Generic XMI Support for the MOOSE Reengineering Environment

Andreas Schlapbach

June 17, 2001

Abstract

This report describes the implementation of generic XMI (XML Metadata Interchange) support for MOOSE, an Extensible Language-Independent Environment for Reengineering Object-Oriented Systems developed at the University of Berne.

Supervised by Dr. Stéphane Ducasse and Sander Tichelaar.
Software Composition Group

Institut für Informatik angewandte Mathematik
Universität Bern

Keywords
MOOSE, FAMIX, XMI, XML, DTD, MOF, SMALLTALK, SAX, Metamodel, UUID, XSL, XSLT, CSS

Contents

1 Introduction **5**

2 Background **9**
2.1 MOOSE and FAMIX . . . 9
2.1.1 MOOSE: Architecture . . . 9
2.1.2 FAMIX: A Language Independent Metamodel . . . 10
2.1.3 The FAMIX Model . . . 10
2.2 Information Exchange in FAMIX . . . 12
2.2.1 From CDIF to XMI . . . 12
2.2.2 Overview of XMI . . . 12
2.2.3 XMI in the Context of this Project . . . 14
2.3 Overview of XML . . . 14
2.3.1 Introduction . . . 14
2.3.2 XML Structure elements . . . 15
2.3.3 XML Example . . . 15
2.3.4 XML Attributes . . . 15
2.3.5 Document Type Definitions (DTDS) . . . 16
2.3.6 XML Document Correctness . . . 16
2.4 Overview of the MOF . . . 16
2.4.1 The MOF Model . . . 17
2.4.2 MOF Classes . . . 17
2.4.3 MOF Associations . . . 19
2.4.4 MOF Packages . . . 20
2.4.5 Other MOF Model Elements . . . 20

3 XMI for MOOSE **21**
3.1 Overview of the XMI Architecture of MOOSE . . . 21
3.2 Generic Class Descriptions . . . 22
3.2.1 The Need for a Generic Class Description . . . 22
3.2.2 Classes for Genericly Describing FAMIX Classes . . . 22
3.2.3 Example of a Generic Class Description . . . 22
3.2.4 Implementation of the Class Property in the FAMIX Metamodel . . . 24
3.2.5 Unique Identifiers for FAMIX Objects . . . 24
3.3 Saving a XMI File . . . 25
3.4 Creating and Saving DTD Files . . . 27
3.4.1 Implementing the MOF . . . 28
3.4.2 Creating an instance of a MOF out of a FAMIX model . . . 28
3.4.3 Creating a DTD from a MOF Instance . . . 28
3.4.4 Example of a FAMIX DTD . . . 30
3.5 Loading in XMI Files . . . 31
3.5.1 Overview of a FAMIX XMI File . . . 31
3.5.2 XML-/SAX-Parser for SMALLTALK . . . 32

3.5.3 Loading XMI files into MOOSE 32
3.6 Post processing XML Using XSL 33
3.6.1 From XML to HTML 34
3.6.2 Overview of XSL 34
3.6.3 Overview of CSS 35
3.6.4 XSL and CSS used for FAMIX 35

4 Conclusion and Further Work **39**
4.1 Conclusion 39
4.2 Further Work 40

A Example of a XMI File **41**

B FAMIX DTD **51**

Chapter 1

Introduction

MOOSE is an Extensible Language-Independent Environment for Reengineering Object-Oriented Systems [DLT00] developed at the University of Berne. MOOSE is written in SMALLTALK and its repository is based on the FAMIX metamodel which provides a language-independent representation of object-oriented sources and contains the required information for the reengineering tasks performed by various tools.

Before this project, CDIF was used as the underlying file format to exchange FAMIX-based information. The aim of this project was to add XMI support to MOOSE. The reason for changing from CDIF to XMI [1] lies largely in the fact that CDIF did not succeed in becoming a widely used standard. And the use of XMI offers two major advantages over CDIF: First, XMI is based on XML, allowing the use of XML based technologies (such as XSL described in section 3.6) and second, XMI is MOF based and therefore offers excellent integration to MOF based metamodels such as UML.

Therefore, in order to be able to save, transfer and load the information provided by FAMIX, the OMG standard XMI (XML Metadata Interchange [XMI98]) was introduced to map any metamodel to a XML DTD and generate XML files based on that metamodel. XMI is a standardised way to exchange models based on the MOF (Meta-Object Facility [Gro99]) and uses XML (Extensible Markup Language [BPSMM00]) as the underlying technology to save this information.

Using XML – likely to become the de facto standard for transferring information between applications – and MOF – likely to become the de facto standard to describe metamodels – assures that other tools are able of loading, processing, and storing the data provided by the MOOSE reengineering environment.

My contribution was to add XMI support to MOOSE. This mainly consisted of the following four parts (see figure 1.1):

1. XMI **Loader**
   Implementing a loader that is capable of reading in XML files and loading them into MOOSE.

2. XMI/DTD **Saver**
   This parts consists of two components:

   (a) XMI **Saver**
   Implementing a saver that is capable of saving the currently loaded model in a

[1] Michael Freidig has implemented a Java-based prototype for generating XMI documents based on the FAMIX metamodel using the Java Core Reflection API. See [Fre00] for more information.

XMI/XML conforming way.

(b) XMI DTD **Producer and Saver**
Implementing the architecture to create DTDS of the current metamodel. This implies transforming the metamodel information to a MOF compliant metamodel plus transforming this information into a valid DTD.

3. **MOF**
Because the XMI rules for producing a DTD are based on the MOF a basic MOF was implemented.

4. **Metametamodel Support**
To be able to change the FAMIX metamodel, a clean and extensible architecture for defining the FAMIX metamodel was introduced.

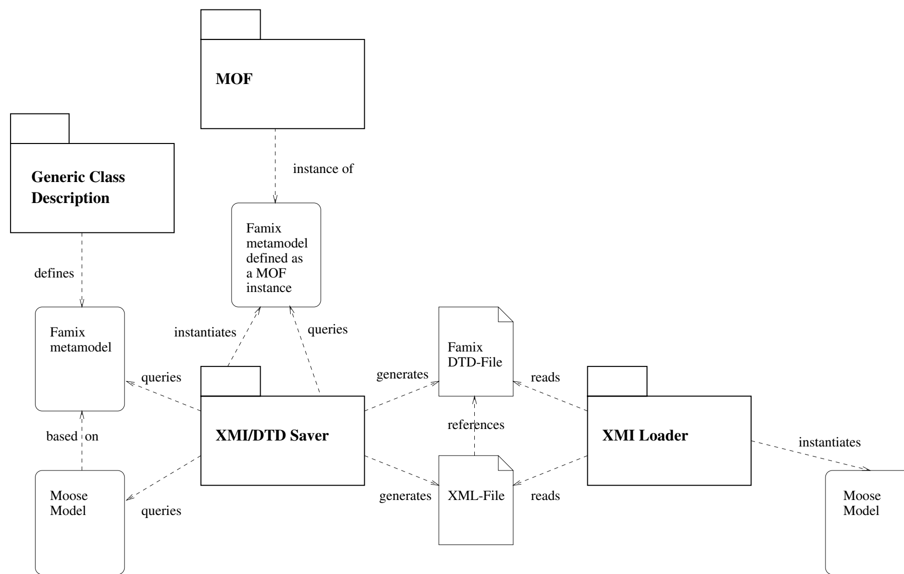

Figure 1.1: An overview of the XMI architecture of MOOSE

Additionally, XSL (Extensible Stylesheet Language [ABC+00]) and XSLT (XSL Transformations [Cla99]) were used to create an easy accessible and visually appealing HTML based representation of the model information saved in the XML file.

The structure of this report is the following: Chapter 2 introduces the background of this project: Section 2.1 gives an overview of the MOOSE architecture and introduces the FAMIX metamodel. Section 2.2 motivates our decision to use XMI instead of CDIF and gives a short introduction into XMI and how it was introduced to MOOSE. Section 2.3 and 2.4 features two major components of XMI: XML and MOF.

Chapter 3 gives an introduction of how XMI support for MOOSE was implemented: Section 3.2 describes the generic architecture for changes to the FAMIX metamodel and describes why and how FAMIX **Properties** and unique identifiers were introduced to FAMIX. The next two section describe the loading and storing of XMI files: Section 3.5 describes how loading XMI files is achieved, sections 3.3 and 3.4 describe how XMI and DTD files are created and saved. Finally, section 3.6 describes how XSL was used to transform XML data to HTML.

The report finishes with chapter 4 giving a conclusion of the project and suggestions for further work.

Chapter 2

Background

This chapter gives an overview of the MOOSE architecture and introduces the FAMIX metamodel. Our decision to use XMI instead of CDIF is motivated and the two major components of XMI– XML and MOF– are introduced.

2.1 MOOSE and FAMIX

2.1.1 MOOSE: Architecture

MOOSE is an Extensible Language-Independent Environment for Reengineering Object-Oriented Systems [DLT00] developed at the University of Berne. It is written in SMALLTALK and consists of a repository to store models of software systems, and provides query and navigation facilities. Models consist of entities representing software artifacts such as classes, methods, etc.

MOOSE uses a layered architecture (see Figure 2.1). Information is transformed from source code into a source code model. The models are based on the FAMIX metamodel which is described in section 2.1.2. The information in this model, in the form of entities representing the software artifacts of the target system, can be analysed, manipulated and used to trigger code transformations by means of refactorings.

- *Extraction/Import.* MOOSE supports multiple languages. Source code can be imported into the metamodel in two different ways:
  1. In the case of VisualWorks SMALLTALK – the language in which MOOSE is implemented – sources can be directly extracted via the metamodel of the SMALLTALK language.
  2. For other source languages MOOSE provides an import interface for CDIF files based on our FAMIX metamodel. CDIF ([Com94]) is an industry-standard interchange format which enables exchanging models via files or streams. Over this interface MOOSE uses external parsers for source languages other than SMALLTALK. Currently C++, JAVA, COBOL and other SMALLTALK dialects are supported.
- *Storage and Tools.* The models are stored in memory. Every model contains entities representing the software artifacts of the target system. Every entity is represented by an object, which allows direct interaction and querying of entities, and consequently an easy way to query and navigate a whole model. MOOSE can maintain and access several models in memory at the same time.

  Additionally the core of MOOSE contains the following functionality:

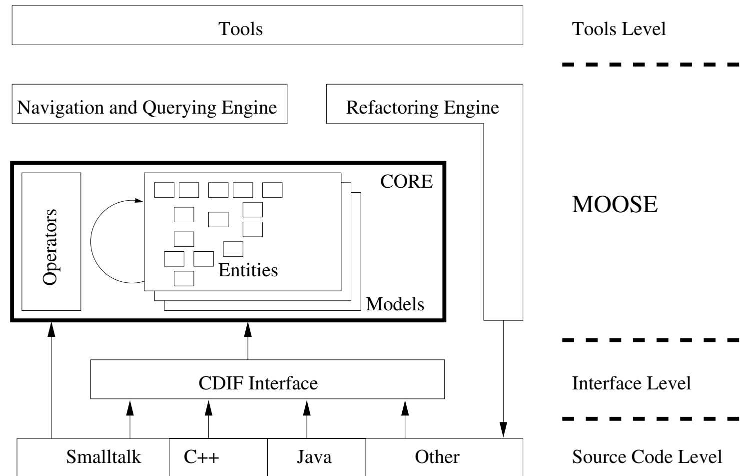

Figure 2.1: Architecture of MOOSE

- *Operators.* Operators can be run on a model to compute additional information regarding the software entities. For example, metrics can be computed and associated with the software entities, or entities can be annotated with additional information such as inferred type information, analysis of the polymorphic calls, etc. Basically any kind of information can be added to an entity.
- *Navigation facilities.* On top of the MOOSE core we have included querying and navigation support.
- *Refactoring Engine.* The MOOSE REFACTORING ENGINE defines language-independent refactorings. The analysis for a code refactoring is based on model information. The code manipulation which a refactoring entails, is being handled by language-specific front-ends.

- *Tools Layer.* The functionality which is provided by MOOSE can be used by tools. This is represented by the top layer of figure 2.1. Tools developed at the university of Berne are: CODE CRAWLER, GAUDI, MOOSE REVEALER, MOOSE FINDER, MOOSE DESIGN FILTER and others. For more information see [TDD00].

2.1.2 FAMIX: A Language Independent Metamodel

MOOSE is based on FAMIX, a language independent and extensible metamodel. FAMIX is *language independent*, because it needs to work with legacy systems in different implementation languages (C++, JAVA, SMALLTALK, ADA). And it is *extensible*: since not all information is known in advance that is needed in future tools, and since for some reengineering problems tools might need to work with language-specific information (e.g. to analyse include hierarchies in C++), language plug-ins are allowed that extend the model with language-specific features. Next to that, tool plug-ins allow to extend the model to store, for instance, analysis results or layout information for graphs. Figure 2.2 shows schematically the use of the FAMIX metamodel: the tools analysing the different languages and exchanging information with each other via FAMIX, possibly extended with language and tool plug-ins.

2.1.3 The FAMIX Model

This section describes the global structure of the FAMIX model. It introduces the core model (which illustrates the core entities and associations) and the abstract part of the

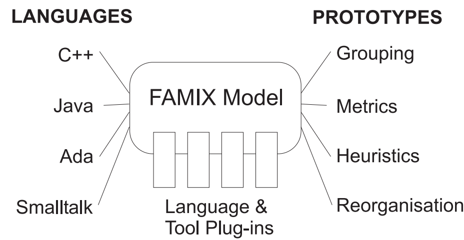

Figure 2.2: Concept of the FAMIX Model

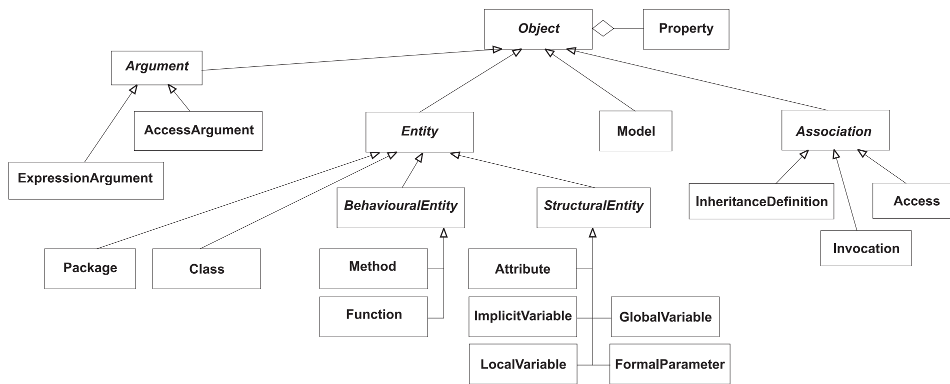

Figure 2.3: The complete FAMIX model

model (defining the abstract superclasses that will be extended). A figure of the complete FAMIX Model is shown in Figure 2.3. For more information, see [DTS99].

The Core Model

The core model (shown in Figure 2.4) specifies the entities and relations that can and should be extracted immediately from source code. The core model consists of the main OO entities, namely **Class**, **Method**, **Attribute** and **InheritanceDefinition**. For reengineering, we need the other two, the associations **Invocation** and **Access**. An **Invocation** represents the definition of a **Method** calling another **Method** and an **Access** represents a **Method** accessing an **Attribute**. These abstractions are needed for reengineering tasks such as dependency analysis, metrics computation and reengineering operations.

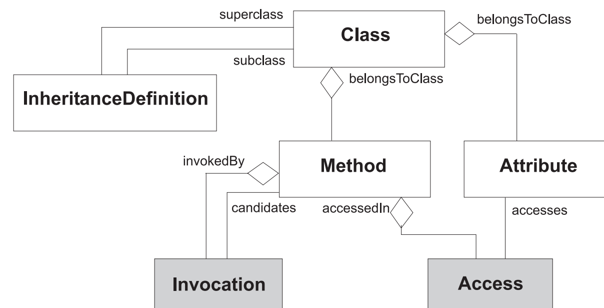

Figure 2.4: The core model

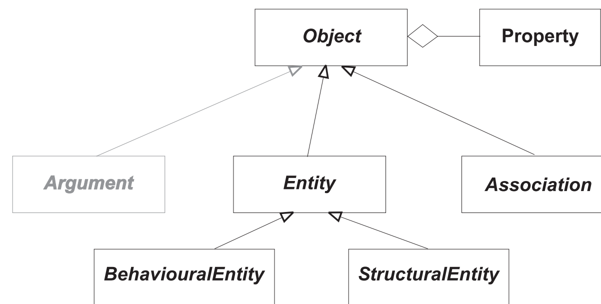

Figure 2.5: Abstract part of the model

The Abstract Part of the Model

The abstract part of the complete model is shown in Figure 2.5. **Object**, **Property**, **Entity** and **Association** are made available to handle the extensions to the model. For specifying language plug-ins, it is allowed to define language specific Objects, plus it is allowed to add language specific attributes to existing Objects. Tool prototypes are more restricted in extensions to the model: they can define tool specific **Properties** for existing Objects. Next to that, they can add attributes to existing Objects, but they cannot extend the repertoire of entities and associations. The abstract classes **StructuralEntity** and **BehaviouralEntity** are needed by the associations.

2.2 Information Exchange in FAMIX

2.2.1 From CDIF to XMI

Initially, to exchange FAMIX-based information between different tools CDIF [Com94] was adopted. CDIF is an industrial standard for transferring models created with different tools. The main reasons for adopting CDIF were, that firstly it is an industry standard, and secondly it has a standard plain text encoding which tackles the requirements of convenient querying and human readability (see [NTD98]).

The reason for changing from CDIF to XMI lies largely in the fact that CDIF did not succeed in becoming a widely used standard (e.g. not much tool support) and can be considered as being dead. Additionally the use of XMI offers two major advantages over CDIF:

1. XMI is based on XML. XML is gaining widespread acceptance as the de facto standard for representing structured information in the context of the world-wide web and beyond. This opens up a wide field for current and upcoming XML based technologies such as XSL (introduced in section 3.6).

2. XMI is MOF based and therefore offers excellent integration to MOF based metamodels such as UML. Especially in the the case of UML, there is a close relationship between the metamodeling concepts of MOF and the modeling concepts of UML. This allows the UML graphical notation to be used to express MOF metamodels. And as the UML metamodel is defined as a MOF metamodel, XMI is the obvious model interchange format for UML.

2.2.2 Overview of XMI

The main purpose of XMI is to enable easy interchange of metadata between modeling tools (based on the OMG UML) and metadata repositories (OMG MOF based) in dis-

tributed heterogeneous environments. XMI integrates three key industry standards:

- XML- Extensible Markup Language, a W3C standard
- UML- Unified Modeling Language, an OMG modeling standard
- MOF- Meta Object Facility, an OMG metamodeling and metadata repository standard

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

Key aspects of the architecture include:

- A four layered metamodeling architecture for general purpose manipulation of metadata in distributed object repositories.
- The use of MOF to define and manipulate metamodels programmatically using fine grained CORBA interfaces.
- The use of UML notation for representing models and metamodels.
- The use of standard information models (UML) to describe the semantics of object analysis and design models.
- The use of SMIF (the current XMI proposal) for stream based interchange of metadata.

The XMI specification mainly consists of:

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

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

2.2.3 XMI in the Context of this Project

While the XMI specification describes a rich infrastructure for metadata exchange, only the XML/DTD creation rules were of main interest for this project. The XML-based Metadata Interchange (XMI) proposal has two major components:

1. The XML DTD Production Rules for producing XML Document Type Definitions (DTDs). XMI DTDs serve as syntax specification for XMI documents, and allow generic XML tools to be used to compose and validate XMI documents. In chapter '*XMI DTD Production*' of [XMI98] rules how an instance of a MOF can be transformed into a DTD are described.

2. The XML Document Production Rules for encoding metadata into an XML compatible format. These production rules can be applied in reverse to decode XMI documents and reconstruct the metadata and are described in length in chapter '*XML Document Production*' of [XMI98].

Within the scope of this project, only the XMI DTD Production Rules were implemented. Together with the implementation of the relevant parts of the MOF, this implementation allows full reuse in other projects completely independent of the MOOSE infrastructure. It is the intent of the author to release this part of the project to the public in a later stage.

As there was already a working infrastructure for reading and saving data within MOOSE, the reading/saving operations for XMI are based on this infrastructure rather then on the rules described in chapter '*XML Document Production*'. This decision has the drawback that the XMI reading/saving is tightly coupled to MOOSE. It was mainly taken due to time concerns. Any reuse of this implementation in projects independent of MOOSE is close to impossible. As an implementation of these rules seems straight forward, a short discussion of how this could be done is given in section 4.2.

As XML and MOF are the two main technologies involved in the creation of DTDs, the next two sections give a short overview of them.

2.3 Overview of XML

2.3.1 Introduction

Extensible Markup Language, abbreviated XML, describes a class of data objects called XML documents and partially describes the behaviour of computer programs which process them. XML is an application profile or restricted form of SGML, the Standard Generalized Markup Language [ISO 8879]. By construction, XML documents are conforming SGML documents.

XML documents are made up of storage units called entities, which contain either parsed or unparsed data. Parsed data is made up of characters, some of which form character data, and some of which form markup. Markup encodes a description of the document's storage layout and logical structure. XML provides a mechanism to impose constraints on the storage layout and logical structure.

A software module called an XML processor is used to read XML documents and provide access to their content and structure. It is assumed that an XML processor is doing its work on behalf of another module, called the application. This specification describes the required behaviour of an XML processor in terms of how it must read XML data and the information it must provide to the application.

The following sections give a short overview of the structure of a XML file. For a full description see the XML W3C recommendation [BPSMM00].

2.3.2 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 (DTD).

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

```
<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.

2.3.3 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 in more detail: Make, Model, Year, Color, and Price. The content of each of the nested elements encodes a value in some agreed format.

2.3.4 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.

2.3.5 Document Type Definitions (DTDS)

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 a Car element 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 XML attribute called 'xmi.label' and that the 'xmi.label' consists of a character data string[1]:

```
<!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'.

2.3.6 XML Document Correctness

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

- A **well-formed** XML document is one where the elements are properly structured as a tree with the opening and closing tags correctly nested. Well-formed documents are essential for information exchange.
- A **valid** XML document is one which is well-formed and that conforms to the structure defined by a DTD. A valid document will only contain elements and attributes defined in the DTD. Similarly, the element contents and attribute values will conform to the DTD. While the DTD need not be specified in an XML document, and a consumer need not use the DTD when decoding the document, the DTD is essential for checking validity.
- The highest level of document correctness ("semantic correctness") is beyond the scope of XML and DTDS as they are currently defined. Only a XML document consumer with deep domain knowledge can check that the information in an XML document makes sense. In the Car example, this might include a check that a particular Color was available for a given combination of Make, Model, and Year.

2.4 Overview of the MOF

The OMG MOF is a generic framework for describing and representing meta-information in a CORBA-based environment. In this context, the term "meta-information" covers any information that describes other information. This is intended to include such things as:

- Mapping descriptions for interoperability tools; e.g. application level bridges,

[1] The #IMPLIED directive indicates that the attribute is optional.

- Metadata for databases and information retrieval systems,
- Models and project management information for software development tools,
- Interface definitions for CORBA objects, COM objects, DCE services and so on,
- Service types for the CORBA Trader.

The MOF is designed to support many different kinds of meta-information. This is achieved by treating the meta-information as information, and formally modelling each distinct kind of meta-information. These formal models are expressed using the metamodeling constructs provided by the MOF Model. Figure 2.6 gives an overview of the full structure of the MOF Model.

2.4.1 The MOF Model

The MOF Model is based on the concepts of entity relationship modelling. The three kinds of building blocks for a meta-information model are objects (described by MOF Classes), links that connect objects (described by MOF Associations), and data values (described by CORBA IDL types). Instances of these constructs are organised as MOF Packages.

2.4.2 MOF Classes

A MOF Class defines the type of an object rather than its implementation. The Class defines the type in the following terms:

Class Name: This is an identifier which obeys the CORBA syntax rules.

Attributes: A MOF object may have a number of Attributes. A MOF Attribute definition has the following components:

- The attribute's **name** which is an identifier which obeys the CORBA syntax rules.
- The attribute's **type** which is either another Class, or a CORBA IDL type.
- The attribute's **scope** which determines whether the attribute values are "instance-level" (i.e. each object has its own attribute value) or "class-wide" (i.e. the attribute values are shared by all object instances).
- The attribute's **multiplicity** which determines the number of attribute values that are allowed for a given object (or class), and whether the attribute values have ordering or uniqueness semantics.
- The attribute's **derivedness** which determines whether the attribute values are stored as part of the object state, or computed from other information.
- The attribute's **changeability** which determines whether the attribute value can be directly updated.

Operations: A MOF object can have a number of operations (in addition to any implicit operations for accessing and updating attribute values and association links). These are defined by the following components:

- The operation's **parameter list** which defines the name, type, direction and multiplicity for each parameter. Parameter types can be another Class or a CORBA data type.
- The operation's **result** which defines the type and multiplicity of the result.
- The operation's **exception** list which defines the exceptions that may be raised.

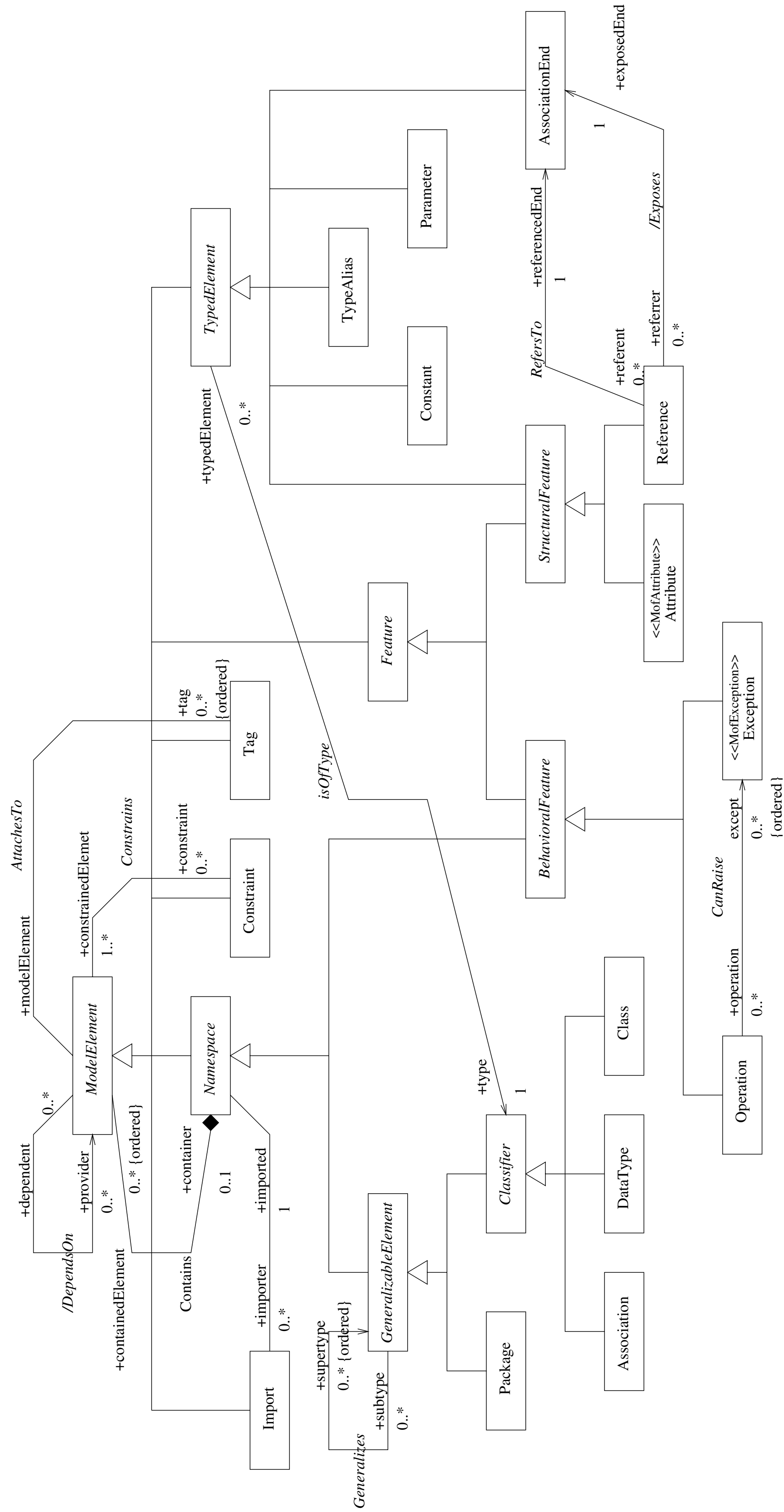

Figure 2.6: The MOF model

References: A MOF object may be defined to be "aware" of being in a relationship with other objects via an Association. This awareness is expressed as a Reference, and results in link navigation and update operations being made available in the MOF object's interface. A Reference has the following components:

- The reference's **name** is an identifier that conforms to the CORBA IDL identifier syntax rules.
- The **referenced Association End** determines what "ends" of what kinds of links an object is "aware" of.
- The reference's **changeability** determines whether the object has operations to update the referenced association.

Supertypes: A MOF Class may inherit from other MOF Classes. The inheritance model is analogous to CORBA interface inheritance; i.e. multiple inheritance with a "diamond" rule. Attributes, Operations and References may be inherited.

Abstract: An abstract MOF Class is defined so that other Classes may inherit from it. The MOF does not support creation of instances of an abstract Class.

Singleton: A singleton MOF Class is one for which only one instance may exist.

Contents: A MOF Class is a "container" for its component features; i.e. any Attributes, Operations and References. It may also contain MOF definitions of CORBA types (e.g. typedefs) and Exceptions.

2.4.3 MOF Associations

A MOF Association defines a class of links between MOF objects. Links are always binary and directed. An Association is defined by the following components

Association Ends: Each MOF Association has precisely two Ends with the following components:

1. The end's **name** is an identifier that satisfies the CORBA IDL syntax rules.
2. The end's **type** is a MOF Class.
3. The end's **multiplicity** constrains the number of links that may involve one object at one end. It also specifies whether the links have a partial order.
4. The end's **aggregation** determines whether the association defines a composite object or a "looser" connection between linked objects.

Derivedness: A MOF Association may be defined to be derivable from other information in a model. For example, one can define "substitutability" as a derived relationship between two instances of a given Class.

Changeability: A MOF Association may be defined so that links cannot be explicitly added or removed. This only makes sense for a derived association.

Contents: A MOF association is a "container" for its Association Ends.

There are a few restrictions on Associations. For example:

- Only one End of an Association can have aggregation of "composite" or "shared".
- The links of an Association are implicitly unique; i.e. an Association cannot have more than one link instance from one given object to another given object.
- If a Class has a Reference to an Association, there are restrictions on creating a link in one context involving objects in another one.

2.4.4 MOF Packages

The third construct of note in the MOF Model is the Package. A MOF Package serves as the unit of modularisation and reuse of meta-information models similar to UML Packages.

A MOF Package is defined by the following components:

Name: This is an identifier that satisfies the CORBA IDL syntax rules.

Contents: A MOF Package is a "container" for Classes, Associations and other Packages as well as MOF definitions of CORBA types and exceptions.

Imports: A MOF Package's imports list defines a set of other MOF Packages whose components may be reused by components defined within the Package.

Supertypes: A MOF Package's supertypes list defines a set of other MOF Packages whose components form a part of the Package.

MOF Packages provide three mechanisms that support modularisation and reuse of models and model components.

- A Package may contain nested sub-Packages. Nesting one Package allows the designer to use scoping rules to impose some structure on the namespace of a large Package.
- A Package may import other Packages. When one Package imports another one, the components in the imported Package are available to be used in the importing Package. Conversely, the components of a Package that is not imported are not visible.
- A Package may inherit from other Packages. When one Package inherits from another one, all of the components of the inherited Package become part of the inheriting Package.

A number of instances of a Package may exist, each with their own collections of Class instances and links. When a Class that is not "aware" that it may participate in an Association (i.e. if it has no Reference to an End of the Association), links may be freely created between Class instances in different schemas. However when the Class is "aware", links between different schemas are restricted.

2.4.5 Other MOF Model Elements

The MOF Model also includes the following "secondary" elements.

Data Types: Non-trivial MOF-based metamodels need to use data types for Attributes and Operation parameters. Basic data types such as CORBA's "integers", "characters" and "strings" are indespensible, and the ability to define constructed types is useful. The MOF Data Type model element supports all primitive and constructed CORBA data types, and also allows a metamodel to use CORBA object references.

Constants: The MOF Model allows a metamodel to define compile time constants that map onto CORBA constants.

Exceptions: The MOF Model requires exceptions raised by operations to be declared; c.f. CORBA exceptions.

Constraints: The MOF Model allows user-defined constraints to be attached to most kinds of MOF Model element. Constraints can be expressed in any language, though there is no requirement on a vendor to automate constraint checking.

Chapter 3

XMI for MOOSE

3.1 Overview of the XMI Architecture of MOOSE

The goal of this project was to introduce XMI support to MOOSE. Figure 3.1 gives an overview of the XMI architecture.

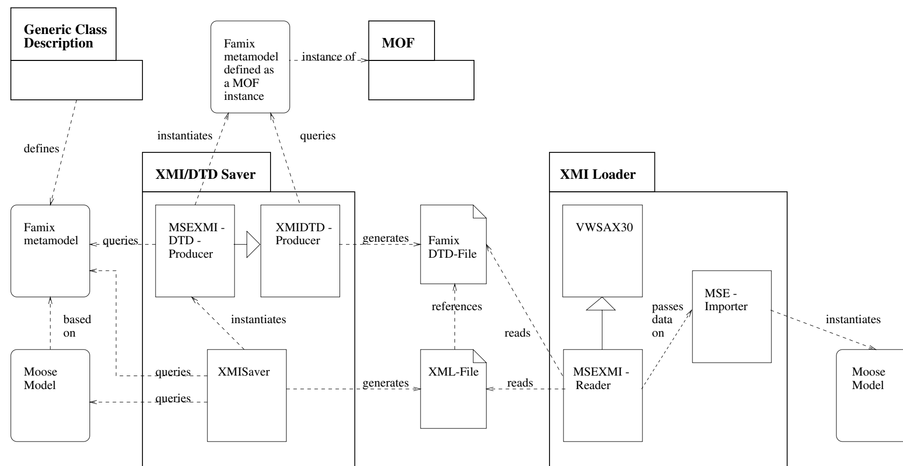

Figure 3.1: The XMI architecture of MOOSE

Within MOOSE, the FAMIX metamodel is described using the classes provided by the package **Generic Class Description**. In that sense this package is the metametamodel of MOOSE, offering a mechanism for describing different metamodels in MOOSE.

The **XMI Saver** package is responsible for saving XMI files as well as creating and saving a corresponding DTD file. The MOOSE model data are saved using the **MSEXMISaver** which creates a XMI file based on the FAMIX metamodel. The **MSEXMIDTDProducer** class is responsible for creating a MOF based representation of the FAMIX metamodel. This instance of a MOF in turn is saved using the **XMIDTDProducer**.

Responsible for loading XMI files is the package **XMI Loader**: The **MSEXMIReader** as a subclass of the VISUALWORKS's **VWSAX30** SAX driver reads in the XMI-file as well as its corresponding DTD file and passes the information describing the FAMIX objects on to the **MSEImporter** class which instantiates a MOOSE model based on this data.

The next four sections give a short introduction of how XMI support for MOOSE was implemented: Section 3.2 describes how the FAMIX metamodel can be defined using the **Generic Class Description** package. Sections 3.3 and 3.4 describe how XMI and DTD files are created and saved. The last section 3.5 describes how loading XMI files is achieved.

3.2 Generic Class Descriptions

3.2.1 The Need for a Generic Class Description

In order to be able to load and save FAMIX entities, the system must know which information to store and how to load it. The saver must know which information has to be stored in which way and the loader must know how to instantiate new objects and how to set their properties. Frankly speaken, the system must know the metamodel it is using.

Before this project, information describing the metamodel was spread over several functions (on class side as well as on the instance side) for each FAMIX entities. There was no inheriting of information from base classes to child classes, all the information was available in the leaf classes only. Every time new information was added to the model, it had to be added by hand to various methods. As there was redundant information in several methods, this lead to inconsistencies, causing subtle bugs which were hard to track down.

The aim of the architecture described below was to get rid of this mess and provide a clean way to add/change the FAMIX metamodel. In that sense, this architecture is the metamodel of FAMIX, i.e. a metametamodel.

3.2.2 Classes for Genericly Describing FAMIX Classes

Every child class of **MSEAbstractModelRoot** has a class side method called **initializeClassDescription**. This method initialises a data structure, describing this class: In the simple case that the class is *not* a FAMIX class it solely consists of this information. Otherwise this data structure describes this FAMIX class with all its attributes.

Before every loading/saving operation this method gets recursively called on every child class of **MSEAbstractModelRoot**. This assures that the information are always up to date.

The three classes describing FAMIX objects for load/save operations are: **MSEModelClassDescriptor** is used to describe a class, **MSEModelAttributeDescriptor** is used to define single value attributes and **MSEModelMVAttributeDescriptor** which is used to define attributes with can consist of multi values like comments. Their interface is described in the Tables 3.1, 3.2 and 3.3.

3.2.3 Example of a Generic Class Description

The following figure shows the **classDescription** for the FAMIX class **MSEClass**:

```
initializeClassDescription
  "self initializeClassDescription"

  classDescription := superclass classDescription copy.
  classDescription
    isFamixClass: true;
    famixClassName: #Class;
    addAttribute: ((MSEModelAttributeDescriptor new)
                  name: #isAbstract;
                  loadWithMethod: #isAbstract:;
                  typeCode: MSEModelAttributeDescriptor famixBoolean;
```

MSEModelClassDescriptor	
isFamixClass: aBoolean	Defines whether this class is a FAMIX class
isFamixClass	Accessor function
className: aString	Name of the FAMIX class
className	Accessor function
addAttribute: aMSEModelAttribute-Descriptor	add a MSEModelAttributeDescriptor to **MSEModelClass**
setAttributes: MSEModelAttribute-Descriptors	add a collection of MSEModelAttribute-Descriptors to **MSEModelClass**
attributes	Accessor function

Table 3.1: **MSEModelClassDescriptor** class

MSEModelAttributeDescriptor	
isDerived: aBoolean	Needed for the DTD generation, always set it to *false*.
isDerived	Accessor function
isMultiValue	returns *false*
loadWithMethod: aSymbol	Method to call to set the attribute.
loadWithMethod	Accessor function
multiplicity: aSymbol	Multiplicity of the attribute. Possible values: *oneTonN* | *oneToOne* | *zeroToN* | *zeroToOne*
multiplicity	Accessor function
name:	Name of the attribute
name	Accessor function
scope:	Scope of the attribute. Possible values: *classifierLevel* | *instanceLevel* Needed for the DTD generation, always set it to *instanceLevel*.
scope	Accessor function
typeCode: aSymbol	Type of attribute in FAMIX. Possible values: *famixBoolean* | *famixIdentifier* | *famixIndex* *famixName* | *famixQualifier* | *famixString*
typeCode	Accessor function

Table 3.2: **MSEModelAttributeDescriptor** class

MSEModelMVAttributeDescriptor	inherits of **MSEModelAttributeDescriptor**
isMultiValue	returns *true*

Table 3.3: **MSEModelMVAttributeDescriptor** class

```
                scope: MSEModelAttributeDescriptor instanceLevel;
                multiplicity: MSEModelAttributeDescriptor zeroToOne;
                isDerived: false;
yourself);
addAttribute: ((MSEModelMVAttributeDescriptor new)
                name: #interfaceSignatures;
                loadWithMethod: #addInterfaceSignature:;
                typeCode: MSEModelAttributeDescriptor famixName;
                scope: MSEModelAttributeDescriptor instanceLevel;
                multiplicity: MSEModelAttributeDescriptor zeroToN;
                isDerived: false;
yourself);
```

It is easy to see that this **classDescription** defines the FAMIX class **Class** which defines the two attributes **isAbstract** and **interfaceSignature**.

3.2.4 Implementation of the Class Property in the FAMIX Metamodel

In the process of refactoring software, various tools are being used to analyse various languages. These tools (such as SNiFF+) tend to gather information that is not described in our generic model (e.g. the keyword **synchronized** in Java or stubs into source code files), but is interesting to save. Therefore **Properties** (see figure 3.2) are introduced, offering a way in the metamodel to store additional properties for any object.

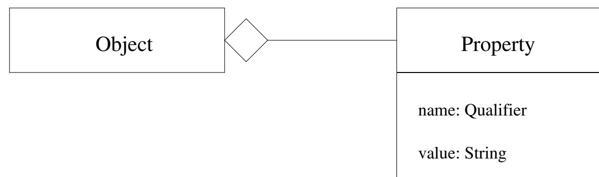

Figure 3.2: The FAMIX class **Property**

While the class **Property** was already defined in the metamodel it was never actually implemented. The reason for this lies in the CDIF exchange format: CDIF allows additional attributes of an entity to be saved in a CDIF file without violating the CDIF exchange format. Contrary to this, the DTD of a XML/XMI file rigorously defines the possible attributes of entities. Adding additional attributes to a XML file which are not defined in the DTD violates the validity of the XML file, causing validating parsers to report an error. Therefore, **Properties** had to be implemented as a clean way to store additional information.

A **Property** has the following attributes:

- **name: Qualifier, multiplicity: 1..1**
  name is a string that identifies a **Property** within an Object. Thus, the name should be unique for all properties of a single Object.
- **value: String, multiplicity: 1..1**
  value contains the value of the property. The meaning of the value is not defined within this model.

As there were no unique identifiers for all the objects in the metamodel they had to be introduced. This is described in the next subsection.

3.2.5 Unique Identifiers for FAMIX Objects

Properties as well as **Measurements** need to have unique identifiers pointing to the objects they belong to. While subclasses of ***Entity*** are uniquely described by their **uniqueName**,

all the other objects are lacking such a feature.

For these objects, Universal Unique Identifiers (UUIDS) as specified in [Gro97] have been introduced. A UUID is an identifier that is unique across both space and time, with respect to the space of all UUIDS. A UUID can be used for multiple purposes, from tagging objects with an extremely short lifetime, to reliably identifying very persistent objects across a network. UUIDS are generated using a combination of the network address and current time at the moment and place it was generated. Assuming that network addresses are unique and that time never runs backwards, this guarantees that UUIDS really are unique.

Calling **newId** on the class side of the class **MSEUUIDGenerator** will return a UUID approximately looking like this:

```
c842bf06-d202-0000-0282-5c410d000000
```

Adding this UUID to a newly created object assures that this object can be uniquely identified.

Unfortunately, having UUIDS for all objects increases the memory consumption of a model. As subclasses of ***Entity*** already have unique identifiers, the following policy regarding unique identifiers is chosen: If a class is a subclass of ***Entity*** its **uniqueName** is used, otherwise a UUID is created and serves as its unique identifier. (Some ideas how to reduce memory consumption of unique identifiers are given in section 4.2.)

3.3 Saving a XMI File

The **XMLSaver** is a subclass of **MSEAbstractSaver** and does all of the writing out of the currently loaded model. The main procedure is straightforward[1]: For every object in the model, stream it and its attributes out according to the XML specifications.

However there are several things to be considered:

- Two Different Types of XML Tags.
  Most of the attributes are stored by embedding their value in a pair of start/end element tags, looking like:

```
<FAMIX.Class xmi.id="c775cbaf-99de-0000-0282-5c410d000000">
  <FAMIX.Entity.name>AbstractDestinationAddress</FAMIX.Entity.name>
  <FAMIX.Entity.uniqueName>AbstractDestinationAddress</FAMIX.Entity.uniqueName>
   ..
</FAMIX.Class>
```

  Compared to this way of saving values, unique identifiers (as described in section 3.2.5) and boolean values are stored as attributes within the element, in the form e.g.:

```
<FAMIX.Class xmi.id="c775cbaf-99de-0000-0282-5c410d000000">
  ..
  <FAMIX.Class.isAbstract xmi.value="true"/>
  ..
</FAMIX.Class>
```

[1] As a matter of fact, considerable time had to be spent to get the MOOSE IO infrastructure back in shape. Time had left its mark on the code base, little hacks everywhere and confusing documentation made a refactoring necessary. Especially extensive use of inject..into and calling perform on concatenated strings made life hard. In the process of cleaning up, the MOOSE IO infrastructure was therefore partially rewritten.

- Encoding Data Corresponding to the XML specifications.
The XML specification requires the characters ’&’, ’<’, ’>’, ”’ and ’"’ do be encoded in a special way. **MSESingleValueToXMLConverter** takes care of this.

- Calculate Measurements and/or Properties.
Mainly out of memory concerns, instances of FAMIX class **Measurement** and **Properties** are not stored in the model as objects. If this information is to be stored too, then due to the fact that these objects are to be stored as explicit objects in the XMI-file, they have to be created at save time and passed on to the saver.

- Finding the Defining FAMIX Class of a FAMIX Attribute.
Saving an object which inherits variables from parent classes requires special treatment: In order to create valid XML files, i.e. compliant to the DTD, it is important to determine the (parent) class which defines the attribute. (See subsection 3.4.4 for an example).

  As the information which (parent) class defines the attribute is not available directly at runtime, a sort of attribute lookup had to be implemented: For every FAMIX attribute, the FAMIX class defining this variable has to be determined. Figure 3.3 shows

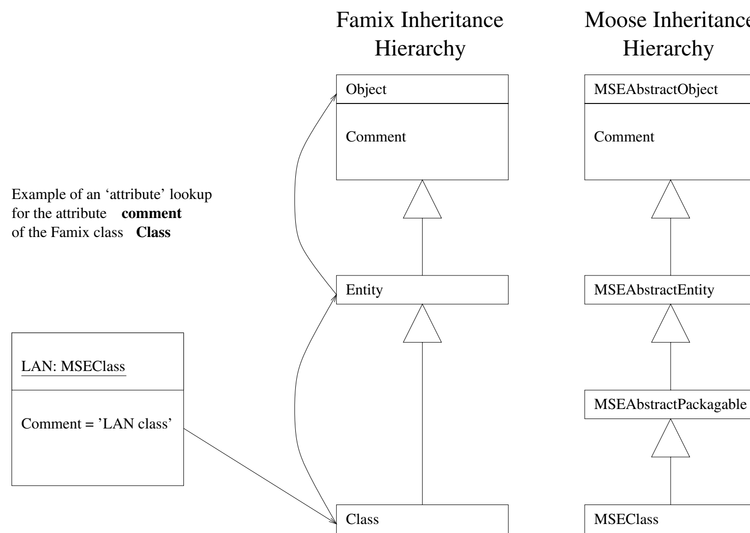

Figure 3.3: ‘Attribute’ lookup

  a schematic overview of the lookup procedure: As FAMIX information is stored on the class side of an object, the lookup starts on the class side of the current object and checks whether this variable is defined as an attribute of this FAMIX class. If the attribute is not defined in this class, the lookup goes up to the FAMIX super class. Here again it is checked whether this variable is defined as being part of this FAMIX class. This procedure is repeated until either the FAMIX class defining this attribute is found or the FAMIX root class is reached. (As there *has* to be a FAMIX class defining this variable, such an event is unlikely to occur.)

  It’s important to note that the FAMIX class hierarchy and the actual MOOSE class hierarchy are not identical. This is largely due to implementation decisions: Introducing further classes as described in FAMIX makes the design cleaner and more extensible. For example (as shown in figure 3.3) in the MOOSE implementation, there is a class **MSEAbstractPackagable** between **MSEClass** (representing **Class** in FAMIX) and **MSEAbstractEntity** (representing **Entity** in FAMIX).

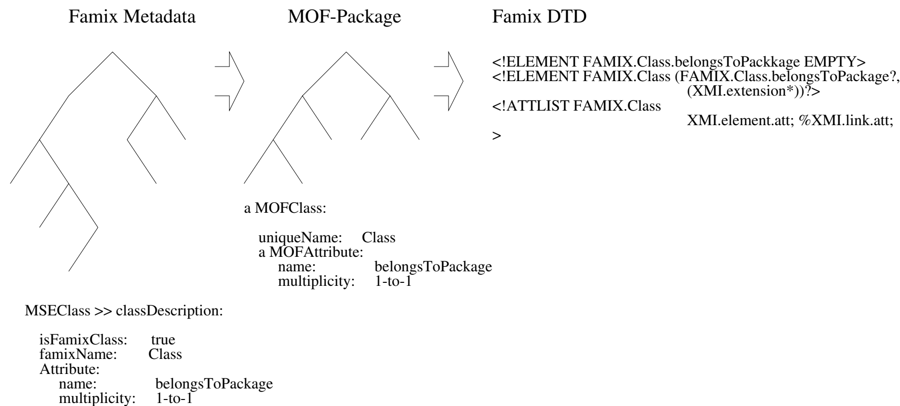

Figure 3.4: Transforming the FAMIX metadata to a FAMIX DTD

This decoupling of the FAMIX class hierarchy from the implementation complicates the lookup. Therefore a method called **famixParent** was implemented, which behaves the same on the FAMIX class hierarchy as **parent** does on the SMALLTALK class hierarchy.

- Treating Multiple Values.
  Contrary to the specification for saving multiple values in CDIF, no special care has to be taken in XML to store multiple values. If there are multiple values within the model, multiple XML entries with the same tag are written to the file. (The entries in the DTD file describe whether multiple XML entries are allowed for a given object. See subsection 3.4.4 for details.)

3.4 Creating and Saving DTD Files

The basic approach for creating and saving a DTD file consists of two steps (see figure 3.4):

1. Creating a MOF Package out of the MOOSE data.

2. Passing the MOF Package to the generic **XMIDTDProducer**, which will generate and stream out a DTD file as specified in [XMI98].

The first step includes transforming the MOOSE data into an instance of a MOF Package. As there was no SMALLTALK MOF implementation, the necessary parts had to be implemented. The next two subsections describe the implementation of the MOF and how it was used to create a MOF instance of the MOOSE model.

The second step is guided by the chapter '*XMI DTD Production*' of the XMI specification. There, rules for generating a valid DTD out of a MOF model are defined. This SMALLTALK implementation of the XMI DTD Production Rules follows closely the pseudo code given for every rule. Therefore only a short overview of how creating a DTD is achieved is given in the last subsection. Interested readers are asked to take a look at the XMI document which describes this procedure[2] in length.

[2] Even tough buggy

3.4.1 Implementing the MOF

Because the XMI rules for producing a DTD are based on the MOF and there was no MOF implementation, the relevant classes of the MOF had to be implemented. In this context, relevant means that these MOF classes are mentioned in the rules of the XMI specification, i.e. needed for the rules to work. Figures 3.5 shows the parts of the MOF implemented (Classes grayed out are *not* implemented). Due to the fact that SMALLTALK does not support multiple inheritance, some properties had to be copied to classes inheriting from multiple parents.

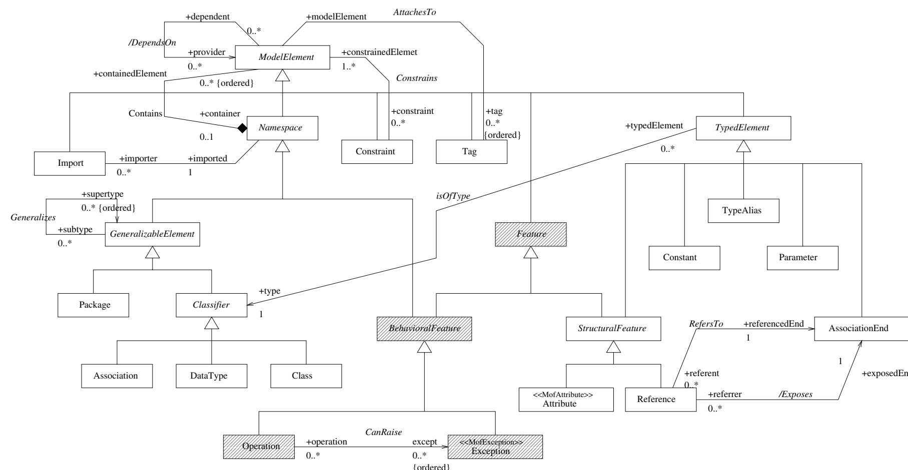

Figure 3.5: Implemented parts of the MOF model (Classes grayed out are *not* implemented)

3.4.2 Creating an instance of a MOF out of a FAMIX model

Producing a MOF means recursively traversing all subclasses of **MSEAbstractModelRoot** and creating a MOF Package from it. For every FAMIX class a MOF Class has to be created and for every attribute of this FAMIX class a MOF Attribute has to be created and added to the MOF Class. Furthermore, references to all the FAMIX superclasses of the newly created MOF Class have to be set. Creating MOF Attributes also implies transforming the MOOSE data types (as described in Table 3.2 or in [DTS99]) to corresponding types of the MOF (which in fact are CORBA data types). Again, the whole procedure is complicated by the fact, that the inheritance hierarchy of the FAMIX model is not equivalent to the inheritance hierarchy of the MOOSE implementation.

On the implementation side, **MSEXMIDTDProducer** is responsible for creating a MOF Package out of the currently loaded model in MOOSE. Within this class, the dictionary **FamixDataTypeToCorbaDataTypesDictionary** is responsible for the transformation of FAMIX data types to MOF data types.

3.4.3 Creating a DTD from a MOF Instance

After a MOF Package has been created of the current MOOSE model, this MOF Package is passed on to the class **XMIDTDProducer** which the opens a file and writes out the DTD according to the DTD production rules.

To give the reader an impression of how this is done, the two main rules describing the production of a PackageDTD and the ClassDTD are given here. Interested readers should con-

sult chapter '*XMI DTD Production*' of [XMI98] for a in-depth explanation of how DTDS are produced of a MOF Package. In fact, the XMI specification defines three set of rules for generating a DTD. Only the simplest rule set called '*Rule set 1: Simple DTD*' was implemented, with the drawback of producing more verbose DTDS than necessary. As the size of the DTDS is only of concern when being transferred over the web, this decision is legitimate.

A complete XMI DTD consists of fixed DTD content which is required for any XMI DTD, followed by at least one set of Package DTD elements. The XMI element, defined in this fixed content, is the XML document root type for a valid XMI document.

A PackageDTD is a sequence of DTD elements of various types, reflecting the contents of the Package. It includes DTD elements describing the Packages and Classes contained in the Package as well as DTD elements for Classifier-level Attributes of the Classes contained in the Package and for the References to compositions made by the Classes of the Package. The rather unusual case of an Association with no References is also handled at the Package level.

The algorithm to generate a PackageDTD is[3]:

```
For Each Class of the Package Do
  For each Attribute of the Class Do
    If isDerived is false Then
      If the scope of the Attribute is classifierLevel Then
        Generate an AttributeElementDef (#4) for the Attribute
      End
    End
  End
End
For Each Association of the Package Do
  If isDerived is false Then
    If the Association contains an AssociationEnd whose aggregation is
    composite Then
      Generate the CompositionDTD (#7) for the Association
    Else If the Association has no References Then
      Generate the AssociationDTD (#10) for the Association
    End
  End
End
For Each Class of the Package Do
  Generate the ClassDTD (#3) for the Class
End
For Each (sub) Package of the Package Do
  Generate the PackageDTD (#2) for the (sub) Package
End
Generate the PackageElementDef (#9) for the Package
```

Of the various rules mentioned in the pseudo code above, the rule for creating a ClassDTD is chosen here, because it shows nicely the top-down approach of the transformation procedure.

A ClassDTD is a set of DTD fragments that describes the contents of a Class. These fragments include element definitions for the instance-scope Attributes of the Class and for its non-composition References. The Classifier-scope Attributes of the Class are defined at the level of the Package that contains the Class, as are the composition References which are included in the Class.

The algorithm to generate a ClassDTD is:

```
For Each Attribute of the Class Do
```

[3] The number in brackets are references to rules describing how this part is achieved.

```
  If isDerived is false Then
    If scope is instanceLevel then
      Generate the AttributeElementDef (#4) for the Attribute
    End
  End
End
For Each Reference of the Class Do
  If the isDerived attribute of the associated Association is false Then
    If the the aggregation of the AssociationEnd which is the exposedEnd of the
    Reference is not composite Then
      Generate the ReferenceElementDef (#5) for the Reference
    End
  End
End
Generate the ClassElementDef (#6) for the Class
```

Overall '*Rule set 1: Simple DTD*' defines 12 rules plus auxiliary functions to transform a MOF Package to a valid DTD. Two things should be mentioned here: First, there is again a transforming of data types, this time by mapping the CORBA data types to XMI data types. Second, apart from the declaration of the XML elements and their attributes, the multiplicity of the elements is encoded in the DTD too. The next subsection gives an example of a DTD produced by this procedure.

3.4.4 Example of a FAMIX DTD

Here is a little part of the FAMIX DTD, defining the entries for the FAMIX class **Method**:

```
<!-- ________________________________________________ -->
<!--                                                  -->
<!-- META_MODEL CLASS: FAMIX.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 (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?,
                        (XMI.extension*))?>
<!ATTLIST FAMIX.Method
                        %XMI.element.att; %XMI.link.att;
>
```

The DTD snippet[4] given above defines elements for the FAMIX class **Method**: The first four element type declarations define four elements, relating to the following **Method** class properties: **belongsToClass**, **hasClassScope**, **isAbstract**, and **isConstructor**. For each of these elements the attribute-list declaration specifies the possible values for these elements: While the first element can be of data type *#PCDATA*[5] or *XMI.reference*, the other three *attribute-list declarations* specify explicitely the values accepted: *true*, *false*, or *null*.

The last element type declaration defines possible elements of the **Method** class itself: The **FAMIX.Method** element can include fourteen elements: The elements defined by the class **Method** plus the inherited elements and additionally a **XMI.extension**, a construct for extensions in XMI.

The optional character following a name governs whether the element or the content particles in the list may occur one or more (+), zero or more (*), or zero or one times (?). The absence of such an operator means that the element or content particle must appear exactly once. So while the element **FAMIX.Method.belongsToClass** is mandatory, the element **FAMIX.Method.hasClassScope** is optional.

It is important to note that order is crucial here: If the elements in the XMI file violate the order defined in the DTD, the XMI file will not pass as containing valid XML.

This DTD snippet also explains why the ‘attribute’ lookup described in subsection 3.3 had to be implemented: For every FAMIX attribute, the FAMIX class defining it is listed in the DTD.

3.5 Loading in XMI Files

3.5.1 Overview of a FAMIX XMI File

Due to the inheritance hierarchy of the FAMIX metamodel, FAMIX’s XMI files have a simple, tree-like structure: A FAMIX XMI file consists of a set of FAMIX objects plus the attributes of these objects. Every FAMIX object has an identifier assigned to it (see subsection 3.2.5 for more information about unique identifiers in FAMIX).

A snippet of a XMI file looks like this:

```
<FAMIX.Class xmi.id="AbstractDestinationAddress">
  <FAMIX.Entity.name>AbstractDestinationAddress</FAMIX.Entity.name>
  <FAMIX.Entity.uniqueName>AbstractDestinationAddress</FAMIX.Entity.uniqueName>
  <FAMIX.Class.isAbstract xmi.value="true"/>
</FAMIX.Class>

<FAMIX.InheritanceDefinition xmi.id="c7cffeec-d537-0000-0282-5c410d000000">
  <FAMIX.InheritanceDefinition.subclass>[6]
     AbstractDestinationAddress
  </FAMIX.InheritanceDefinition.subclass>
  <FAMIX.InheritanceDefinition.superclass>
    Object
  </FAMIX.InheritanceDefinition.superclass>
</FAMIX.InheritanceDefinition>

<FAMIX.Method xmi.id="AbstractDestinationAddress.isDestinationFor:(Object)">
  <FAMIX.Entity.name>isDestinationFor:</FAMIX.Entity.name>
  <FAMIX.Entity.uniqueName>
    AbstractDestinationAddress.isDestinationFor:(Object)
  </FAMIX.Entity.uniqueName>
```

[4] A complete DTD file of the FAMIX metamodel is available here: http://www.iam.unibe.ch/~schlpbch/-XMIforMoose

[5] The keyword #PCDATA derives from the term "parsed character data."

[6] Line breaks inserted by hand.

```
  <FAMIX.BehaviouralEntity.accessControlQualifier>
    public
  </FAMIX.BehaviouralEntity.accessControlQualifier>
  <FAMIX.BehaviouralEntity.signature>
    isDestinationFor:(Object)
  </FAMIX.BehaviouralEntity.signature>
  <FAMIX.Method.belongsToClass>
    AbstractDestinationAddress
  </FAMIX.Method.belongsToClass>
  <FAMIX.Method.hasClassScope xmi.value="false"/>
  <FAMIX.Method.isAbstract xmi.value="true"/>
  <FAMIX.Method.isConstructor xmi.value="false"/>
</FAMIX.Method>
```

The objects and their attributes in the file correspond directly to instances of FAMIX objects. It is easy to see that the XMI code snippet[7] above describes an abstract class called **AbstractDestinationAddress**, inheriting from a class called **Object**. Furthermore the class **AbstractDestinationAddress** has the method **isDestinationFor:(Object)** assigned to it.

3.5.2 XML-/SAX-Parser for SMALLTALK

XML-Parsers work by building up a Document Object Model (DOM) tree of the read XML/XMI file. The DOM protocol converts a XML document into a collection of objects in memory. These objects can then be manipulated in various ways, e.g. data can be modified, removed or inserted. This mechanism is also known as the "random access" protocol, because any part of the data can be visited at any time.

A disadvantage of DOM trees is that they can become very big and thus use up a lot of memory. And in our case, the random accessibility provided by DOM trees is not needed: Thanks to the simple, tree-like structure, FAMIX's XMI files can be streamed in and instantiated sequentially in one pass.

Therefore the DOM tree approach was not taken. Instead VISUALWORKS free **VWSAX-30**[8] validating[9] SAX driver was used. SAX (Simple API for XML) is an event-driven, serial-access mechanism for accessing XML documents which is faster and less memory-intensive then using a DOM tree approach.

While parsing XML files, SAX drivers invoke certain methods in response to different parsing events. Whenever a SAX driver has read a certain part of a document, it calls a specific method. SAX drivers respond to events like the start of a document, the start of an element, etc. A full list of the content handler implemented by the **VWSAX30** driver is given in table 3.4.

By overwriting these functions in a class inheriting from a SAX-driver, the user can define the actions to be taken after the occurrence of such an event.

3.5.3 Loading XMI files into MOOSE

As there already was a loading procedure for loading CDIF files, the main task of loading XMI files was to overwrite the content handler methods of the **VWSAX30** driver,n **MSEXMIReader**, collecting the data received and passing it in the correct order to the **MSEImporter** which instantiates the objects in the MOOSE model.

[7]Complete XMI files of FAMIX models are available here: http://www.iam.unibe.ch/~schlpbch/XMIforMoose

[8]**VWSAX30** is available here: ftp://ftp.o-x.org/outgoing/SaxParserForVw/

[9]Unfortunately, due to the crashes in validating mode this feature of the **VWSAX30** driver had to be turned off. In order to assure the validity of the XMI and DTD files created, **Xerces** of the Apache XML project (http://xml.apache.org) and Microsoft's free **xmlint** (http://msdn.microsoft.com/downloads/) were used.

Content Handlers of the *VWSAX30* Driver	
characters: aString	Receive notification of character data
endDocument	Receive notification of the end of a document
endElement: localName namespace: nameSpace prefix: nameSpacePrefix	Receive notification of the end of an element
ignorableWhitespace: aString	Receive notification of ignorable whitespace in element content
processingInstruction: targetString data: dataString	Receive notification of a processing instruction
startDocument	Receive notification of the start of a document
startElement: localName namespace: nameSpace prefix: nameSpacePrefix attributes: atts	Receive notification of the start of an element

Table 3.4: Content handlers of the **VWSAX30** driver

In order to perform this task, the **MSEXMIReader** has to be able to distinguish whether or not the content of the **startElement** event received describes the start of a new FAMIX object or is just an attribute belonging to this object. In order to make this decision, the following procedure was implemented:

A dictionary describing which tags define the start of a FAMIX entity is set up, using the **Generic Class Descriptions** described in section 3.2. All classes beneath **MSEAbstract-ModelRoot** are recursively 'scanned' to check whether they are FAMIX classes. If true, the MOOSE class name of this FAMIX class is extracted from the description and stored in a dictionary[10].

Now, whenever a **startElement** event occurs, a lookup in this dictionary is made to see whether or not it signals the start of a new MOOSE object. The following dispatch occurs:

- **The event describes the start of a new object.**
  A new collection has to be created and a pair consisting of the MOOSE class name and the name of this instance is added to it.

- **The event describes the start of an attribute of an object.**
  A pair describing the attribute and its value is added to the collection.

A similar dispatch has to be done for **endElement** events: Only if the content of the **endElement** event describes the end of a class definition, then the collection describing this object can be passed on to the **MSEImporter**, which in turn will actually instantiate the object within the model.

This procedure is repeated until a **endDocument** event occurs, signalling that the end of the XMI file was read.

3.6 Post processing XML Using XSL

As described in subsection 2.2.1, XML is likely to become the de facto standard for transferring information between applications and offers great support for storing, processing and loading data. Currently many new technologies for processing XML data are being developed. This section describes one of it: XSL, a W3C recommendation capable of transforming XML data into print documents like HTML.

[10] Setting up this dictionary anew before every load guarantees that the loader is informed of changes to the FAMIX Model, i.e. has no problems loading new classes.

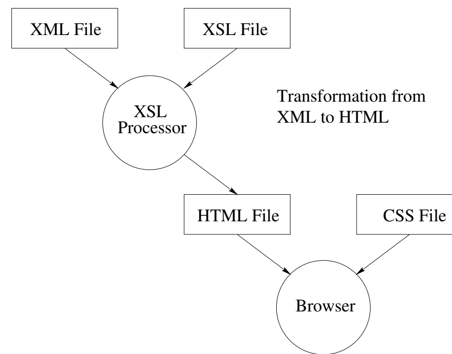

Figure 3.6: Transformation from XML to HTML

3.6.1 From XML to HTML

The XML file produced of a MOOSE Model contains all the information of the model in a verbose but not easy to read way. Therefore a post processing that extracts this information in an easy to read and navigate way (similar to the way Javadoc does for Java files) would enhance its accessibility. As XSL has the properties to accomplish this task, it was used to generate a set of easy to navigate HTML files, describing the extracted information in a appealing way.

Figure 3.6 illustrates the general process of how transforming XML to HTML is done. The main driver of the XML transformation is the XSL processor. The XSL processor reads in both the XML document and the XSL stylesheet. The XSL stylesheet describes a set of patterns to match within the XML document and the transformations to apply when a match is found. Each match/transformation pair is called an XSL template. Pattern matches are described in terms of the tags and attributes for elements found within an XML document. Transformations extract information from the XML document and format it into HTML.

The transformation process works in a way very analogous to the way scripting languages such as Perl or AWK operate. In that sense XSL could be called a scripting language, especially since it contains elements of control flow similar to a scripting language.

Cascading Style Sheets (CSS)[11] [LB99] are used to provide formating information to the browser when displaying the generated HTML. CSS provide the advantage of eliminating the repetition of many formating tags for HTML elements; instead they are extracted at display time from a style in the stylesheet.

3.6.2 Overview of XSL

The full XSL language logically consists of three component languages which are described in three W3C recommendations:

XPATH (XML **Path Language) [CD99]** A language for referencing specific parts of an XML document.

XSLT (XSL **Transformations) [Cla99]** A language for describing how to transform one XML document (represented as a tree) into another.

[11] To be precise: Only the CSS properties described in CSS 1 were used. Even tough CSS 2 is the current recommendation, most browsers even still have problems supporting CSS 1.

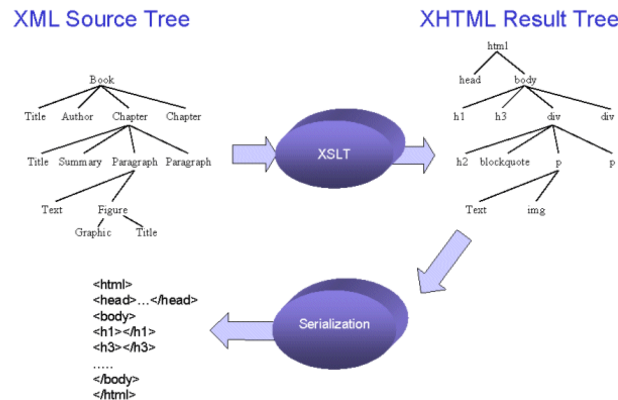

Figure 3.7: Applying the template rules to the XML Tree

XSL **(Extensible Stylesheet Language) [ABC+00]** XSLT plus a description of a set of Formatting Objects and Formatting Properties.

XSL goes through two main steps in formatting the data. First it produces a source tree by matching specified patterns with XML elements. This tree is then processed to produce a results tree, based on actions specified in template rules. A XML parser then takes the XML data and the XSL formatting instructions and transforms the results tree into an HTML document for display in a Web client. Even tough the most widely used and supported output format, HTML is not the only possible output. XSL is independent of the display language, so it is possible to develop a XSL template file that translates into other types of files.

3.6.3 Overview of CSS

CSS is a style sheet language that allows authors and users to attach style (e.g., fonts, colouring or spacing) to structured documents (e.g., HTML documents and XML applications). By separating the presentation style of documents from the content of documents, CSS simplifies Web authoring and site maintenance.

CSS supports media-specific style sheets so that authors may tailor the presentation of their documents to visual browsers, printers, hand held devices, etc. This specification also supports content positioning, down loadable fonts, table layout, features for internationalisation, automatic counters and numbering, and some properties related to user interface.

Using CSS, different attributes can be applied to every tag in a document. These attributes are expressed in "rules"; a rule is made up of a selector (the tag to which it applies) and a declaration (the attributes to be applied).

Apart from separating presentation style from content, CSS also has the advantage of reducing the overall size of the produced files. Information how to format an attribute is stored only once in the CSS file and does not have to be included for every attribute in a document.

3.6.4 XSL and CSS used for FAMIX

As mentioned above, XSL and CSS were used to produce a set of HTML files of the MOOSE model saved in a XMI file. A short example of how this was achieved is given here.

An entry for the **belongsToClass** property of the FAMIX class **Method** could look like this in the XMI file:

```
<FAMIX.Method>
  <FAMIX.Method.belongsToClass>
     AbstractDestinationAddress
  </FAMIX.Method.belongsToClass>
</FAMIX.Method>
```

In order to transform this into HTML, the following XSL template to match with the above XML entry is defined:

```
<xsl:template match="FAMIX.Method">
  <xsl:variable name="belongsToClass">
    <xsl:value-of select="FAMIX.Method.belongsToClass">
  </xsl:variable>

  <div align="center">
    <table border="1" width="75%" cellpadding="2">
      <tr>
        <td class="method-title" width="20%">belongsToClass</td>
        <td class="method-name">
          <a name="$belongsToClass">
            <xsl:value-of select="belongsToClass">
          </a>
        </td>
      </tr>
    </table>
  </div>
</xsl:template>
```

Invoking a XSL processor, passing a XML and a XSL file containing the code snippets described above as arguments, produces the following HTML snippet:

```
<div align="center">
  <table border="1" width="75%" cellpadding="2" >
    <tr>
       <td class="method-title" width="20%">belongsToClass</td>
       <td class="method-name">
         <a name="belongsToClass">
         AbstractDestinationAddress
         </a>
       </td>
     </tr>
  </table>
</div>
```

method-title and **method-name** are hooks for CSS, describing how the table cells are rendered. A CSS entry describing the look of the table's cells could look like this:

```
.method-title {
        background-color:#9999cc;
        color : #ffffff;
        font-family : Verdana, Helvetica;
        font-size : 12pt;
        font-weight : bold;
}

.method-name {
        background-color:#ffffe0;
        color : #6633cc;
        font-family : Verdana, Helvetica;
        font-size : 12pt;
}
```

The XSL processor used is a Java-based, open-source tool called **LotusXSL** [12] developed by IBM's alphaworks project. **LotusXSL** is a complete and a robust reference implementation of the W3C Recommendations for XSL Transformations (XSLT) and the XML Path Language (XPATH).

Note that the XSL transformation produces a single HTML file. As this leads to very big files which would simply take to long to download, **LotusXSL** offers an extension to the W3C Recommendation which allows the production of multiple files from one XML file. Using this redirect extension, for every FAMIX object a HTML file was produced with hypertext links easing the navigation. Additionally a menu using HTML frames and a little Javascript script to simulate pull down menus was generated.

By simply invoking the XSL producer, a complete documentation of a FAMIX model is generated, illustrating the power of this technology. A screen shot showing the result is given in figure 3.8[13].

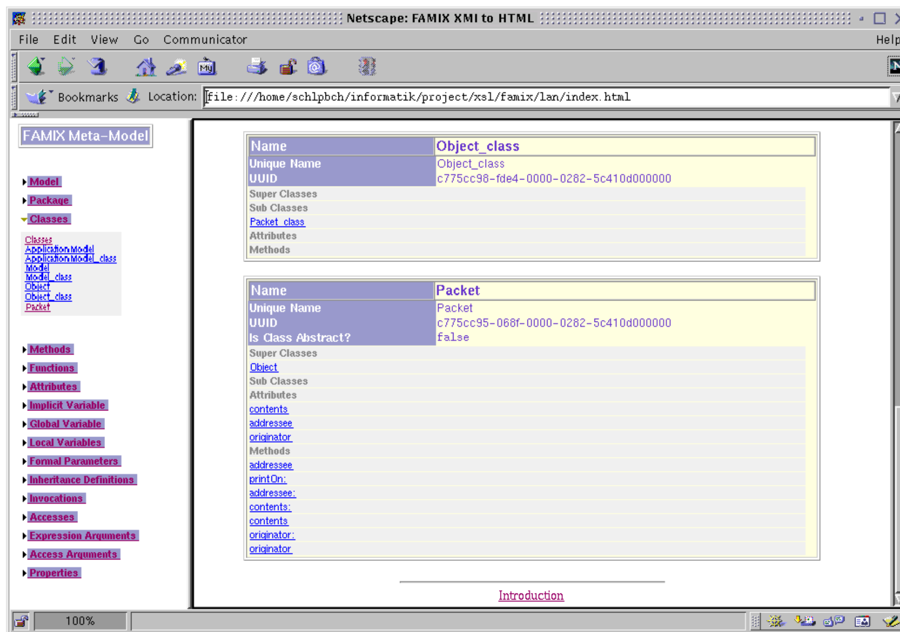

Figure 3.8: Result of using XSL for FAMIX

[12]**LotusXSL** is available here: http://www.alphaworks.ibm.com/aw.nsf/techmain/lotusxsl..

[13]An example of a XML file transformed to HTML using XSL is available here: http://www.iam.unibe.ch/~schlpbch/XMIforMoose

Chapter 4

Conclusion and Further Work

4.1 Conclusion

This report describes how XMI support was added to the MOOSE Reengineering Environment. It describes the key technologies involved: XMI, XML, and MOF. The use of XMI offers two major advantages: First, XMI is based on XML, allowing the use of XML based technology (such as XSL described in section 3.6) and second, XMI is MOF based and therefore offers excellent integration to MOF based metamodels such as UML.

XMI for MOOSE consists of four parts:

1. XMI **Loader**
   A loader that is capable of reading in XML files and loading them into MOOSE.
2. XMI/DTD **Saver**
   This parts consists of two components:
   (a) XMI **Saver**
   A saver that is capable of saving the currently loaded model in a XMI/XML conforming way.
   (b) XMI DTD **Producer and Saver**
   An architecture to create DTDS of the current metamodel. This implies transforming the metamodel information to a MOF compliant metamodel plus transforming this information into a valid DTD.
3. **MOF**
   A basic MOF implementation.
4. **Metametamodel Support**
   To be able to change the FAMIX metamodel, a clean and extensible architecture for defining the FAMIX metamodel was introduced.

Benefits of XMI for MOOSE are:

1. Using XMI assures that other tools are able of loading, processing, and storing the data provided by the MOOSE reengineering environment.
2. The metametamodel support provided by the **Generic Class Description** package opens the road for extensions and offers the freedom to experiment with different metamodels.
3. The generic DTD XMI-Producer and the basic MOF implementation can be reused independent of MOOSE in any project using XMI.

Additionally a post processing of the created XMI file using XSL and CSS is described, showing the strength and potential of XML as the underlying technology for data storage.

4.2 Further Work

While the MOF and the DTD production rules are generic and independent of MOOSE, the saving of a XMI file is tightly coupled to the MOOSE architecture (see subsection 2.2.3).

In the chapter '*XML Document Production Rules*' of the XMI [XMI98] specification describes a generic way of saving XML files. Therefore, it would be obvious to drop the current implementation in favour of this reusable approach.

An implementation would require two steps:

1. Transforming the data in the MOOSE model into a representation compliant to the rules for saving XMI.
2. Streaming out this transformed model according to the rules described in *XML Document Production Rules.*

Creating and saving the XMI in this generic way would allow a reuse of this component in any project using XMI, independent of MOOSE.

Further improvements in the area of memory consumption must be made: All objects having UUIDS increases the memory consumption for large models significantly, decreasing the speed of the whole system. Especially with multiple models being loaded at the same time, memory becomes scarce. Investigations how to reduce memory consumption are necessary. Possible solutions could include using a more elaborate approach of creating UUIDS (e.g. just in time) or less memory consuming identifiers (e.g. plain integers).

Appendix A

Example of a XMI File

The example of a XMI file attached below is part of a LAN simulation. Its UML diagram is shown in figure A.1:

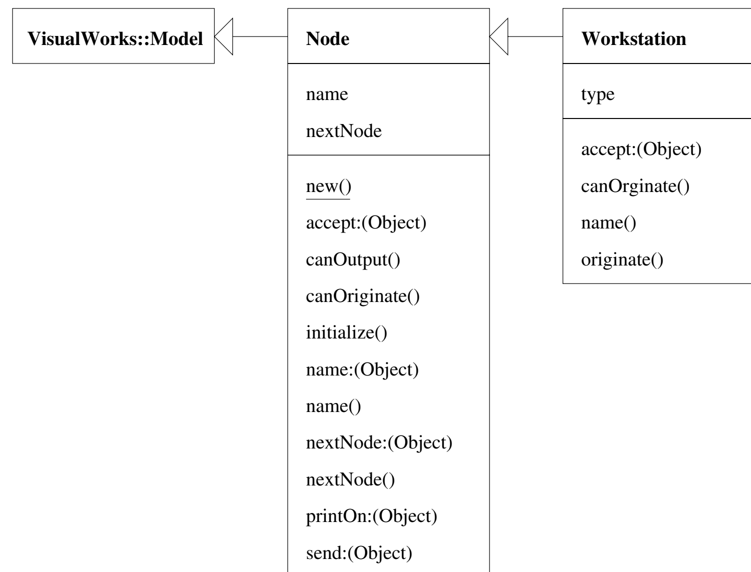

Figure A.1: UML diagram of a part a LAN simulation

The XMI file produced contains instance scope well as class scope information ('new' is a class-scope method). Note that in the file, a **Property** object (as described in subsection 3.2.4) is aggregated to the class *Model*. The class *Model* is not part of the LAN, but part of the VISUALWORKS class hierarchy. Therefore the class *Model* consists solely of an empty placeholder (a 'stub').

Example of a XMI File

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE XMI SYSTEM "famix20.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="December 22, 2000 2:37:34.000">
  <XMI.header>
        <XMI.documentation>
          <XMI.exporter>XMI MASTER</XMI.exporter>
          <XMI.exporterVersion>1.0</XMI.exporterVersion>
        </XMI.documentation>
        <XMI.metamodel xmi.name="FAMIX" xmi.version="2.0"/>
  </XMI.header>
  <XMI.content>
    <FAMIX.Model xmi.id="c8f76491-d850-0000-0282-5c410d000000">
        <FAMIX.Model.exporterName>MOOSE</FAMIX.Model.exporterName>
        <FAMIX.Model.exporterVersion>2.14</FAMIX.Model.exporterVersion>
        <FAMIX.Model.exporterDate>2000/12/22</FAMIX.Model.exporterDate>
        <FAMIX.Model.exporterTime>2:37:21</FAMIX.Model.exporterTime>
        <FAMIX.Model.publisherName>Unknown</FAMIX.Model.publisherName>
        <FAMIX.Model.parsedSystemName>Unknown</FAMIX.Model.parsedSystemName>
        <FAMIX.Model.extractionLevel>3</FAMIX.Model.extractionLevel>
        <FAMIX.Model.sourceLanguage>Smalltalk</FAMIX.Model.sourceLanguage>
        <FAMIX.Model.sourceDialect>VisualWorks</FAMIX.Model.sourceDialect>
```

```
</FAMIX.Model>

<FAMIX.Class xmi.id="Node_class">
    <FAMIX.Entity.name>Node_class</FAMIX.Entity.name>
    <FAMIX.Entity.uniqueName>Node_class</FAMIX.Entity.uniqueName>
    <FAMIX.Class.isAbstract xmi.value="false"/>
</FAMIX.Class>

<FAMIX.InheritanceDefinition xmi.id="c8f76491-d851-0000-0282-5c410d000000">
    <FAMIX.InheritanceDefinition.subclass>Node_class</FAMIX.InheritanceDefinition.subclass>
    <FAMIX.InheritanceDefinition.superclass>Model_class</FAMIX.InheritanceDefinition.superclass>
</FAMIX.InheritanceDefinition>

<FAMIX.Method xmi.id="Node_class.new()">
    <FAMIX.Entity.name>new</FAMIX.Entity.name>
    <FAMIX.Entity.uniqueName>Node_class.new()</FAMIX.Entity.uniqueName>
    <FAMIX.BehaviouralEntity.accessControlQualifier>public</FAMIX.BehaviouralEntity.accessControlQualifier>
    <FAMIX.BehaviouralEntity.signature>new()</FAMIX.BehaviouralEntity.signature>
    <FAMIX.Method.belongsToClass>Node_class</FAMIX.Method.belongsToClass>
    <FAMIX.Method.hasClassScope xmi.value="false"/>
    <FAMIX.Method.isAbstract xmi.value="false"/>
    <FAMIX.Method.isConstructor xmi.value="true"/>
</FAMIX.Method>

<FAMIX.Class xmi.id="Node">
    <FAMIX.Entity.name>Node</FAMIX.Entity.name>
    <FAMIX.Entity.uniqueName>Node</FAMIX.Entity.uniqueName>
    <FAMIX.Class.isAbstract xmi.value="false"/>
</FAMIX.Class>

<FAMIX.InheritanceDefinition xmi.id="c8f76491-d852-0000-0282-5c410d000000">
    <FAMIX.InheritanceDefinition.subclass>Node</FAMIX.InheritanceDefinition.subclass>
    <FAMIX.InheritanceDefinition.superclass>Model</FAMIX.InheritanceDefinition.superclass>
</FAMIX.InheritanceDefinition>

<FAMIX.Attribute xmi.id="Node.name">
    <FAMIX.Entity.name>name</FAMIX.Entity.name>
    <FAMIX.Entity.uniqueName>Node.name</FAMIX.Entity.uniqueName>
    <FAMIX.Attribute.belongsToClass>Node</FAMIX.Attribute.belongsToClass>
    <FAMIX.Attribute.accessControlQualifier>protected</FAMIX.Attribute.accessControlQualifier>
    <FAMIX.Attribute.hasClassScope xmi.value="false"/>
</FAMIX.Attribute>

<FAMIX.Attribute xmi.id="Node.nextNode">
```

```
    <FAMIX.Entity.name>nextNode</FAMIX.Entity.name>
    <FAMIX.Entity.uniqueName>Node.nextNode</FAMIX.Entity.uniqueName>
    <FAMIX.Attribute.belongsToClass>Node</FAMIX.Attribute.belongsToClass>
    <FAMIX.Attribute.accessControlQualifier>protected</FAMIX.Attribute.accessControlQualifier>
    <FAMIX.Attribute.hasClassScope xmi.value="false"/>
</FAMIX.Attribute>

<FAMIX.Method xmi.id="Node.printOn:(Object)">
    <FAMIX.Entity.name>printOn:</FAMIX.Entity.name>
    <FAMIX.Entity.uniqueName>Node.printOn:(Object)</FAMIX.Entity.uniqueName>
    <FAMIX.BehaviouralEntity.accessControlQualifier>public</FAMIX.BehaviouralEntity.accessControlQualifier>
    <FAMIX.BehaviouralEntity.signature>printOn:(Object)</FAMIX.BehaviouralEntity.signature>
    <FAMIX.Method.belongsToClass>Node</FAMIX.Method.belongsToClass>
    <FAMIX.Method.hasClassScope xmi.value="false"/>
    <FAMIX.Method.isAbstract xmi.value="false"/>
    <FAMIX.Method.isConstructor xmi.value="false"/>
</FAMIX.Method>

<FAMIX.Method xmi.id="Node.nextNode()">
    <FAMIX.Entity.name>nextNode</FAMIX.Entity.name>
    <FAMIX.Entity.uniqueName>Node.nextNode()</FAMIX.Entity.uniqueName>
    <FAMIX.BehaviouralEntity.accessControlQualifier>public</FAMIX.BehaviouralEntity.accessControlQualifier>
    <FAMIX.BehaviouralEntity.signature>nextNode()</FAMIX.BehaviouralEntity.signature>
    <FAMIX.Method.belongsToClass>Node</FAMIX.Method.belongsToClass>
    <FAMIX.Method.hasClassScope xmi.value="false"/>
    <FAMIX.Method.isAbstract xmi.value="false"/>
    <FAMIX.Method.isConstructor xmi.value="false"/>
</FAMIX.Method>

<FAMIX.Access xmi.id="c8f76491-d853-0000-0282-5c410d000000">
    <FAMIX.Access.accesses>Node.nextNode</FAMIX.Access.accesses>
    <FAMIX.Access.accessedIn>Node.nextNode()</FAMIX.Access.accessedIn>
    <FAMIX.Access.isAccessLValue xmi.value="false"/>
</FAMIX.Access>

<FAMIX.Method xmi.id="Node.send:(Object)">
    <FAMIX.Entity.name>send:</FAMIX.Entity.name>
    <FAMIX.Entity.uniqueName>Node.send:(Object)</FAMIX.Entity.uniqueName>
    <FAMIX.BehaviouralEntity.accessControlQualifier>public</FAMIX.BehaviouralEntity.accessControlQualifier>
    <FAMIX.BehaviouralEntity.signature>send:(Object)</FAMIX.BehaviouralEntity.signature>
    <FAMIX.Method.belongsToClass>Node</FAMIX.Method.belongsToClass>
    <FAMIX.Method.hasClassScope xmi.value="false"/>
    <FAMIX.Method.isAbstract xmi.value="false"/>
    <FAMIX.Method.isConstructor xmi.value="false"/>
```

```
</FAMIX.Method>

<FAMIX.Method xmi.id="Node.canOutput()">
    <FAMIX.Entity.name>canOutput</FAMIX.Entity.name>
    <FAMIX.Entity.uniqueName>Node.canOutput()</FAMIX.Entity.uniqueName>
    <FAMIX.BehaviouralEntity.accessControlQualifier>public</FAMIX.BehaviouralEntity.accessControlQualifier>
    <FAMIX.BehaviouralEntity.signature>canOutput()</FAMIX.BehaviouralEntity.signature>
    <FAMIX.Method.belongsToClass>Node</FAMIX.Method.belongsToClass>
    <FAMIX.Method.hasClassScope xmi.value="false"/>
    <FAMIX.Method.isAbstract xmi.value="false"/>
    <FAMIX.Method.isConstructor xmi.value="false"/>
</FAMIX.Method>

<FAMIX.Method xmi.id="Node.canOriginate()">
    <FAMIX.Entity.name>canOriginate</FAMIX.Entity.name>
    <FAMIX.Entity.uniqueName>Node.canOriginate()</FAMIX.Entity.uniqueName>
    <FAMIX.BehaviouralEntity.accessControlQualifier>public</FAMIX.BehaviouralEntity.accessControlQualifier>
    <FAMIX.BehaviouralEntity.signature>canOriginate()</FAMIX.BehaviouralEntity.signature>
    <FAMIX.Method.belongsToClass>Node</FAMIX.Method.belongsToClass>
    <FAMIX.Method.hasClassScope xmi.value="false"/>
    <FAMIX.Method.isAbstract xmi.value="false"/>
    <FAMIX.Method.isConstructor xmi.value="false"/>
</FAMIX.Method>

<FAMIX.Method xmi.id="Node.name()">
    <FAMIX.Entity.name>name</FAMIX.Entity.name>
    <FAMIX.Entity.uniqueName>Node.name()</FAMIX.Entity.uniqueName>
    <FAMIX.BehaviouralEntity.accessControlQualifier>public</FAMIX.BehaviouralEntity.accessControlQualifier>
    <FAMIX.BehaviouralEntity.signature>name()</FAMIX.BehaviouralEntity.signature>
    <FAMIX.Method.belongsToClass>Node</FAMIX.Method.belongsToClass>
    <FAMIX.Method.hasClassScope xmi.value="false"/>
    <FAMIX.Method.isAbstract xmi.value="false"/>
    <FAMIX.Method.isConstructor xmi.value="false"/>
</FAMIX.Method>

<FAMIX.Access xmi.id="c8f76491-d854-0000-0282-5c410d000000">
    <FAMIX.Access.accesses>Node.name</FAMIX.Access.accesses>
    <FAMIX.Access.accessedIn>Node.name()</FAMIX.Access.accessedIn>
    <FAMIX.Access.isAccessLValue xmi.value="false"/>
</FAMIX.Access>

<FAMIX.Method xmi.id="Node.nextNode:(Object)">
    <FAMIX.Entity.name>nextNode:</FAMIX.Entity.name>
    <FAMIX.Entity.uniqueName>Node.nextNode:(Object)</FAMIX.Entity.uniqueName>
```

```
    <FAMIX.BehaviouralEntity.accessControlQualifier>public</FAMIX.BehaviouralEntity.accessControlQualifier>
    <FAMIX.BehaviouralEntity.signature>nextNode:(Object)</FAMIX.BehaviouralEntity.signature>
    <FAMIX.Method.belongsToClass>Node</FAMIX.Method.belongsToClass>
    <FAMIX.Method.hasClassScope xmi.value="false"/>
    <FAMIX.Method.isAbstract xmi.value="false"/>
    <FAMIX.Method.isConstructor xmi.value="false"/>
</FAMIX.Method>

<FAMIX.Access xmi.id="c8f76491-d855-0000-0282-5c410d000000">
    <FAMIX.Access.accesses>Node.nextNode</FAMIX.Access.accesses>
    <FAMIX.Access.accessedIn>Node.nextNode:(Object)</FAMIX.Access.accessedIn>
    <FAMIX.Access.isAccessLValue xmi.value="true"/>
</FAMIX.Access>

<FAMIX.Method xmi.id="Node.initialize()">
    <FAMIX.Entity.name>initialize</FAMIX.Entity.name>
    <FAMIX.Entity.uniqueName>Node.initialize()</FAMIX.Entity.uniqueName>
    <FAMIX.BehaviouralEntity.accessControlQualifier>public</FAMIX.BehaviouralEntity.accessControlQualifier>
    <FAMIX.BehaviouralEntity.signature>initialize()</FAMIX.BehaviouralEntity.signature>
    <FAMIX.Method.belongsToClass>Node</FAMIX.Method.belongsToClass>
    <FAMIX.Method.hasClassScope xmi.value="false"/>
    <FAMIX.Method.isAbstract xmi.value="false"/>
    <FAMIX.Method.isConstructor xmi.value="false"/>
</FAMIX.Method>

<FAMIX.Method xmi.id="Node.name:(Object)">
    <FAMIX.Entity.name>name:</FAMIX.Entity.name>
    <FAMIX.Entity.uniqueName>Node.name:(Object)</FAMIX.Entity.uniqueName>
    <FAMIX.BehaviouralEntity.accessControlQualifier>public</FAMIX.BehaviouralEntity.accessControlQualifier>
    <FAMIX.BehaviouralEntity.signature>name:(Object)</FAMIX.BehaviouralEntity.signature>
    <FAMIX.Method.belongsToClass>Node</FAMIX.Method.belongsToClass>
    <FAMIX.Method.hasClassScope xmi.value="false"/>
    <FAMIX.Method.isAbstract xmi.value="false"/>
    <FAMIX.Method.isConstructor xmi.value="false"/>
</FAMIX.Method>

<FAMIX.Access xmi.id="c8f76491-d856-0000-0282-5c410d000000">
    <FAMIX.Access.accesses>Node.name</FAMIX.Access.accesses>
    <FAMIX.Access.accessedIn>Node.name:(Object)</FAMIX.Access.accessedIn>
    <FAMIX.Access.isAccessLValue xmi.value="true"/>
</FAMIX.Access>

<FAMIX.Method xmi.id="Node.accept:(Object)">
    <FAMIX.Entity.name>accept:</FAMIX.Entity.name>
```

```
    <FAMIX.Entity.uniqueName>Node.accept:(Object)</FAMIX.Entity.uniqueName>
    <FAMIX.BehaviouralEntity.accessControlQualifier>public</FAMIX.BehaviouralEntity.accessControlQualifier>
    <FAMIX.BehaviouralEntity.signature>accept:(Object)</FAMIX.BehaviouralEntity.signature>
    <FAMIX.Method.belongsToClass>Node</FAMIX.Method.belongsToClass>
    <FAMIX.Method.hasClassScope xmi.value="false"/>
    <FAMIX.Method.isAbstract xmi.value="false"/>
    <FAMIX.Method.isConstructor xmi.value="false"/>
</FAMIX.Method>

<FAMIX.Class xmi.id="WorkStation_class">
    <FAMIX.Entity.name>WorkStation_class</FAMIX.Entity.name>
    <FAMIX.Entity.uniqueName>WorkStation_class</FAMIX.Entity.uniqueName>
    <FAMIX.Class.isAbstract xmi.value="false"/>
</FAMIX.Class>

<FAMIX.InheritanceDefinition xmi.id="c8f76491-d857-0000-0282-5c410d000000">
    <FAMIX.InheritanceDefinition.subclass>WorkStation_class</FAMIX.InheritanceDefinition.subclass>
    <FAMIX.InheritanceDefinition.superclass>Node_class</FAMIX.InheritanceDefinition.superclass>
</FAMIX.InheritanceDefinition>

<FAMIX.Class xmi.id="WorkStation">
    <FAMIX.Entity.name>WorkStation</FAMIX.Entity.name>
    <FAMIX.Entity.uniqueName>WorkStation</FAMIX.Entity.uniqueName>
    <FAMIX.Class.isAbstract xmi.value="false"/>
</FAMIX.Class>

<FAMIX.InheritanceDefinition xmi.id="c8f76491-d858-0000-0282-5c410d000000">
    <FAMIX.InheritanceDefinition.subclass>WorkStation</FAMIX.InheritanceDefinition.subclass>
    <FAMIX.InheritanceDefinition.superclass>Node</FAMIX.InheritanceDefinition.superclass>
</FAMIX.InheritanceDefinition>

<FAMIX.Attribute xmi.id="WorkStation.type">
    <FAMIX.Entity.name>type</FAMIX.Entity.name>
    <FAMIX.Entity.uniqueName>WorkStation.type</FAMIX.Entity.uniqueName>
    <FAMIX.Attribute.belongsToClass>WorkStation</FAMIX.Attribute.belongsToClass>
    <FAMIX.Attribute.accessControlQualifier>protected</FAMIX.Attribute.accessControlQualifier>
    <FAMIX.Attribute.hasClassScope xmi.value="false"/>
</FAMIX.Attribute>

<FAMIX.Method xmi.id="WorkStation.canOriginate()">
    <FAMIX.Entity.name>canOriginate</FAMIX.Entity.name>
    <FAMIX.Entity.uniqueName>WorkStation.canOriginate()</FAMIX.Entity.uniqueName>
    <FAMIX.BehaviouralEntity.accessControlQualifier>public</FAMIX.BehaviouralEntity.accessControlQualifier>
    <FAMIX.BehaviouralEntity.signature>canOriginate()</FAMIX.BehaviouralEntity.signature>
```

```
    <FAMIX.Method.belongsToClass>WorkStation</FAMIX.Method.belongsToClass>
    <FAMIX.Method.hasClassScope xmi.value="false"/>
    <FAMIX.Method.isAbstract xmi.value="false"/>
    <FAMIX.Method.isConstructor xmi.value="false"/>
</FAMIX.Method>

<FAMIX.Method xmi.id="WorkStation.accept:(Object)">
    <FAMIX.Entity.name>accept:</FAMIX.Entity.name>
    <FAMIX.Entity.uniqueName>WorkStation.accept:(Object)</FAMIX.Entity.uniqueName>
    <FAMIX.BehaviouralEntity.accessControlQualifier>public</FAMIX.BehaviouralEntity.accessControlQualifier>
    <FAMIX.BehaviouralEntity.signature>accept:(Object)</FAMIX.BehaviouralEntity.signature>
    <FAMIX.Method.belongsToClass>WorkStation</FAMIX.Method.belongsToClass>
    <FAMIX.Method.hasClassScope xmi.value="false"/>
    <FAMIX.Method.isAbstract xmi.value="false"/>
    <FAMIX.Method.isConstructor xmi.value="false"/>
</FAMIX.Method>

<FAMIX.Method xmi.id="WorkStation.name()">
    <FAMIX.Entity.name>name</FAMIX.Entity.name>
    <FAMIX.Entity.uniqueName>WorkStation.name()</FAMIX.Entity.uniqueName>
    <FAMIX.BehaviouralEntity.accessControlQualifier>public</FAMIX.BehaviouralEntity.accessControlQualifier>
    <FAMIX.BehaviouralEntity.signature>name()</FAMIX.BehaviouralEntity.signature>
    <FAMIX.Method.belongsToClass>WorkStation</FAMIX.Method.belongsToClass>
    <FAMIX.Method.hasClassScope xmi.value="false"/>
    <FAMIX.Method.isAbstract xmi.value="false"/>
    <FAMIX.Method.isConstructor xmi.value="false"/>
</FAMIX.Method>

<FAMIX.Method xmi.id="WorkStation.originate:(Object)">
    <FAMIX.Entity.name>originate:</FAMIX.Entity.name>
    <FAMIX.Entity.uniqueName>WorkStation.originate:(Object)</FAMIX.Entity.uniqueName>
    <FAMIX.BehaviouralEntity.accessControlQualifier>public</FAMIX.BehaviouralEntity.accessControlQualifier>
    <FAMIX.BehaviouralEntity.signature>originate:(Object)</FAMIX.BehaviouralEntity.signature>
    <FAMIX.Method.belongsToClass>WorkStation</FAMIX.Method.belongsToClass>
    <FAMIX.Method.hasClassScope xmi.value="false"/>
    <FAMIX.Method.isAbstract xmi.value="false"/>
    <FAMIX.Method.isConstructor xmi.value="false"/>
</FAMIX.Method>

<FAMIX.Class xmi.id="Model_class">
    <FAMIX.Entity.name>Model_class</FAMIX.Entity.name>
    <FAMIX.Entity.uniqueName>Model_class</FAMIX.Entity.uniqueName>
</FAMIX.Class>
```

```
    <FAMIX.Property xmi.id="c8f76491-d85a-0000-0282-5c410d000000">
        <FAMIX.Property.name>stub</FAMIX.Property.name>
        <FAMIX.Property.value>"true"</FAMIX.Property.value>
        <FAMIX.Property.belongsToID>Model_class</FAMIX.Property.belongsToID>
    </FAMIX.Property>

    <FAMIX.Class xmi.id="Model">
        <FAMIX.Entity.name>Model</FAMIX.Entity.name>
        <FAMIX.Entity.uniqueName>Model</FAMIX.Entity.uniqueName>
    </FAMIX.Class>

    <FAMIX.Property xmi.id="c8f76491-d85c-0000-0282-5c410d000000">
        <FAMIX.Property.name>stub</FAMIX.Property.name>
        <FAMIX.Property.value>"true"</FAMIX.Property.value>
        <FAMIX.Property.belongsToID>Model</FAMIX.Property.belongsToID>
    </FAMIX.Property>
  </XMI.content>
</XMI>
```

Appendix B

FAMIX DTD

```
<?xml version="1.0" encoding="UTF-8" ?>
<!-- ______________________________________________________________ -->
<!--                                                                -->
<!-- generated with XMLDTD-Generator                                -->
<!--                                                                -->
<!-- Timestamp: December 22, 2000 2:37:33.000                        -->
<!--                                                                -->
<!-- by Andreas Schlapbach <schlpbch@iam.unibe.ch>                  -->
<!-- ______________________________________________________________ -->
<!-- ______________________________________________________________ -->
<!--                                                                -->
<!-- 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,    -->
<!-- metamodel, and metametamodel                                   -->
<!-- ______________________________________________________________ -->

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

<!-- ______________________________________________________________ -->
<!--                                                                -->
<!-- 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 metamodel class must have to conform to -->
<!-- the XMI specification.                                       -->
<!-- ____________________________________________________________ -->

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

<!-- ____________________________________________________________ -->
<!--                                                              -->
<!-- XMI.link.att defines the attributes that each XML element that -->
<!-- corresponds to a metamodel class must have to enable it to   -->
<!-- function as a simple XLink as well as refer to model         -->
<!-- constructs within the same XMI file.                         -->
<!-- ____________________________________________________________ -->

<!ENTITY % XMI.link.att
                        '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.metamodel identifies the metamodel(s) for the transferred -->
<!-- data                                                         -->
<!-- ____________________________________________________________ -->

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

<!-- ____________________________________________________________ -->
<!--                                                              -->
<!-- XMI.metametamodel identifies the metametamodel(s) for the    -->
<!-- transferred data                                             -->
<!-- ____________________________________________________________ -->

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

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

<!ELEMENT XMI.content ANY >

<!-- ____________________________________________________________ -->
<!--                                                              -->
<!-- XMI.extensions contains data to transfer that does not conform -->
```

```
<!-- to the metamodel(s) in the header                               -->
<!-- _______________________________________________________________ -->

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

<!-- _______________________________________________________________ -->
<!--                                                                 -->
<!-- extension contains information related to a specific model      -->
<!-- construct that is not defined in the metamodel(s) in the header -->
<!-- _______________________________________________________________ -->

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

<!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 Information: Famix   -->
<!-- ________________________________________________ -->

<!-- ________________________________________________ -->
<!--                                                  -->
<!-- Meta-Model Class: FAMIX.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: FAMIX.Argument   -->
<!-- ________________________________________________ -->

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

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

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

<!-- ________________________________________________ -->
<!--                                                  -->
<!-- Meta-Model Class: FAMIX.AccessArgument   -->
<!-- ________________________________________________ -->

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

<!-- ________________________________________________ -->
<!--                                                  -->
<!-- Meta-Model Class: FAMIX.ExpressionArgument   -->
<!-- ________________________________________________ -->

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

<!-- ________________________________________________ -->
<!--                                                  -->
<!-- Meta-Model Class: FAMIX.Association   -->
<!-- ________________________________________________ -->

<!ELEMENT FAMIX.Association (FAMIX.Object.sourceAnchor?,
```

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

<!-- _________________________________________________ -->
<!--                                                   -->
<!-- Meta-Model Class: FAMIX.Access   -->
<!-- _________________________________________________ -->

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

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

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

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

<!-- _________________________________________________ -->
<!--                                                   -->
<!-- Meta-Model Class: FAMIX.Include   -->
<!-- _________________________________________________ -->

<!ELEMENT FAMIX.Include.includedFile (#PCDATA | XMI.reference)*>

<!ELEMENT FAMIX.Include.includingFile (#PCDATA | XMI.reference)*>

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

<!-- _________________________________________________ -->
<!--                                                   -->
<!-- Meta-Model Class: FAMIX.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: FAMIX.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.candidates (#PCDATA | XMI.reference)*>

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

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

<!-- _________________________________________________ -->
<!--                                                   -->
<!-- Meta-Model Class: FAMIX.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: FAMIX.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 | 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: FAMIX.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: FAMIX.Method   -->
<!-- _________________________________________________ -->

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

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

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

<!ELEMENT FAMIX.Method.isConstructor EMPTY>
<!ATTLIST FAMIX.Method.isConstructor
                        xmi.value ( true | false | null | 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?,
                        (XMI.extension*))?>
<!ATTLIST FAMIX.Method
                        %XMI.element.att; %XMI.link.att;
>

<!-- _________________________________________________ -->
<!--                                                   -->
<!-- Meta-Model Class: FAMIX.Class   -->
<!-- _________________________________________________ -->

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

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

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

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

<!-- _________________________________________________ -->
<!--                                                   -->
<!-- Meta-Model Class: FAMIX.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: FAMIX.StructuralEntity   -->
<!-- ________________________________________________ -->

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

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

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

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

<!-- ________________________________________________ -->
<!--                                                  -->
<!-- Meta-Model Class: FAMIX.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.StructuralEntity.interfaceSignatures*,
                        FAMIX.FormalParameter.belongsToBehaviour,
                        FAMIX.FormalParameter.position,
                        (XMI.extension*))?>
<!ATTLIST FAMIX.FormalParameter
                        %XMI.element.att; %XMI.link.att;
>

<!-- ________________________________________________ -->
<!--                                                  -->
<!-- Meta-Model Class: FAMIX.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.StructuralEntity.interfaceSignatures*,
                        FAMIX.LocalVariable.belongsToBehaviour,
                        (XMI.extension*))?>
<!ATTLIST FAMIX.LocalVariable
                        %XMI.element.att; %XMI.link.att;
>
```

```
<!-- ________________________________________________________ -->
<!--                                                          -->
<!-- Meta-Model Class: FAMIX.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 | 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.StructuralEntity.interfaceSignatures*,
                        FAMIX.Attribute.belongsToClass,
                        FAMIX.Attribute.accessControlQualifier?,
                        FAMIX.Attribute.hasClassScope?,
                        (XMI.extension*))?>
<!ATTLIST FAMIX.Attribute
                        %XMI.element.att; %XMI.link.att;
>

<!-- ________________________________________________________ -->
<!--                                                          -->
<!-- Meta-Model Class: FAMIX.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.StructuralEntity.interfaceSignatures*,
                        FAMIX.GlobalVariable.belongsToPackage?,
                        (XMI.extension*))?>
<!ATTLIST FAMIX.GlobalVariable
                        %XMI.element.att; %XMI.link.att;
>

<!-- ________________________________________________________ -->
<!--                                                          -->
<!-- Meta-Model Class: FAMIX.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.StructuralEntity.interfaceSignatures*,
                        FAMIX.ImplicitVariable.belongsToContext?,
                        (XMI.extension*))?>
<!ATTLIST FAMIX.ImplicitVariable
                        %XMI.element.att; %XMI.link.att;
>

<!-- ________________________________________________________ -->
<!--                                                          -->
<!-- Meta-Model Class: FAMIX.SourceFile   -->
<!-- ________________________________________________________ -->

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

```
                        %XMI.element.att; %XMI.link.att;
>

<!-- __________________________________________________ -->
<!--                                                    -->
<!-- Meta-Model Class: FAMIX.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.Object.sourceAnchor?,
                        FAMIX.Object.comments*,
                        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: FAMIX.Measurement   -->
<!-- __________________________________________________ -->

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

<!ELEMENT FAMIX.Measurement.value (#PCDATA | XMI.reference)*>

<!ELEMENT FAMIX.Measurement.belongsToEntity (#PCDATA | XMI.reference)*>

<!ELEMENT FAMIX.Measurement.relatedEntity (#PCDATA | XMI.reference)*>

<!ELEMENT FAMIX.Measurement (FAMIX.Measurement.name,
                        FAMIX.Measurement.value,
                        FAMIX.Measurement.belongsToEntity,
                        FAMIX.Measurement.relatedEntity?,
                        (XMI.extension*))?>
<!ATTLIST FAMIX.Measurement
                        %XMI.element.att; %XMI.link.att;
>

<!-- __________________________________________________ -->
<!--                                                    -->
<!-- Meta-Model Class: FAMIX.Property   -->
<!-- __________________________________________________ -->

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

<!ELEMENT FAMIX.Property.value (#PCDATA | XMI.reference)*>

<!ELEMENT FAMIX.Property.belongsToID (#PCDATA | XMI.reference)*>

<!ELEMENT FAMIX.Property (FAMIX.Property.name,
                        FAMIX.Property.value,
                        FAMIX.Property.belongsToID,
                        (XMI.extension*))?>
<!ATTLIST FAMIX.Property
```

```
                        %XMI.element.att; %XMI.link.att;
>

<!-- _________________________________________________ -->
<!--                                                   -->
<!-- Meta-Model Package: Famix   -->
<!-- _________________________________________________ -->

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

List of Figures

1.1 An overview of the XMI architecture of MOOSE 6

2.1 Architecture of MOOSE 10
2.2 Concept of the FAMIX Model 11
2.3 The complete FAMIX model 11
2.4 The core model 11
2.5 Abstract part of the model 12
2.6 The MOF model 18

3.1 The XMI architecture of MOOSE 21
3.2 The FAMIX class **Property** 24
3.3 'Attribute' lookup 26
3.4 Transforming the FAMIX metadata to a FAMIXDTD 27
3.5 Implemented parts of the MOF model (Classes grayed out are *not* implemented) 28
3.6 Transformation from XML to HTML 34
3.7 Applying the template rules to the XML Tree 35
3.8 Result of using XSL for FAMIX 37

A.1 UML diagram of a part a LAN simulation 41

List of Tables

3.1 **MSEModelClassDescriptor** class 23
3.2 **MSEModelAttributeDescriptor** class 23
3.3 **MSEModelMVAttributeDescriptor** class 23
3.4 Content handlers of the **VWSAX30** driver 33

Bibliography

[ABC^{+}00] Sharon Adler, Anders Berglund, Jeff Caruso, Stephen Deach, Paul Grosso, Eduardo Gutentag, Alex Milowski, Scott Parnell, Jeremy Richman, and Steve Zille. Extensible stylesheet language (XSL) 1.0 - w3c working draft 18-october-2000. Technical Report WD-xsl-20001018, World Wide Web Consortium, October 2000. See: http://www.w3.org/TR/2000/WD-xsl-20001018/.

[BPSMM00] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, and Eve Maler. Extensible Markup Language (XML) 1.0 (second edition) - w3c recommendation 6-october-2000. Technical Report REC-xml-20001006, World Wide Web Consortium, October 2000. See: http://www.w3.org/TR/2000/REC-xml-20001006.

[CD99] James Clark and Steve DeRose. XML path language (XPath) 1.0 - w3c recommendation 16-november-1999. Technical Report REC-xpath-19991116, World Wide Web Consortium, November 1999. See: http://www.w3.org/TR/1999/REC-xpath-19991116.

[Cla99] James Clark. XSL transformations (XSLT) 1.0 - w3c recommendation 16-november-1999. Technical Report REC-xslt-19991116, World Wide Web Consortium, November 1999. See: http://www.w3.org/TR/1999/REC-xslt-19991116.

[Com94] CDIF Technical Committee. Cdif framework for modeling and extensibility. Technical Report EIA/IS-107, Electronic Industries Association, jan 1994. See: http://www.cdif.org/.

[DLT00] Stéphane Ducasse, Michele Lanza, and Sander Tichelaar. Moose: an extensible language-independent environment for reengineering object-oriented systems. In *Proceedings of the Second International Symposium on Constructing Software Engineering Tools (CoSET 2000)*, June 2000.

[DTS99] Serge Demeyer, Sander Tichelaar, and Patrick Steyaert. FAMIX 2.0 - the FAMOOS information exchange model. Technical report, University of Berne, August 1999.

[Fre00] Michael Freidig. XMI for FAMIX. Informatikprojekt, University of Berne, June 2000.

[Gro97] Open Group. DCE 1.1: Remote procedure call. Technical Report C706 August 1997, The Open Group, August 1997. See: http://www.opengroup.org/onlinepubs/009629399/.

[Gro99] Object Management Group. Meta Object Facility (MOF) specification. Technical Report MOF V1.3 RTF, Object Management Group, September 1999.

[LB99] Håkon Wium Lie and Bert Bos. Cascading style sheets, level 1 - w3c recommendation 17 dec 1996, revised 11 jan 1999. Technical Report REC-CSS1-19990111, World Wide Web Consortium, January 1999. See: http://www.w3.org/TR/REC-CSS1-961217.

[NTD98] Oscar Nierstrasz, Sander Tichelaar, and Serge Demeyer. CDIF as the interchange format between reengineering tools. In *OOPSLA'98 Workshop on Model Engineering, Methods and Tools Integration with CDIF*, October 1998.

[TDD00] Sander Tichelaar, Stéphane Ducasse, and Serge Demeyer. FAMIX: Exchange experiences with CDIF and XMI. In *Proceedings of the ICSE 2000 Workshop on Standard Exchange Format (WoSEF 2000)*, June 2000.

[XMI98] XML Metadata Interchange (XMI). Technical Report OMG Document ad/98-10-05, Object Management Group, February 1998.