessary to implement a CDIF reader. The XMI writer could then be reused.

Currently using Java reflective interface covers only a subset of FAMIX from the source code. Other tools such as SNiFF+ offer extended capabilities in this area and can also extract invocation and attribute accesses. The class `JavaReflectiveReader` could be replaced by one that uses the SNIFF tool and generate XMI documents for a higher level of extraction.

The DTDGenerator used in this project is capable of generating DTDs for every MOF meta-model. Using this generator XMI DTDs for meta-models and models other than FAMIX can be generated. Interesting areas here are models and meta-models for content- and knowledgemanagement and data warehousing.
Existing DTDs could be reengineered by finding a MOF based model or meta-model for the meta-data that should be encoded as a DTD and then use the DTD Generator to generate it. Because the resulting DTDs are standardized they are easier to read an to maintain than those created with a proprietary methodology.

Based on the experienced gained in this project XMI documents become very large and although human readable it is not easy to identify objects an find required information using a standard text editor. To solve this problem new query and visualization tools are required. A query tool could be based on the DOM API and implement functionality to retrieve sets of XMI objects. It should offer a high-level query language similar to SQL in order to define queries. The result of a query should the be visualized with a GUI – tool that offers navigability through the result set and the displaying of data.

7 References

[MOF99] Object Management Group, Meta Object Facility (MOF) Specification, OMG Document ad/99-09-09, Object Managment Group, September 1999

[XMI98] Object Management Group, XML Metadata Interchange (XMI) Specification, OMG Document ad/98-10-05, Object Managment Group, October 1998

[Deme99a] Serge Demeyer, Sander Tichelaar and Patrick Steyaert, FAMIX 2.0, The Famoos Information Exchange Model, University of Berne, September 1999

[Deme99b] Serge Demeyer, Stéphane Ducasse, Sander Tichelaar, Why FAMIX not UML, UML Shortcomings for Coping with Round-trip Engineering, University of Berne, July 1999

[Tich99] Sander Tichelaar, FAMIX Java Language Plugin 1.0, University of Berne, September 1999

[SUN99] Sun Microsystems 901 San Antonio Rd., Palo Alto, CA 94303 USA, The Java Development Kit, 1994-1999

[Camp96] Mary Campione, Kathy Walrath, The Java tutorial, Addison-Wesley, 1996

[Page00] Meilir Page – Jones, Fundamentals of Object-Oriented Design in UML, Addison-Wesley, 2000

[Gros99] Tim Grose, XMI Application Framework, IBM Corp., November 1999

[IBM99] IBM Alphaworks, XMI Toolkit Version 1.05, November 1999

[IBM99a] IBM Alphaworks, XML4J, XML Parser for Java

[Brad98] The XML companion, Neil Bradley, Addison-Wesley 1998

[DOM98] Document Object Model (DOM) Level 1 Specification, Version 1.0,W3C Recommendation, October, 1998

8 Appendix A: FAMIX DTD

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

<!-- XMI Automatic DTD Generation       -->
<!-- Date: Mon May 15 21:24:29 CEST 2000 -->
<!-- Meta-model: FAMIX             -->

<!-- _______________________________________________________________ -->
<!--                                                                 -->
<!-- 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 "1.0"
            timestamp CDATA #IMPLIED
            verified (true | false) #IMPLIED
>

<!-- _______________________________________________________________ -->
<!--                                                                 -->
<!-- XMI.header contains documentation and identifies the model,     -->
<!-- meta-model, and metameta-model                                    -->
<!-- _______________________________________________________________ -->

<!ELEMENT XMI.header (XMI.documentation?, XMI.model*, XMI.meta-model*,
                      XMI.metameta-model*) >

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

<!-- _______________________________________________________________ -->
<!--                                                                 -->
<!-- XMI.element.att defines the attributes that each XML element    -->
<!-- that corresponds to a meta-model 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 meta-model 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
               ’xml:link CDATA #IMPLIED inline (true | false) #IMPLIED
                actuate (show | user) #IMPLIED href CDATA #IMPLIED role
                CDATA #IMPLIED title CDATA #IMPLIED show (embed | replace
                | new) #IMPLIED behavior CDATA #IMPLIED xmi.idref IDREF
                #IMPLIED xmi.uuidref 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.meta-model identifies the meta-model(s) for the transferred   -->
<!-- data                                                            -->
<!-- _______________________________________________________________ -->

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

<!-- _______________________________________________________________ -->
<!--                                                                 -->
<!-- XMI.metameta-model identifies the metameta-model(s) for the       -->
<!-- transferred data                                                -->
<!-- _______________________________________________________________ -->

<!ELEMENT XMI.metameta-model ANY >
<!ATTLIST XMI.metameta-model
            %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 meta-model(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 meta-model(s) in the header -->
<!-- _______________________________________________________________ -->

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

<!-- _______________________________________________________________ -->
<!--                                                                 -->
<!-- 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 meta-model                                                   -->
<!-- _______________________________________________________________ -->

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

<!-- _______________________________________________________________ -->
<!--                                                                 -->
<!-- This section contains the declaration of XML elements           -->
<!-- representing data types                                         -->
<!-- _______________________________________________________________ -->

<!ELEMENT XMI.TypeDefinitions ANY >

<!ELEMENT XMI.field ANY >

<!ELEMENT XMI.seqItem ANY >

<!ELEMENT XMI.octetStream (#PCDATA) >

<!ELEMENT XMI.unionDiscrim ANY >

<!ELEMENT XMI.enum EMPTY >
<!ATTLIST XMI.enum
            xmi.value CDATA #REQUIRED
>

<!ELEMENT XMI.any ANY >
<!ATTLIST XMI.any
            %XMI.link.att;
            xmi.type CDATA #IMPLIED
            xmi.name CDATA #IMPLIED
>

<!ELEMENT XMI.CorbaTypeCode (XMI.CorbaTcAlias | XMI.CorbaTcStruct |
                             XMI.CorbaTcSequence | XMI.CorbaTcArray |
                             XMI.CorbaTcEnum | XMI.CorbaTcUnion |
                             XMI.CorbaTcExcept | XMI.CorbaTcString |
                             XMI.CorbaTcWstring | XMI.CorbaTcShort |
                             XMI.CorbaTcLong | XMI.CorbaTcUshort |
                             XMI.CorbaTcUlong | XMI.CorbaTcFloat |
                             XMI.CorbaTcDouble | XMI.CorbaTcBoolean |
                             XMI.CorbaTcChar | XMI.CorbaTcWchar |
                             XMI.CorbaTcOctet | XMI.CorbaTcAny |
                             XMI.CorbaTcTypeCode | XMI.CorbaTcPrincipal |
                             XMI.CorbaTcNull | XMI.CorbaTcVoid |
                             XMI.CorbaTcLongLong |
                             XMI.CorbaTcLongDouble) >
<!ATTLIST XMI.CorbaTypeCode
            %XMI.element.att;
>

<!ELEMENT XMI.CorbaTcAlias (XMI.CorbaTypeCode) >
<!ATTLIST XMI.CorbaTcAlias
            xmi.tcName CDATA #REQUIRED
            xmi.tcId   CDATA #IMPLIED
>
```

```
<!ELEMENT XMI.CorbaTcStruct (XMI.CorbaTcField)* >
<!ATTLIST XMI.CorbaTcStruct
            xmi.tcName CDATA #REQUIRED
            xmi.tcId   CDATA #IMPLIED
>

<!ELEMENT XMI.CorbaTcField (XMI.CorbaTypeCode) >
<!ATTLIST XMI.CorbaTcField
            xmi.tcName CDATA #REQUIRED
>

<!ELEMENT XMI.CorbaTcSequence (XMI.CorbaTypeCode |
                               XMI.CorbaRecursiveType) >
<!ATTLIST XMI.CorbaTcSequence
            xmi.tcLength CDATA #REQUIRED
>

<!ELEMENT XMI.CorbaRecursiveType EMPTY >
<!ATTLIST XMI.CorbaRecursiveType
            xmi.offset CDATA #REQUIRED
>

<!ELEMENT XMI.CorbaTcArray (XMI.CorbaTypeCode) >
<!ATTLIST XMI.CorbaTcArray
            xmi.tcLength CDATA #REQUIRED
>

<!ELEMENT XMI.CorbaTcObjRef EMPTY >
<!ATTLIST XMI.CorbaTcObjRef
            xmi.tcName CDATA #REQUIRED
            xmi.tcId   CDATA #IMPLIED
>

<!ELEMENT XMI.CorbaTcEnum (XMI.CorbaTcEnumLabel) >
<!ATTLIST XMI.CorbaTcEnum
            xmi.tcName CDATA #REQUIRED
            xmi.tcId   CDATA #IMPLIED
>

<!ELEMENT XMI.CorbaTcEnumLabel EMPTY >
<!ATTLIST XMI.CorbaTcEnumLabel
            xmi.tcName CDATA #REQUIRED
>

<!ELEMENT XMI.CorbaTcUnionMbr (XMI.CorbaTypeCode, XMI.any) >
<!ATTLIST XMI.CorbaTcUnionMbr
            xmi.tcName CDATA #REQUIRED
>

<!ELEMENT XMI.CorbaTcUnion (XMI.CorbaTypeCode, XMI.CorbaTcUnionMbr*) >
<!ATTLIST XMI.CorbaTcUnion
            xmi.tcName CDATA #REQUIRED
            xmi.tcId   CDATA #IMPLIED
>

<!ELEMENT XMI.CorbaTcExcept (XMI.CorbaTcField)* >
<!ATTLIST XMI.CorbaTcExcept
            xmi.tcName CDATA #REQUIRED
            xmi.tcId   CDATA #IMPLIED
>

<!ELEMENT XMI.CorbaTcString EMPTY >
<!ATTLIST XMI.CorbaTcString
            xmi.tcLength CDATA #REQUIRED
>

<!ELEMENT XMI.CorbaTcWstring EMPTY >
<!ATTLIST XMI.CorbaTcWstring
            xmi.tcLength CDATA #REQUIRED
>

<!ELEMENT XMI.CorbaTcFixed EMPTY >
<!ATTLIST XMI.CorbaTcFixed
            xmi.tcDigits CDATA #REQUIRED
            xmi.tcScale  CDATA #REQUIRED
>

<!ELEMENT XMI.CorbaTcShort EMPTY >

<!ELEMENT XMI.CorbaTcLong EMPTY >

<!ELEMENT XMI.CorbaTcUshort EMPTY >

<!ELEMENT XMI.CorbaTcUlong EMPTY >

<!ELEMENT XMI.CorbaTcFloat EMPTY >

<!ELEMENT XMI.CorbaTcDouble EMPTY >
```

```
<!ELEMENT XMI.CorbaTcBoolean EMPTY >

<!ELEMENT XMI.CorbaTcChar EMPTY >

<!ELEMENT XMI.CorbaTcWchar EMPTY >

<!ELEMENT XMI.CorbaTcOctet EMPTY >

<!ELEMENT XMI.CorbaTcAny EMPTY >

<!ELEMENT XMI.CorbaTcTypeCode EMPTY >

<!ELEMENT XMI.CorbaTcPrincipal EMPTY >

<!ELEMENT XMI.CorbaTcNull EMPTY >

<!ELEMENT XMI.CorbaTcVoid EMPTY >

<!ELEMENT XMI.CorbaTcLongLong EMPTY >

<!ELEMENT XMI.CorbaTcLongDouble EMPTY >

<!-- _______________________________________________________________ -->
<!--                                                                 -->
<!-- META-MODEL:  FAMIX                                               -->
<!-- _______________________________________________________________ -->


<!-- _______________________________________________________________ -->
<!--                                                                 -->
<!-- META-MODEL CLASS:  Model                                         -->
<!-- _______________________________________________________________ -->


<!ELEMENT FAMIX.Model.exporterName (#PCDATA | XMI.reference)* >

<!ELEMENT FAMIX.Model.exporterVersion (#PCDATA | XMI.reference)* >

<!ELEMENT FAMIX.Model.exporterDate (#PCDATA | XMI.reference)* >

<!ELEMENT FAMIX.Model.exporterTime (#PCDATA | XMI.reference)* >

<!ELEMENT FAMIX.Model.publisherName (#PCDATA | XMI.reference)* >

<!ELEMENT FAMIX.Model.parsedSystemName (#PCDATA | XMI.reference)* >

<!ELEMENT FAMIX.Model.extractionLevel (#PCDATA | XMI.reference)* >

<!ELEMENT FAMIX.Model.sourceLanguage (#PCDATA | XMI.reference)* >

<!ELEMENT FAMIX.Model.sourceDialect (#PCDATA | XMI.reference)* >

<!ELEMENT FAMIX.Model (FAMIX.Model.exporterName?,
                       FAMIX.Model.exporterVersion?,
                       FAMIX.Model.exporterDate?,
                       FAMIX.Model.exporterTime?,
                       FAMIX.Model.publisherName?,
                       FAMIX.Model.parsedSystemName?,
                       FAMIX.Model.extractionLevel?,
                       FAMIX.Model.sourceLanguage?,
                       FAMIX.Model.sourceDialect?, XMI.extension*)? >
<!ATTLIST FAMIX.Model
            %XMI.element.att;
            %XMI.link.att;
>

<!-- _______________________________________________________________ -->
<!--                                                                 -->
<!-- META-MODEL CLASS:  Object                                        -->
<!-- _______________________________________________________________ -->


<!ELEMENT FAMIX.Object.sourceAnchor (#PCDATA | XMI.reference)* >

<!ELEMENT FAMIX.Object.comments (#PCDATA | XMI.reference)* >

<!ELEMENT FAMIX.Object (FAMIX.Object.sourceAnchor?,
                        FAMIX.Object.comments?, XMI.extension*)? >
<!ATTLIST FAMIX.Object
            %XMI.element.att;
            %XMI.link.att;
>

<!-- _______________________________________________________________ -->
<!--                                                                 -->
<!-- META-MODEL CLASS:  Entity                                        -->
<!-- _______________________________________________________________ -->
```

```
<!ELEMENT FAMIX.Entity.name (#PCDATA | XMI.reference)* >

<!ELEMENT FAMIX.Entity.uniqueName (#PCDATA | XMI.reference)* >

<!ELEMENT FAMIX.Entity (FAMIX.Object.sourceAnchor?,
                        FAMIX.Object.comments?, FAMIX.Entity.name,
                        FAMIX.Entity.uniqueName, XMI.extension*)? >
<!ATTLIST FAMIX.Entity
            %XMI.element.att;
            %XMI.link.att;
>

<!-- _______________________________________________________________ -->
<!--                                                                 -->
<!-- META-MODEL CLASS:  Class                                         -->
<!-- _______________________________________________________________ -->


<!ELEMENT FAMIX.Class.belongsToPackage (#PCDATA | XMI.reference)* >

<!ELEMENT FAMIX.Class.isAbstract EMPTY >
<!ATTLIST FAMIX.Class.isAbstract
            xmi.value ( true | false | null ) #REQUIRED
>

<!ELEMENT FAMIX.Class.isInterface EMPTY >
<!ATTLIST FAMIX.Class.isInterface
            xmi.value ( true | false | null ) #REQUIRED
>

<!ELEMENT FAMIX.Class.isPublic EMPTY >
<!ATTLIST FAMIX.Class.isPublic
            xmi.value ( true | false | null ) #REQUIRED
>

<!ELEMENT FAMIX.Class.isFinal EMPTY >
<!ATTLIST FAMIX.Class.isFinal
            xmi.value ( true | false | null ) #REQUIRED
>

<!ELEMENT FAMIX.Class (FAMIX.Object.sourceAnchor?,
                       FAMIX.Object.comments?, FAMIX.Entity.name,
                       FAMIX.Entity.uniqueName,
                       FAMIX.Class.belongsToPackage?,
                       FAMIX.Class.isAbstract?,
                       FAMIX.Class.isInterface?, FAMIX.Class.isPublic?,
                       FAMIX.Class.isFinal?, XMI.extension*)? >
<!ATTLIST FAMIX.Class
            %XMI.element.att;
            %XMI.link.att;
>

<!-- _______________________________________________________________ -->
<!--                                                                 -->
<!-- META-MODEL CLASS:  Package                                       -->
<!-- _______________________________________________________________ -->


<!ELEMENT FAMIX.Package.belongsToPackage (#PCDATA | XMI.reference)* >

<!ELEMENT FAMIX.Package (FAMIX.Object.sourceAnchor?,
                         FAMIX.Object.comments?, FAMIX.Entity.name,
                         FAMIX.Entity.uniqueName,
                         FAMIX.Package.belongsToPackage?,
                         XMI.extension*)? >
<!ATTLIST FAMIX.Package
            %XMI.element.att;
            %XMI.link.att;
>

<!-- _______________________________________________________________ -->
<!--                                                                 -->
<!-- META-MODEL CLASS:  BehaviouralEntity                             -->
<!-- _______________________________________________________________ -->


<!ELEMENT FAMIX.BehaviouralEntity.accessControlQualifier (#PCDATA |
                                                           XMI.reference)*
                                                            >

<!ELEMENT FAMIX.BehaviouralEntity.signature (#PCDATA | XMI.reference)* >

<!ELEMENT FAMIX.BehaviouralEntity.isPureAccessor EMPTY >
<!ATTLIST FAMIX.BehaviouralEntity.isPureAccessor
            xmi.value ( true | false | null ) #REQUIRED
>

<!ELEMENT FAMIX.BehaviouralEntity.declaredReturnType (#PCDATA |
                                                       XMI.reference)* >

<!ELEMENT FAMIX.BehaviouralEntity.declaredReturnClass (#PCDATA |
                                                        XMI.reference)* >

<!ELEMENT FAMIX.BehaviouralEntity (FAMIX.Object.sourceAnchor?,
                                   FAMIX.Object.comments?,
                                   FAMIX.Entity.name,
                                   FAMIX.Entity.uniqueName,
                                   FAMIX.BehaviouralEntity.accessControlQualifier?,
                                   FAMIX.BehaviouralEntity.signature,
                                   FAMIX.BehaviouralEntity.isPureAccessor?,
                                   FAMIX.BehaviouralEntity.declaredReturnType?,
                                   FAMIX.BehaviouralEntity.declaredReturnClass?,
                                   XMI.extension*)? >
<!ATTLIST FAMIX.BehaviouralEntity
            %XMI.element.att;
            %XMI.link.att;
>

<!-- _______________________________________________________________ -->
<!--                                                                 -->
<!-- META-MODEL CLASS:  Method                                        -->
<!-- _______________________________________________________________ -->


<!ELEMENT FAMIX.Method.belongsToClass (#PCDATA | XMI.reference)* >

<!ELEMENT FAMIX.Method.hasClassScope EMPTY >
<!ATTLIST FAMIX.Method.hasClassScope
            xmi.value ( true | false | null ) #REQUIRED
>

<!ELEMENT FAMIX.Method.isAbstract EMPTY >
<!ATTLIST FAMIX.Method.isAbstract
            xmi.value ( true | false | null ) #REQUIRED
>

<!ELEMENT FAMIX.Method.isConstructor EMPTY >
<!ATTLIST FAMIX.Method.isConstructor
            xmi.value ( true | false | null ) #REQUIRED
>

<!ELEMENT FAMIX.Method.isFinal EMPTY >
<!ATTLIST FAMIX.Method.isFinal
            xmi.value ( true | false | null ) #REQUIRED
>

<!ELEMENT FAMIX.Method.isSynchronized EMPTY >
<!ATTLIST FAMIX.Method.isSynchronized
            xmi.value ( true | false | null ) #REQUIRED
>

<!ELEMENT FAMIX.Method.isNative EMPTY >
<!ATTLIST FAMIX.Method.isNative
            xmi.value ( true | false | null ) #REQUIRED
>

<!ELEMENT FAMIX.Method (FAMIX.Object.sourceAnchor?,
                        FAMIX.Object.comments?, FAMIX.Entity.name,
                        FAMIX.Entity.uniqueName,
                        FAMIX.BehaviouralEntity.accessControlQualifier?,
                        FAMIX.BehaviouralEntity.signature,
                        FAMIX.BehaviouralEntity.isPureAccessor?,
                        FAMIX.BehaviouralEntity.declaredReturnType?,
                        FAMIX.BehaviouralEntity.declaredReturnClass?,
                        FAMIX.Method.belongsToClass?,
                        FAMIX.Method.hasClassScope?,
                        FAMIX.Method.isAbstract?,
                        FAMIX.Method.isConstructor?,
                        FAMIX.Method.isFinal?,
                        FAMIX.Method.isSynchronized?,
                        FAMIX.Method.isNative?, XMI.extension*)? >
<!ATTLIST FAMIX.Method
            %XMI.element.att;
            %XMI.link.att;
>

<!-- _______________________________________________________________ -->
<!--                                                                 -->
<!-- META-MODEL CLASS:  Function                                      -->
<!-- _______________________________________________________________ -->


<!ELEMENT FAMIX.Function.belongsToPackage (#PCDATA | XMI.reference)* >

<!ELEMENT FAMIX.Function (FAMIX.Object.sourceAnchor?,
                          FAMIX.Object.comments?, FAMIX.Entity.name,
                          FAMIX.Entity.uniqueName,
                          FAMIX.BehaviouralEntity.accessControlQualifier?,
                          FAMIX.BehaviouralEntity.signature,
                          FAMIX.BehaviouralEntity.isPureAccessor?,
                          FAMIX.BehaviouralEntity.declaredReturnType?,
                          FAMIX.BehaviouralEntity.declaredReturnClass?,
                          FAMIX.Function.belongsToPackage?,
                          XMI.extension*)? >
<!ATTLIST FAMIX.Function
            %XMI.element.att;
            %XMI.link.att;
>

<!-- _______________________________________________________________ -->
<!--                                                                 -->
<!-- META-MODEL CLASS:  StructuralEntity                              -->
<!-- _______________________________________________________________ -->


<!ELEMENT FAMIX.StructuralEntity.declaredType (#PCDATA |
                                               XMI.reference)* >

<!ELEMENT FAMIX.StructuralEntity.declaredClass (#PCDATA |
                                                XMI.reference)* >

<!ELEMENT FAMIX.StructuralEntity (FAMIX.Object.sourceAnchor?,
                                  FAMIX.Object.comments?,
                                  FAMIX.Entity.name,
                                  FAMIX.Entity.uniqueName,
                                  FAMIX.StructuralEntity.declaredType?,
                                  FAMIX.StructuralEntity.declaredClass?,
                                  XMI.extension*)? >
<!ATTLIST FAMIX.StructuralEntity
            %XMI.element.att;
            %XMI.link.att;
>

<!-- _______________________________________________________________ -->
<!--                                                                 -->
<!-- META-MODEL CLASS:  Attribute                                     -->
<!-- _______________________________________________________________ -->


<!ELEMENT FAMIX.Attribute.belongsToClass (#PCDATA | XMI.reference)* >

<!ELEMENT FAMIX.Attribute.accessControlQualifier (#PCDATA |
                                                  XMI.reference)* >

<!ELEMENT FAMIX.Attribute.hasClassScope EMPTY >
<!ATTLIST FAMIX.Attribute.hasClassScope
            xmi.value ( true | false | null ) #REQUIRED
>

<!ELEMENT FAMIX.Attribute.isFinal EMPTY >
<!ATTLIST FAMIX.Attribute.isFinal
            xmi.value ( true | false | null ) #REQUIRED
>

<!ELEMENT FAMIX.Attribute.isTransient EMPTY >
<!ATTLIST FAMIX.Attribute.isTransient
            xmi.value ( true | false | null ) #REQUIRED
>

<!ELEMENT FAMIX.Attribute.isVolatile EMPTY >
<!ATTLIST FAMIX.Attribute.isVolatile
            xmi.value ( true | false | null ) #REQUIRED
>

<!ELEMENT FAMIX.Attribute (FAMIX.Object.sourceAnchor?,
                           FAMIX.Object.comments?, FAMIX.Entity.name,
                           FAMIX.Entity.uniqueName,
                           FAMIX.StructuralEntity.declaredType?,
                           FAMIX.StructuralEntity.declaredClass?,
                           FAMIX.Attribute.belongsToClass,
                           FAMIX.Attribute.accessControlQualifier?,
                           FAMIX.Attribute.hasClassScope?,
                           FAMIX.Attribute.isFinal?,
                           FAMIX.Attribute.isTransient?,
                           FAMIX.Attribute.isVolatile?, XMI.extension*)? >
<!ATTLIST FAMIX.Attribute
            %XMI.element.att;
            %XMI.link.att;
>

<!-- _______________________________________________________________ -->
<!--                                                                 -->
<!-- META-MODEL CLASS:  GlobalVariable                                -->
<!-- _______________________________________________________________ -->
```

```
<!ELEMENT FAMIX.GlobalVariable.belongsToPackage (#PCDATA |
                                                 XMI.reference)* >

<!ELEMENT FAMIX.GlobalVariable (FAMIX.Object.sourceAnchor?,
                                FAMIX.Object.comments?,
                                FAMIX.Entity.name,
                                FAMIX.Entity.uniqueName,
                                FAMIX.StructuralEntity.declaredType?,
                                FAMIX.StructuralEntity.declaredClass?,
                                FAMIX.GlobalVariable.belongsToPackage?,
                                XMI.extension*)? >
<!ATTLIST FAMIX.GlobalVariable
            %XMI.element.att;
            %XMI.link.att;
>

<!-- _______________________________________________________________ -->
<!--                                                                 -->
<!-- META-MODEL CLASS:  ImplicitVariable                              -->
<!-- _______________________________________________________________ -->


<!ELEMENT FAMIX.ImplicitVariable.belongsToContext (#PCDATA |
                                                   XMI.reference)* >

<!ELEMENT FAMIX.ImplicitVariable (FAMIX.Object.sourceAnchor?,
                                  FAMIX.Object.comments?,
                                  FAMIX.Entity.name,
                                  FAMIX.Entity.uniqueName,
                                  FAMIX.StructuralEntity.declaredType?,
                                  FAMIX.StructuralEntity.declaredClass?,
                                  FAMIX.ImplicitVariable.belongsToContext?,
                                  XMI.extension*)? >
<!ATTLIST FAMIX.ImplicitVariable
            %XMI.element.att;
            %XMI.link.att;
>

<!-- _______________________________________________________________ -->
<!--                                                                 -->
<!-- META-MODEL CLASS:  LocalVariable                                 -->
<!-- _______________________________________________________________ -->


<!ELEMENT FAMIX.LocalVariable.belongsToBehaviour (#PCDATA |
                                                  XMI.reference)* >

<!ELEMENT FAMIX.LocalVariable (FAMIX.Object.sourceAnchor?,
                               FAMIX.Object.comments?,
                               FAMIX.Entity.name,
                               FAMIX.Entity.uniqueName,
                               FAMIX.StructuralEntity.declaredType?,
                               FAMIX.StructuralEntity.declaredClass?,
                               FAMIX.LocalVariable.belongsToBehaviour,
                               XMI.extension*)? >
<!ATTLIST FAMIX.LocalVariable
            %XMI.element.att;
            %XMI.link.att;
>

<!-- _______________________________________________________________ -->
<!--                                                                 -->
<!-- META-MODEL CLASS:  FormalParameter                               -->
<!-- _______________________________________________________________ -->


<!ELEMENT FAMIX.FormalParameter.belongsToBehaviour (#PCDATA |
                                                    XMI.reference)* >

<!ELEMENT FAMIX.FormalParameter.position (#PCDATA | XMI.reference)* >

<!ELEMENT FAMIX.FormalParameter (FAMIX.Object.sourceAnchor?,
                                 FAMIX.Object.comments?,
                                 FAMIX.Entity.name,
                                 FAMIX.Entity.uniqueName,
                                 FAMIX.StructuralEntity.declaredType?,
                                 FAMIX.StructuralEntity.declaredClass?,
                                 FAMIX.FormalParameter.belongsToBehaviour,
                                 FAMIX.FormalParameter.position,
                                 XMI.extension*)? >
<!ATTLIST FAMIX.FormalParameter
            %XMI.element.att;
            %XMI.link.att;
>

<!-- _______________________________________________________________ -->
<!--                                                                 -->
<!-- META-MODEL CLASS:  Association                                   -->
<!-- _______________________________________________________________ -->
```

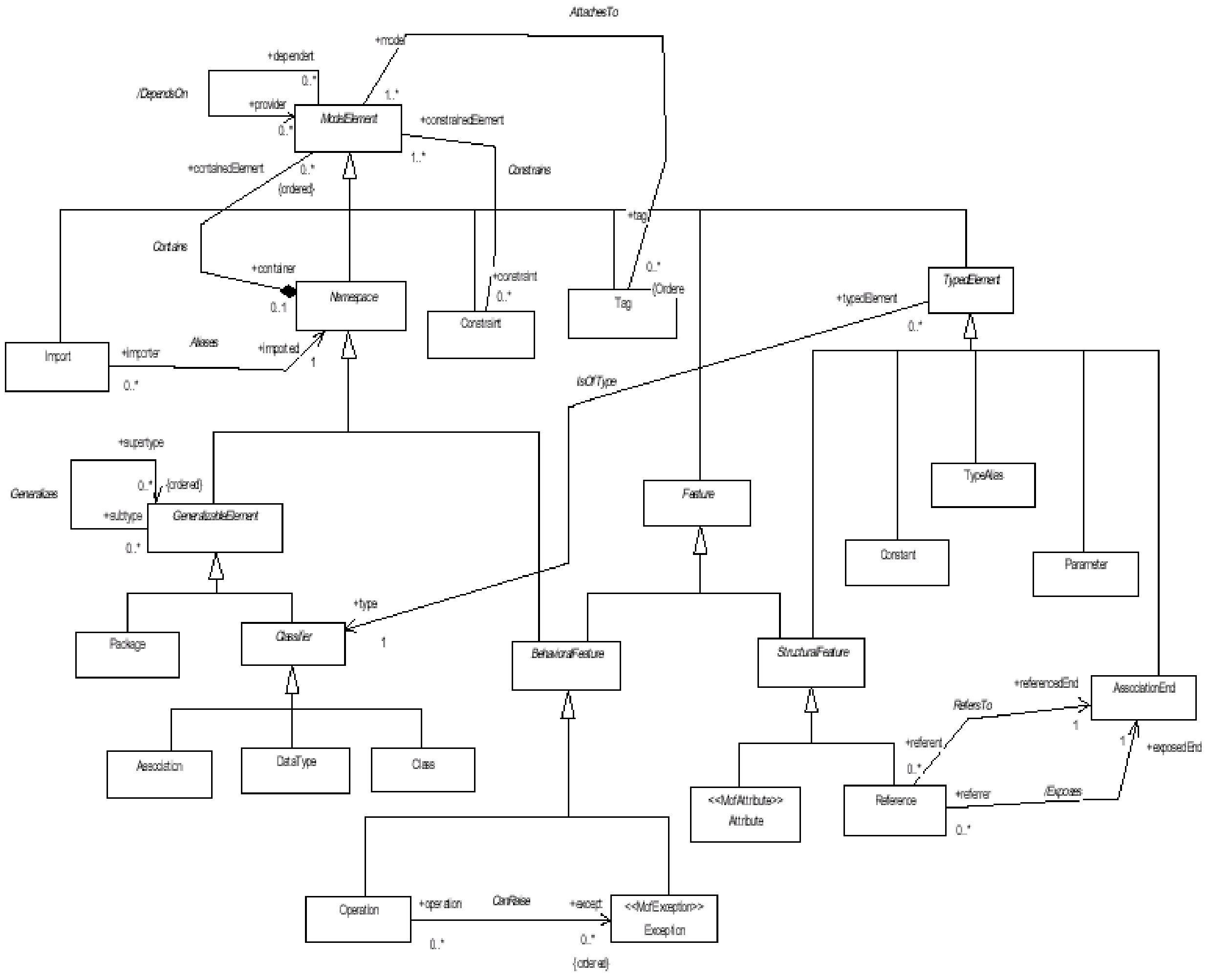
```
<!ELEMENT FAMIX.Association (FAMIX.Object.sourceAnchor?,
                             FAMIX.Object.comments?, XMI.extension*)? >
<!ATTLIST FAMIX.Association
            %XMI.element.att;
            %XMI.link.att;
>

<!-- _______________________________________________________________ -->
<!--                                                                 -->
<!-- META-MODEL CLASS:  InheritanceDefinition                         -->
<!-- _______________________________________________________________ -->


<!ELEMENT FAMIX.InheritanceDefinition.subclass (#PCDATA |
                                                XMI.reference)* >

<!ELEMENT FAMIX.InheritanceDefinition.superclass (#PCDATA |
                                                  XMI.reference)* >

<!ELEMENT FAMIX.InheritanceDefinition.accessControlQualifier (#PCDATA |
                                                              XMI.reference)*
                                                               >

<!ELEMENT FAMIX.InheritanceDefinition.index (#PCDATA | XMI.reference)* >

<!ELEMENT FAMIX.InheritanceDefinition (FAMIX.Object.sourceAnchor?,
                                       FAMIX.Object.comments?,
                                       FAMIX.InheritanceDefinition.subclass,
                                       FAMIX.InheritanceDefinition.superclass,
                                       FAMIX.InheritanceDefinition.accessControlQualifier?,
                                       FAMIX.InheritanceDefinition.index?,
                                       XMI.extension*)? >
<!ATTLIST FAMIX.InheritanceDefinition
            %XMI.element.att;
            %XMI.link.att;
>

<!-- _______________________________________________________________ -->
<!--                                                                 -->
<!-- META-MODEL CLASS:  Invocation                                    -->
<!-- _______________________________________________________________ -->


<!ELEMENT FAMIX.Invocation.invokedBy (#PCDATA | XMI.reference)* >

<!ELEMENT FAMIX.Invocation.invokes (#PCDATA | XMI.reference)* >

<!ELEMENT FAMIX.Invocation.base (#PCDATA | XMI.reference)* >

<!ELEMENT FAMIX.Invocation (FAMIX.Object.sourceAnchor?,
                            FAMIX.Object.comments?,
                            FAMIX.Invocation.invokedBy,
                            FAMIX.Invocation.invokes,
                            FAMIX.Invocation.base?, XMI.extension*)? >
<!ATTLIST FAMIX.Invocation
            %XMI.element.att;
            %XMI.link.att;
>

<!-- _______________________________________________________________ -->
<!--                                                                 -->
<!-- META-MODEL CLASS:  Access                                        -->
<!-- _______________________________________________________________ -->


<!ELEMENT FAMIX.Access.accesses (#PCDATA | XMI.reference)* >

<!ELEMENT FAMIX.Access.accessedIn (#PCDATA | XMI.reference)* >

<!ELEMENT FAMIX.Access (FAMIX.Object.sourceAnchor?,
                        FAMIX.Object.comments?, FAMIX.Access.accesses,
                        FAMIX.Access.accessedIn, XMI.extension*)? >
<!ATTLIST FAMIX.Access
            %XMI.element.att;
            %XMI.link.att;
>

<!-- _______________________________________________________________ -->
<!--                                                                 -->
<!-- META-MODEL CLASS:  Argument                                      -->
<!-- _______________________________________________________________ -->


<!ELEMENT FAMIX.Argument.position (#PCDATA | XMI.reference)* >

<!ELEMENT FAMIX.Argument (FAMIX.Object.sourceAnchor?,
                          FAMIX.Object.comments?,
                          FAMIX.Argument.position, XMI.extension*)? >
<!ATTLIST FAMIX.Argument
            %XMI.element.att;
            %XMI.link.att;
>

<!-- _______________________________________________________________ -->
<!--                                                                 -->
<!-- META-MODEL CLASS:  ExpressionArgument                            -->
<!-- _______________________________________________________________ -->


<!ELEMENT FAMIX.ExpressionArgument (FAMIX.Object.sourceAnchor?,
                                    FAMIX.Object.comments?,
                                    FAMIX.Argument.position,
                                    XMI.extension*)? >
<!ATTLIST FAMIX.ExpressionArgument
            %XMI.element.att;
            %XMI.link.att;
>

<!-- _______________________________________________________________ -->
<!--                                                                 -->
<!-- META-MODEL CLASS:  AccessArgument                                -->
<!-- _______________________________________________________________ -->


<!ELEMENT FAMIX.AccessArgument (FAMIX.Object.sourceAnchor?,
                                FAMIX.Object.comments?,
                                FAMIX.Argument.position,
                                XMI.extension*)? >
<!ATTLIST FAMIX.AccessArgument
            %XMI.element.att;
            %XMI.link.att;
>

<!ELEMENT FAMIX ((FAMIX.Model | FAMIX.Object | FAMIX.Entity |
                 FAMIX.Class | FAMIX.Package | FAMIX.BehaviouralEntity |
                 FAMIX.Method | FAMIX.Function | FAMIX.StructuralEntity |
                 FAMIX.Attribute | FAMIX.GlobalVariable |
                 FAMIX.ImplicitVariable | FAMIX.LocalVariable |
                 FAMIX.FormalParameter | FAMIX.Association |
                 FAMIX.InheritanceDefinition | FAMIX.Invocation |
                 FAMIX.Access | FAMIX.Argument |
                 FAMIX.ExpressionArgument | FAMIX.AccessArgument)*) >
<!ATTLIST FAMIX
            %XMI.element.att;
            %XMI.link.att;
>
```

9 Appendix B: XMI Document for famix::xmi::doctest::testdata

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE XMI SYSTEM "FAMIX.dtd" [
  <!ELEMENT Data ANY>
  <!ATTLIST Data
     xmi.id ID #IMPLIED
     xmi.label CDATA #IMPLIED
     xmi.uuid CDATA #IMPLIED
     xml:link CDATA #IMPLIED
     inline (true|false) #IMPLIED
     actuate (show|user) #IMPLIED
     href CDATA #IMPLIED
     role CDATA #IMPLIED
     title CDATA #IMPLIED
     show (embed|replace|new) #IMPLIED
     behavior CDATA #IMPLIED
     xmi.idref IDREF #IMPLIED
     xmi.uuidref CDATA #IMPLIED
     Data.name CDATA #IMPLIED
     Data.type CDATA #IMPLIED
     Data.delete CDATA #IMPLIED
     Data.data CDATA #IMPLIED>
]>
<XMI xmi.version="1.0" timestamp="Mon May 15 21:24:25 CEST 2000">
  <XMI.header>
    <XMI.documentation>
      <XMI.exporter>XMI MASTER</XMI.exporter>
      <XMI.exporterVersion>1.0</XMI.exporterVersion>
    </XMI.documentation>
    <XMI.meta-model xmi.name="FAMIX" xmi.version="2.0"/>
  </XMI.header>
  <XMI.content>
    <FAMIX.Package xmi.id="_1">
      <FAMIX.Entity.name>testdata</FAMIX.Entity.name>
      <FAMIX.Entity.uniqueName>FAMIX::xmi::doctest::testdata</FAMIX.Entity.uniqueName>
      <FAMIX.Package.belongsToPackage>FAMIX::xmi::doctest</FAMIX.Package.belongsToPackage>
    </FAMIX.Package>
    <FAMIX.Class xmi.id="_2">
      <FAMIX.Entity.name>testInterfaceA</FAMIX.Entity.name>
      <FAMIX.Entity.uniqueName>FAMIX::xmi::doctest::testdata::testInterfaceA</FAMIX.Entity.uniqueName>
      <FAMIX.Class.belongsToPackage>FAMIX::xmi::doctest::testdata</FAMIX.Class.belongsToPackage>
      <FAMIX.Class.isAbstract xmi.value="true"/>
      <FAMIX.Class.isInterface xmi.value="true"/>
      <FAMIX.Class.isPublic xmi.value="true"/>
      <FAMIX.Class.isFinal xmi.value="false"/>
    </FAMIX.Class>
    <FAMIX.Method xmi.id="_3">
      <FAMIX.Entity.name>methodInInterfacA</FAMIX.Entity.name>
      <FAMIX.Entity.uniqueName>FAMIX::xmi::doctest::testdata::testInterfaceA.methodInInterfacA()</FAMIX.Entity.uniqueName>
      <FAMIX.BehaviouralEntity.accessControlQualifier>public</FAMIX.BehaviouralEntity.accessControlQualifier>
      <FAMIX.BehaviouralEntity.signature>methodInInterfacA()</FAMIX.BehaviouralEntity.signature>
      <FAMIX.BehaviouralEntity.isPureAccessor xmi.value="null"/>
      <FAMIX.BehaviouralEntity.declaredReturnType>void</FAMIX.BehaviouralEntity.declaredReturnType>
      <FAMIX.Method.belongsToClass>FAMIX::xmi::doctest::testdata::testInterfaceA</FAMIX.Method.belongsToClass>
      <FAMIX.Method.hasClassScope xmi.value="false"/>
      <FAMIX.Method.isAbstract xmi.value="true"/>
      <FAMIX.Method.isConstructor xmi.value="false"/>
      <FAMIX.Method.isFinal xmi.value="false"/>
      <FAMIX.Method.isSynchronized xmi.value="false"/>
      <FAMIX.Method.isNative xmi.value="false"/>
    </FAMIX.Method>
    <FAMIX.Class xmi.id="_4">
      <FAMIX.Entity.name>testInterfaceB</FAMIX.Entity.name>
      <FAMIX.Entity.uniqueName>FAMIX::xmi::doctest::testdata::testInterfaceB</FAMIX.Entity.uniqueName>
      <FAMIX.Class.belongsToPackage>FAMIX::xmi::doctest::testdata</FAMIX.Class.belongsToPackage>
      <FAMIX.Class.isAbstract xmi.value="true"/>
      <FAMIX.Class.isInterface xmi.value="true"/>
      <FAMIX.Class.isPublic xmi.value="true"/>
      <FAMIX.Class.isFinal xmi.value="false"/>
    </FAMIX.Class>
    <FAMIX.InheritanceDefinition xmi.id="_5">
      <FAMIX.InheritanceDefinition.subclass>FAMIX::xmi::doctest::testdata::testInterfaceB</FAMIX.InheritanceDefinition.subclass>
      <FAMIX.InheritanceDefinition.superclass>FAMIX::xmi::doctest::testdata::testInterfaceA</FAMIX.InheritanceDefinition.superclass>
      <FAMIX.InheritanceDefinition.index>0</FAMIX.InheritanceDefinition.index>
    </FAMIX.InheritanceDefinition>
    <FAMIX.Method xmi.id="_6">
      <FAMIX.Entity.name>methodInInterfacB</FAMIX.Entity.name>
      <FAMIX.Entity.uniqueName>FAMIX::xmi::doctest::testdata::testInterfaceB.methodInInterfacB(int,java::lang::String)</FAMIX.Entity.uniqueName>
      <FAMIX.BehaviouralEntity.accessControlQualifier>public</FAMIX.BehaviouralEntity.accessControlQualifier>
      <FAMIX.BehaviouralEntity.signature>methodInInterfacB(int,java::lang::String)</FAMIX.BehaviouralEntity.signature>
      <FAMIX.BehaviouralEntity.isPureAccessor xmi.value="null"/>
      <FAMIX.BehaviouralEntity.declaredReturnType>java::lang::String</FAMIX.BehaviouralEntity.declaredReturnType>
      <FAMIX.BehaviouralEntity.declaredReturnClass>java::lang::String</FAMIX.BehaviouralEntity.declaredReturnClass>
      <FAMIX.Method.belongsToClass>FAMIX::xmi::doctest::testdata::testInterfaceB</FAMIX.Method.belongsToClass>
      <FAMIX.Method.hasClassScope xmi.value="false"/>
      <FAMIX.Method.isAbstract xmi.value="true"/>
      <FAMIX.Method.isConstructor xmi.value="false"/>
      <FAMIX.Method.isFinal xmi.value="false"/>
      <FAMIX.Method.isSynchronized xmi.value="false"/>
      <FAMIX.Method.isNative xmi.value="false"/>
    </FAMIX.Method>
    <FAMIX.Class xmi.id="_7">
      <FAMIX.Entity.name>XMITestClass</FAMIX.Entity.name>
      <FAMIX.Entity.uniqueName>FAMIX::xmi::doctest::testdata::XMITestClass</FAMIX.Entity.uniqueName>
      <FAMIX.Class.belongsToPackage>FAMIX::xmi::doctest::testdata</FAMIX.Class.belongsToPackage>
      <FAMIX.Class.isAbstract xmi.value="true"/>
    <FAMIX.Class.isInterface xmi.value="false"/>
    <FAMIX.Class.isPublic xmi.value="true"/>
    <FAMIX.Class.isFinal xmi.value="false"/>
  </FAMIX.Class>
  <FAMIX.InheritanceDefinition xmi.id="_8">
    <FAMIX.InheritanceDefinition.subclass>FAMIX::xmi::doctest::testdata::XMITestClass</FAMIX.InheritanceDefinition.subclass>
    <FAMIX.InheritanceDefinition.superclass>FAMIX::xmi::doctest::testdata::testInterfaceB</FAMIX.InheritanceDefinition.superclass>
    <FAMIX.InheritanceDefinition.index>0</FAMIX.InheritanceDefinition.index>
  </FAMIX.InheritanceDefinition>
  <FAMIX.InheritanceDefinition xmi.id="_9">
    <FAMIX.InheritanceDefinition.subclass>FAMIX::xmi::doctest::testdata::XMITestClass</FAMIX.InheritanceDefinition.subclass>
    <FAMIX.InheritanceDefinition.superclass>java::lang::Object</FAMIX.InheritanceDefinition.superclass>
    <FAMIX.InheritanceDefinition.index>1</FAMIX.InheritanceDefinition.index>
  </FAMIX.InheritanceDefinition>
  <FAMIX.Attribute xmi.id="_10">
    <FAMIX.Entity.name>attr1</FAMIX.Entity.name>
    <FAMIX.Entity.uniqueName>FAMIX::xmi::doctest::testdata::XMITestClass.attr1</FAMIX.Entity.uniqueName>
    <FAMIX.StructuralEntity.declaredType>int</FAMIX.StructuralEntity.declaredType>
    <FAMIX.Attribute.belongsToClass>FAMIX::xmi::doctest::testdata::XMITestClass</FAMIX.Attribute.belongsToClass>
    <FAMIX.Attribute.accessControlQualifier>public</FAMIX.Attribute.accessControlQualifier>
    <FAMIX.Attribute.hasClassScope xmi.value="true"/>
    <FAMIX.Attribute.isFinal xmi.value="true"/>
    <FAMIX.Attribute.isTransient xmi.value="false"/>
    <FAMIX.Attribute.isVolatile xmi.value="false"/>
  </FAMIX.Attribute>
  <FAMIX.Attribute xmi.id="_11">
    <FAMIX.Entity.name>attr2</FAMIX.Entity.name>
    <FAMIX.Entity.uniqueName>FAMIX::xmi::doctest::testdata::XMITestClass.attr2</FAMIX.Entity.uniqueName>
    <FAMIX.StructuralEntity.declaredType>java::lang::String</FAMIX.StructuralEntity.declaredType>
    <FAMIX.StructuralEntity.declaredClass>java::lang::String</FAMIX.StructuralEntity.declaredClass>
    <FAMIX.Attribute.belongsToClass>FAMIX::xmi::doctest::testdata::XMITestClass</FAMIX.Attribute.belongsToClass>
    <FAMIX.Attribute.accessControlQualifier>private</FAMIX.Attribute.accessControlQualifier>
    <FAMIX.Attribute.hasClassScope xmi.value="false"/>
    <FAMIX.Attribute.isFinal xmi.value="false"/>
    <FAMIX.Attribute.isTransient xmi.value="false"/>
    <FAMIX.Attribute.isVolatile xmi.value="true"/>
  </FAMIX.Attribute>
  <FAMIX.Attribute xmi.id="_12">
    <FAMIX.Entity.name>attr3</FAMIX.Entity.name>
    <FAMIX.Entity.uniqueName>FAMIX::xmi::doctest::testdata::XMITestClass.attr3</FAMIX.Entity.uniqueName>
    <FAMIX.StructuralEntity.declaredType>int</FAMIX.StructuralEntity.declaredType>
    <FAMIX.Attribute.belongsToClass>FAMIX::xmi::doctest::testdata::XMITestClass</FAMIX.Attribute.belongsToClass>
    <FAMIX.Attribute.accessControlQualifier>protected</FAMIX.Attribute.accessControlQualifier>
    <FAMIX.Attribute.hasClassScope xmi.value="false"/>
    <FAMIX.Attribute.isFinal xmi.value="false"/>
    <FAMIX.Attribute.isTransient xmi.value="true"/>
    <FAMIX.Attribute.isVolatile xmi.value="false"/>
  </FAMIX.Attribute>
  <FAMIX.Attribute xmi.id="_13">
    <FAMIX.Entity.name>attr_4</FAMIX.Entity.name>
```

```
      <FAMIX.Entity.uniqueName>FAMIX::xmi::doctest::testdata::XMITestClass.attr_4</FAMIX.Entity.uniqueName>
      <FAMIX.StructuralEntity.declaredType>int</FAMIX.StructuralEntity.declaredType>
      <FAMIX.Attribute.belongsToClass>FAMIX::xmi::doctest::testdata::XMITestClass</FAMIX.Attribute.belongsToClass>
      <FAMIX.Attribute.accessControlQualifier>private</FAMIX.Attribute.accessControlQualifier>
      <FAMIX.Attribute.hasClassScope xmi.value="false"/>
      <FAMIX.Attribute.isFinal xmi.value="false"/>
      <FAMIX.Attribute.isTransient xmi.value="false"/>
      <FAMIX.Attribute.isVolatile xmi.value="false"/>
    </FAMIX.Attribute>
    <FAMIX.Method xmi.id="_14">
      <FAMIX.Entity.name>XMITestClass</FAMIX.Entity.name>
      <FAMIX.Entity.uniqueName>FAMIX::xmi::doctest::testdata::XMITestClass.XMITestClass()</FAMIX.Entity.uniqueName>
      <FAMIX.BehaviouralEntity.accessControlQualifier>public</FAMIX.BehaviouralEntity.accessControlQualifier>
      <FAMIX.BehaviouralEntity.signature>XMITestClass()</FAMIX.BehaviouralEntity.signature>
      <FAMIX.BehaviouralEntity.isPureAccessor xmi.value="null"/>
      <FAMIX.BehaviouralEntity.declaredReturnType></FAMIX.BehaviouralEntity.declaredReturnType>
      <FAMIX.BehaviouralEntity.declaredReturnClass></FAMIX.BehaviouralEntity.declaredReturnClass>
      <FAMIX.Method.belongsToClass>FAMIX::xmi::doctest::testdata::XMITestClass</FAMIX.Method.belongsToClass>
      <FAMIX.Method.hasClassScope xmi.value="false"/>
      <FAMIX.Method.isAbstract xmi.value="false"/>
      <FAMIX.Method.isConstructor xmi.value="true"/>
      <FAMIX.Method.isFinal xmi.value="false"/>
      <FAMIX.Method.isSynchronized xmi.value="false"/>
      <FAMIX.Method.isNative xmi.value="false"/>
    </FAMIX.Method>
    <FAMIX.Method xmi.id="_15">
      <FAMIX.Entity.name>methodA</FAMIX.Entity.name>
      <FAMIX.Entity.uniqueName>FAMIX::xmi::doctest::testdata::XMITestClass.methodA()</FAMIX.Entity.uniqueName>
      <FAMIX.BehaviouralEntity.accessControlQualifier>private</FAMIX.BehaviouralEntity.accessControlQualifier>
      <FAMIX.BehaviouralEntity.signature>methodA()</FAMIX.BehaviouralEntity.signature>
      <FAMIX.BehaviouralEntity.isPureAccessor xmi.value="null"/>
      <FAMIX.BehaviouralEntity.declaredReturnType>void</FAMIX.BehaviouralEntity.declaredReturnType>
      <FAMIX.Method.belongsToClass>FAMIX::xmi::doctest::testdata::XMITestClass</FAMIX.Method.belongsToClass>
      <FAMIX.Method.hasClassScope xmi.value="false"/>
      <FAMIX.Method.isAbstract xmi.value="true"/>
      <FAMIX.Method.isConstructor xmi.value="false"/>
      <FAMIX.Method.isFinal xmi.value="false"/>
      <FAMIX.Method.isSynchronized xmi.value="true"/>
      <FAMIX.Method.isNative xmi.value="false"/>
    </FAMIX.Method>
    <FAMIX.Method xmi.id="_16">
      <FAMIX.Entity.name>methodInInterfacB</FAMIX.Entity.name>
      <FAMIX.Entity.uniqueName>FAMIX::xmi::doctest::testdata::XMITestClass.methodInInterfacB(int,java::lang::String)</FAMIX.Entity.uniqueName>
      <FAMIX.BehaviouralEntity.accessControlQualifier>public</FAMIX.BehaviouralEntity.accessControlQualifier>
      <FAMIX.BehaviouralEntity.signature>methodInInterfacB(int,java::lang::String)</FAMIX.BehaviouralEntity.signature>
      <FAMIX.BehaviouralEntity.isPureAccessor xmi.value="null"/>
      <FAMIX.BehaviouralEntity.declaredReturnType>java::lang::String</FAMIX.BehaviouralEntity.declaredReturnType>
      <FAMIX.BehaviouralEntity.declaredReturnClass>java::lang::String</FAMIX.BehaviouralEntity.declaredReturnClass>
      <FAMIX.Method.belongsToClass>FAMIX::xmi::doctest::testdata::XMITestClass</FAMIX.Method.belongsToClass>
      <FAMIX.Method.hasClassScope xmi.value="false"/>
      <FAMIX.Method.isAbstract xmi.value="false"/>
      <FAMIX.Method.isConstructor xmi.value="false"/>
      <FAMIX.Method.isFinal xmi.value="true"/>
      <FAMIX.Method.isSynchronized xmi.value="false"/>
      <FAMIX.Method.isNative xmi.value="false"/>
    </FAMIX.Method>
    <FAMIX.Method xmi.id="_17">
      <FAMIX.Entity.name>methodC</FAMIX.Entity.name>
      <FAMIX.Entity.uniqueName>FAMIX::xmi::doctest::testdata::XMITestClass.methodC()</FAMIX.Entity.uniqueName>
      <FAMIX.BehaviouralEntity.accessControlQualifier>private</FAMIX.BehaviouralEntity.accessControlQualifier>
      <FAMIX.BehaviouralEntity.signature>methodC()</FAMIX.BehaviouralEntity.signature>
      <FAMIX.BehaviouralEntity.isPureAccessor xmi.value="null"/>
      <FAMIX.BehaviouralEntity.declaredReturnType>void</FAMIX.BehaviouralEntity.declaredReturnType>
      <FAMIX.Method.belongsToClass>FAMIX::xmi::doctest::testdata::XMITestClass</FAMIX.Method.belongsToClass>
      <FAMIX.Method.hasClassScope xmi.value="true"/>
      <FAMIX.Method.isAbstract xmi.value="false"/>
      <FAMIX.Method.isConstructor xmi.value="false"/>
      <FAMIX.Method.isFinal xmi.value="false"/>
      <FAMIX.Method.isSynchronized xmi.value="false"/>
      <FAMIX.Method.isNative xmi.value="false"/>
    </FAMIX.Method>
    <FAMIX.Method xmi.id="_18">
      <FAMIX.Entity.name>methodInInterfacA</FAMIX.Entity.name>
      <FAMIX.Entity.uniqueName>FAMIX::xmi::doctest::testdata::XMITestClass.methodInInterfacA()</FAMIX.Entity.uniqueName>
      <FAMIX.BehaviouralEntity.accessControlQualifier>public</FAMIX.BehaviouralEntity.accessControlQualifier>
      <FAMIX.BehaviouralEntity.signature>methodInInterfacA()</FAMIX.BehaviouralEntity.signature>
      <FAMIX.BehaviouralEntity.isPureAccessor xmi.value="null"/>
      <FAMIX.BehaviouralEntity.declaredReturnType>void</FAMIX.BehaviouralEntity.declaredReturnType>
      <FAMIX.Method.belongsToClass>FAMIX::xmi::doctest::testdata::XMITestClass</FAMIX.Method.belongsToClass>
      <FAMIX.Method.hasClassScope xmi.value="false"/>
      <FAMIX.Method.isAbstract xmi.value="true"/>
      <FAMIX.Method.isConstructor xmi.value="false"/>
      <FAMIX.Method.isFinal xmi.value="false"/>
      <FAMIX.Method.isSynchronized xmi.value="false"/>
      <FAMIX.Method.isNative xmi.value="false"/>
    </FAMIX.Method>
  </XMI.content>
</XMI>
```

10 Appendix C : The MOF Model


AttachesTo
+model
+dependent
/DependsOn
+provider
ModelElement
+constrainedElement
Constrains
+containedElement
{ordered}
Contains
+container
Namespace
+constraint
Constraint
+tag
Tag
{Ordere
TypedElement
+typedElement
Import
+importer
Aliases
+imported
IsOfType
+supertype
Generalizes
{ordered}
+subtype
GeneralizableElement
Feature
TypeAlias
Constant
Parameter
+type
Package
Classifier
BehavioralFeature
StructuralFeature
+referencedEnd
AssociationEnd
RefersTo
+referent
+exposedEnd
Association
DataType
Class
<<MofAttribute>> Attribute
Reference
+referer
/Exposes
Operation
+operation
CanRaise
+except
<<MofException>> Exception
{ordered}