Towards an XMI–based Model Interchange Format for Graph Transformation Systems

Dániel Varró Gergely Varró
András Pataricza
Budapest University of Technology and Economics
Department of Measurement and Information Systems

August, 2000

Abstract

Tools developed to demonstrate the practical applicability of the mainly theoretical foundations of graph transformation systems are of immense importance. Therefore, the interaction (moreover, integration) of these tools is a major challenge for the graph transformation (GraTra) community in order to increase the efficiency of international (academic or industrial) research.

A first step towards integration is a standardized, common model interchange format providing a uniform graph description and rule representation that is capable of handling the most fundamental concepts of graph transformation. A potential candidate is the novel standard the web, the Extensible Markup Language (XML), which allows the interchange of models in a distributed environment (i.e. Internet).

In this report, we demonstrate on several examples that

- a common, XML–based GraTra model interchange format should be constructed from a metamodel following the concepts of the Meta Object Facility standard;
- XMI (XML Metadata Interchange) is the most suitable XML–based language for such a format;
- the proposed XMI format can be derived automatically from the corresponding GraTra metamodel.

Chapter 1

Major Modelling Concepts

1.1 Introduction

Traditionally, theoretical research fields under an extensive development in practical applications (like graph transformation systems themselves) are characterized by a rapid tool development of various complexity (from simple demonstrations to complex systems with visual language support).

As each of these tools (made by individual research groups) has its own speciality (and the loss of generality as side effect); they differ from each other not only in the applied graph transformation approach (e.g. single pushout [4] or double pushout [3]) but in the underlying data structures and rule representations as well.

In addition, the number of international research projects in the EU grew at an immense rate in the last decade providing financial support for academic areas of practical relevance. Such international projects (e.g. Appligraph [2] and Getgrats [5] in the fields of graph transformation) with different research groups from several countries require a good coordination between the partners. However, this coordination also necessitates the efficient integration and/or interaction of individually developed tools.

The starting point of such a wide interaction is a common standardized exchange format (including the most fundamental concepts) intended to serve as an underlying data structure for software modules of different tools.

Moreover, it is desirable to construct such a format that supports the distributed development of tools and the interchange of tool data via the Internet. As a result of its immense development and flexibility, the attention was turned to the novel, structured language of the web, the Extensible Markup Language (XML). XML combines the advantages of its predecessors (the simplicity of HTML and document structure description of SGML) into an easily parsable and verifiable language which is desired to play a major role in the next generation of Internet applications.

As a result of its growing relevance, industrial and academic research communities set up committees to agree on an XML–based interchange format of their field.

- The semantics of the Unified Modeling Language (UML) was defined by means of a meta-model from which an XML (or rather XMI) based description can be derived [9].

- At the Petri 2000 meeting, the Petri Net community settled on the major principles of a standard data exchange format for Petri Nets.

During her panel statement at GraTra 2000 [11], G. Taentzer lanced an initiative for the development of such a standard for graph transformation in order to achieve that peripheral tools (graph editors, rule analyzers) could independently be developed using the functionality of a common kernel machine. As a result, graph transformation tools would be easier to connect to each other and to other software applications. She also outlined potential future projects in the direction of a common programming interface (e.g. CORBA).

In the current report, we introduce a very first attempt for a common model interchange format for graph transformation systems based on the powerful metamodelling standards of Meta Object Facility (MOF) and XML Metadata Interchange (XMI), the special dialect of XML — both developed by the Object Management Group (OMG). Following the MOF concepts, an abstract metamodel of graph transformation is defined by a visual language which is a subset of UML, thus, its description is easy to understand both in academic and industrial research communities. Moreover, the corresponding XMI format can automatically be derived from a well–formed MOF metamodel.

1.1.1 The Structure of the Report

Our report is structured as follows.

- The rest of Chapter 1 summarizes the most fundamental issues on MOF metamodelling.
- Chapter 2 gives the reader a short introduction on XML, and a more detailed description on XMI containing several examples.
- In Chapter 3, our proposed model interchange format is discussed by constructing a MOF–based GraTra metamodel.
- In addition to this previous topic, Appendix A lists the complete generated GraTra DTD.
- Appendix B contains the complete examples mentioned in Chapter 2.

1.2 Meta Object Facility

The concepts of metamodelling originate in the need for an effective design process of **formal specification and modelling languages**. The large number of similar languages — often supported nowadays by visual diagrams — necessitates a common description language (called **metameta–model** later) that is able to describe the instances of these languages as **sentences**. Traditionally, such a description is based upon a set of production rules called a **grammar**.

However, the sentences of this top–level modelling language (called later as a **model**) can be used for designing a lower level grammar for generating lower level languages hence a **model hierarchy** is available in this sense with several **meta–layers** where the sentences of a higher level language can be used for specifying a lower level language.

This hierarchy can be observed in Figure 1.1, and later demonstrated also in Table 1.1.

1.2.1 Basic MOF Notation

Metadata is a general term for data which in some sense describes a language, and usually defined in the terms of the **Meta Object Facility (MOF)** standard [7].

In this MOF context, the term **model** has a broader meaning than in its general sense (namely, description of something in the real world). Here, *a **model** is a collection of metadata* that is related in the following ways:

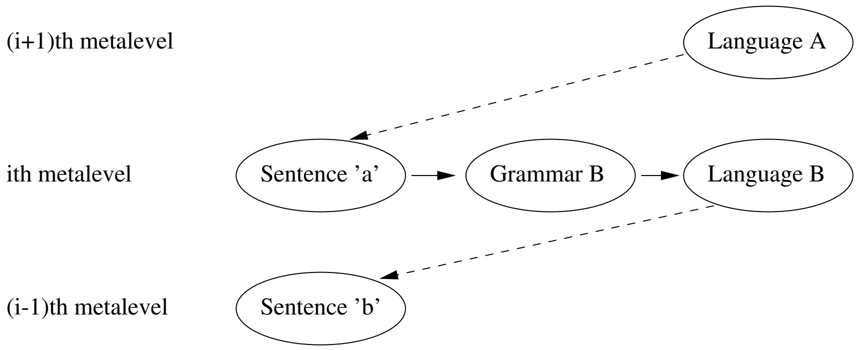

Figure 1.1: Meta–layers in language specification.

- The collection of metadata describes **information**.
- All the metadata conforms to rules controlling its structure and consistency, i.e. it has a common **abstract syntax**.
- The metadata has a **meaning** in a common semantic context.

Metadata itself is a kind of information, hence it can be described by metadata as well. In the MOF terminology, *metadata that describes metadata is called **meta–metadata***, and *the model that consists of a meta–metadata is called a **metamodel***. Metamodels are integrated into a *topmost level meta–metamodel, which is the **MOF Model***, by defining a common syntax for the definition of several metamodel types.

The MOF metadata framework is typically depicted as a four layer architecture that can be observed in Table 1.1.

Meta–level	MOF terms	Examples
M3	meta–metamodel	The MOF Model
M2	meta–metadata metamodel	GraTra metamodel (interchange format)
M1	metadata model	GraTra models, e.g. a graph grammar
M0	data	modelled systems, e.g. a graph

Table 1.1: MOF Metadata Architecture

1.2.2 The MOF Model

*The **MOF Model** is an abstract language for defining MOF metamodels.* Originally, it was developed to provide a general means for describing the language constructs of UML.

Although MOF and UML was designed for different purpose (i.e. metadata versus system modelling), the MOF Model and the core of the UML metamodel are closely related in their modelling concepts (classes for objects of similar structure, associations as relations between these classes, generalization, etc.); therefore, the corresponding UML notation is commonly used for MOF–based metamodels as well. Nevertheless, in order to distinguish between the metamodel elements of UML and the basic constructs of the MOF Model, latter ones printed with capitals.

The main metadata modelling constructs provided by the MOF are the following:

- ***Classes*** *are type descriptions of "first class instance" MOF meta–objects.* Classes defined at the M2 meta–level have their instances at the M1 level. The structural features of Classes can be described at both object and class level by **Attributes** (value holders in an instance of the class), **Operations** (specifying the name and type signature by which the behaviour is invoked). Classes may also inherit their structure and behaviour from other Classes by **Generalization**.

- ***Associations*** *support binary relations between Class instances.* Each Association has two **AssociationEnds** that may specify aggregation semantics and structural constraints on cardinality and uniqueness. When a Class is the type of an AssociationEnd, the Class may contain a **Reference** that allows navigability of the Association's links (i.e. Classes) from a Class instance.

- ***Packages*** *are collections of related Classes and Associations.* Packages can be composed by importing other Packages by inheriting from them. They can also be nested, which provides a form of information hiding.

- ***DataTypes*** *allow the use of non–object types* for Operation parameters and Attributes.

- ***Constraints*** *are used to describe semantic restrictions on elements in a MOF metamodel by defining well–formedness rules for the metadata described by a metamodel.* The **Object Constraint Language (OCL)** [8] is often used as a formal language for expressing constraints.

A semi–formal description of the UML metamodel [9] (containing each of the more than 120 pictorial objects) was released by the OMG using the constructs of the MOF Model as a convincing demonstration that such a rich language as UML can be described by a small subset of its own modelling language.

Examples for a MOF–based metamodel can be found in Section 2.4 when a simple Petri Net metamodel will be introduced. However, these examples lack the demonstration of DataTypes and Constraints in order to emphasize on model structures.

Conclusion This chapter illustrated that the MOF Model provides a natural method to specify and design various metamodels even in different levels of hierarchy. This visual language is a well–defined subset of UML, the widely–known standard of object–oriented software design, thus its modelling concepts can easily be understood in academic research as well as in industrial applications.

Chapter 2

XML Metadata Interchange

2.1 Basic Concepts

The main purpose of **XMI (XML Metadata Interchange)** is to provide an easy interchange of metadata between modelling tools (using UML as their modelling language) and metadata repositories (OMG MOF based) in distributed heterogeneous environments (e.g. Internet) [6].

2.1.1 Introduction

The standardization of models and metamodels interchange (i.e. the invention of XMI format) ha also been proposed by the OMG.

XMI integrates three major industrial standards:

- **MOF**: using the four layered metamodelling architecture for general purpose manipulation of metadata;
- **UML**: representing models and metamodels;
- **XML (eXtensible Markup Language)**: allowing the models to be changed as streams or files.

An XMI compliant model (e.g. a user UML model) usually contains the following:

- The **metamodel** of the problem space (e.g. UML metamodel or GraTra metamodel) as an origin of a uniform interchange format based on MOF.
- The **Document Type Definition (DTD)** of the metamodel (generated automatically from the MOF description).
- The **XMI Document** (in an XML format) describing the user model itself. This document can be syntactically (by an LL(1) parser) and semantically verified (with respect to a given DTD).

Concerning their purpose, DTDs can be regarded as context–free Chomsky grammars describing the XMI documents as their corresponding formal languages.

Meta–level	Metadata	XMI DTDs	XMI Documents
M3	The MOF Model	MOF DTD	
M2	UML Metamodel GraTra Metamodel	UML DTD GraTra DTD	MOF MetaModel Documents
M1	UML Models GraTra Models		UML Models GraTra Documents
M0	Instances		

Table 2.1: XMI and the MOF Metadata Architecture

2.1.2 Main Design Goals

The XMI proposal (of OMG) is planned to meet the following design goals:

Design Goal 1. *Providing a metadata interchange format for any MOF metamodel.*

XMI supports the four layer metadata architecture of OMG in a similar manner as shown in Table 2.1. For instance, a UML model will be encoded against a UML DTD which corresponds to the UML metamodel.

Design Goal 2. *Supporting an automated transfer syntax (i.e. DTD) generation from a given MOF compliant metamodel.*

The classical way of defining a data interchange format is to create a specification document which describes the syntax in BNF or a similar notation and includes a natural language description of non–syntactic aspects. The problem with this approach is that errors and omissions inevitably remain in the specification. The result is that the person responsible for coding import and export modules needs to "interpret" the specification. Divergence in people's interpretations of a specification often leads to unsuccessful data exchange.

The XMI specification is designed to allow automated generation of XML DTDs based on the original MOF specification of a metamodel, hence a faithful reflection of the original metamodel is much more guaranteed.

Design Goal 3. *Following the established principles of XML document design.*

XMI documents are keeping XML's tree–based element structure and its nesting over linkage. This allows generic XML tools to validate documents against a given XML DTD without any hard–wired knowledge of the validity rules for the document.

Design Goal 4. *Supporting the interchange of incomplete models (model fragments).*

The closure of an entire model often consists of much more model elements than are required by a consumer. A consumer may already have many of the elements, or alternatively may have no interest in them. In these circumstances, the production, transmission and consumption of redundant or needless metadata can be a substantial burden to all parties.

Design Goal 5. *An XMI model need not be fully validated as a precondition for metadata interchange.*

Only syntactic correctness and a minimum set of semantic constraints are required since the requirement of full well–formedness would be too restrictive. Ideas often need to be shared before all the details are elaborated.

Design Goal 6. *Supporting versions of models.*

The XMI proposal allows model and metamodel version information to be included in the XMI header. However, it is up to the producers and consumers of XMI streams to manage the allocation of version numbers, and to cope with compatibility between versions and model life cycles.

Design Goal 7. *Allowing the extension of standard models by non–standard model properties*

An XMI document consists of two parts. The first part contains metadata that conforms to a particular MOF metamodel. The second part contains additional metadata that is not described by the base metamodel. This part may have multiple sections, each corresponding to the model extension made by a particular tool (i.e. each tool of different vendors may contain non–standard model attributes as well, such as attributes for graphical representation).

2.2 An Introduction to XML

2.2.1 The Roots of XML

Today, **HTML (HyperText Markup Language)** is the predominant language for expressing web pages, although it has the major disadvantage that HTML tags express presentation rather than semantic information.

The more powerful **SGML (Standard Generalized Markup Language)** separates view from content and data from metadata, but due to its complexity, and the complexity of the tools required, it has not achieved widespread uptake.

XML, the Extensible Markup Language, is a new document format designed by the World Wide Web Consortium (W3C) to bring structured information to the Web. The XML document format embeds the content within HTML–like tags that express the structure. Presentation information is kept in distinct style information format, written mainly in **XSL (Extensible Style Language)**.

XML provides the ability to express rules for the structure (i.e. grammar) of a document. These two features allow automatic separation of data and metadata, and allow generic tools to validate an XML document against its grammar.

XML is an open, tool and vendor independent standard, with low cost on both user and developer side (e.g. a free XML parser written in Java is available), therefore XML is expected to be the next step in the evolution of the Web.

2.2.2 XML Structuring Concepts

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 specified by DTD rules.

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:

Example 1. *`Label` is a sample XML element with a balanced tag pair (`<Tag>`,`</Tag>`) and with the string content `a sample XML tag`*

```
<Label> a sample XML tag </Label>
```

The content of an element may include other elements which may also contain other elements, etc. 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 provides XML with its tree data structure and is a key architectural feature missing from HTML.

Example 2. *This is a simple XMI document describing a `Player` (football player) in details. It contains three attribute tags (`Name,Number,AvgGoals`).*

(New lines and indentation have no semantic significance in XML. They are included here and later on simply to highlight the structure of the example document.)

```
<Player>
  <Name> F. Puskas </Name>
  <Number> 10 </Number>
  <AvgGoals> 12.2 </AvgGoals>
</Player>
```

XML Attributes In addition to contents, an XML element may contain attributes. **XML attributes** are expressed in the opening tag of the element as a list of name–value pairs following the tag name.

Example 3. *A sample attribute of element `Player` is `xmi.label`.*

```
<Player xmi.label="mp1"> </Player>
```

XML defines a special attribute, the ID (identifier), which can be used to attach a unique identifier to an element within a document context. 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.

Document Type Definitions An **XML DTD** (which is responsible for the syntax of a document) defines the different kinds of elements that can appear in a valid document, and the patterns of element nesting that are allowed.

Example 4. *A DTD for the football player example above could contain the following declaration indicating that a `Player` must contain each of the `Name, Number`, and `AvgGoals` elements.*

```
<!ELEMENT Player (Name, Number, AvgGoals)>
```

The declaration for an element can have a more complex grammar, including multiplicities (zero or 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`.

Example 5. *The following DTD component specifies that every `Player` element has an optional `xmi.label` XML attribute and that the `xmi.label` consists of a character data string: (The `#IMPLIED` directive indicates that the attribute is optional.)*

```
<!ATTLIST Player 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.soc.cer/player.dtd"`.

2.2.3 XML Document Correctness

There are three levels of correctness associated with XML document; 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 conforms to the structure defined by a corresponding 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, it is unnecessary for document decoding, it 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.

2.3 DTD Generation

In this section the process of XMI DTD generation will be sketched. The more complete, semi–formal algorithm can be found in [6], while this paper only describes its main features throughout a running example. This example produces the corresponding DTD for a sample MOF–based **metamodel of Petri Nets** (Figure 2.1) to describe only some major features of Petri Nets.

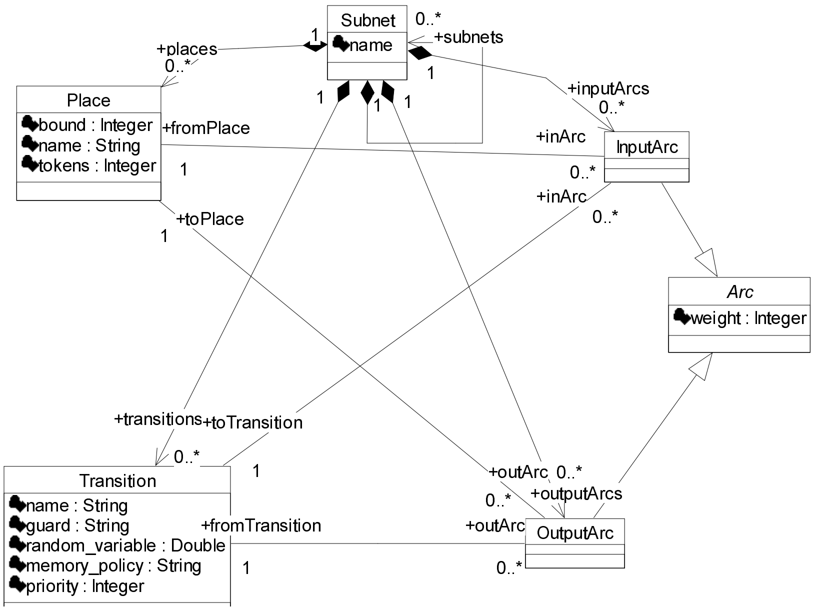

Figure 2.1: A sample MOF metamodel of Petri Nets

A Petri Net model may contain `Places` and `Transitions` linked by `InputArcs` (from a `Place` to a `Transition`) and `OutputArcs` (from a `Transition` to a `Place`). `InputArcs` and

`OutputArcs` inherit their structure from their common superclass `Arc`. In order to support model decomposition, all the elements are nested in a `Subnet`.

Each `Place` has a `name` and may contain a number of tokens (`tokens`) as its Attributes (tokens are not modelled as distinct objects this time). There are two References from a `Place`; one is related to `InputArc` with a Role name `inputArcs` and Multiplicity `0..*`, the other is related to `OutputArc` with a Role name `outputArcs` and Multiplicity `0..*`.

Further metamodel element can be interpreted similarly. Black diamonds at `Subnet` indicate that a `Subnet` may contain other metamodel elements.

This Petri Net metamodel in Figure 2.1 is our own construct only for demonstration purposes. However, a standard Petri Net metamodel is under construction and will soon be announced by the Petri Net community.

2.3.1 The Skeleton of DTD Generation

A complete XMI DTD consists of a fixed DTD content (which is required for any XMI DTD), followed by Package DTD elements for each of the outermost Packages in the MOF–based metamodel. The fixed DTD content can be found in the example of Appendix B.1.

```
1.  DTD ::= FixedContent 2:PackageDTD+
```

Package Packages may contain other Packages, Classes, classifier–level Attributes, Associations and Compositions between Classes. We find a single Package (Petri Net: PN) five Classes several types of Associations and Compositions in our example (Figure 2.1). Please note that the UML–like notation with capital letters refers to the metaclasses of the MOF Model (see Section 1.2.2)

DTD generation for a Package starts with the DTD generation for all of its contents (recursively), followed by the definition of the Package itself (further refinements are indicated by numbers followed by a colon like e.g. `2:`; terminal labels are indicated in single quotation marks).

```
2:  PackageDTD ::= (2:PackageDTD | 3:ClassDTD | CompositionDTD |
                    AssociationDTD | 4:AttributeElementDef)*
                    PackageElementDef
```

CompositionDTDs and AssociationDTDs are rarely used when encoding a typical MOF–based metamodel, since the Classes at both AssociationEnds are responsible for the links between themselves.

Whenever a Package contains another Package, the entire procedure of Package DTD Generation has to be repeated one (containment) level below.

Class The DTD generation for a Class can be performed in 3 steps.

1. Generating a definition for each Attributes (containing the attributes of the superclass as well).
2. Generating a definition for each Reference (role of each Association or Compositions) of the Class.
3. Generating a definition for the Class itself.

```
3:  ClassDTD ::= (4:AttributeElementDef | ReferenceElementDef)*
                 5:ClassElementDef
```

Attribute An AttributeElementDef is the XML element definition for an Attribute. It gives the name and type of the Attribute. The Attributes inherited from the superclass of the Class are not redefined at any level.

```
4:  AttributeElementDef ::=
              '<!ELEMENT' AttribName AttribContents'>'
```

Here `AttribName` stands for the name of the Attribute while `AttribContents` is usually equal to the string `'(#PCDATA | XMI.Reference)*'`.

Reference The generation of a `ReferenceElementDef` is very similar to the generation of `AttributeElementDef`. The only difference is in its `AttribContents`, which contains all the possible types or classes that this reference role can point to (i.e. all the subclasses in the inheritance tree) and the corresponding multiplicity of the role.

Multiplicity The encoding of MOF Multiplicities (indicated in Table 2.2) are not trivial. As XMI should also support the interchange of model fragments, the corresponding DTDs for complete and partial models are slightly different. The importance of generating more than one DTD for the same metamodel comes from the requirement that partial models should also be verified against a DTD for incomplete models.

MOF Model	Complete XMI DTD	Partial XMI DTD
1		?
0..1	?	?
0..*	*	*
1..*	+	*
k..*	+	*

Table 2.2: Encoding MOF Model multiplicities in XMI

- In case of **complete XMI models**, the default multiplicity is *"exactly one"* (`1` in the MOF Model) therefore it is not indicated explicitly (blank field).
- *"At most one"* (`0..1`) is related to a question mark (`"?"`).
- *"Zero or more"* (`0..*`) is identical to `"*"`.
- However, we lose information in case of *"at least k"* semantics in the MOF Model, since both *"at least one"* (`1..*`) and *"at least k"* (`k..*`) is related to `"+"`.

In case of **partial (incomplete) XMI models**, only two different notations are used as an incomplete model may violate the lower bounds of a multiplicity (i.e. the model is incomplete because of a missing mandatory element) but must not violate the upper bounds (since a violated upper bound in an incomplete model also violates the upper bounds of a complete model).

- Therefore *"at most one"* and *"exactly one"* is related to `"?"`.
- *"Zero or more"*, *"at least one"* and *"at least k"* is encoded to `"*"`.

Example 6. *This example contains the sample element definitions for all the Attributes of Class* ***'Place'*** *(in Figure 2.1) supposing that Classes* ***'InputArc'*** *and* ***'OutputArc'*** *will be declared later on. Please note that a* ***Place*** *can be attached to several (possible none)* ***InputArcs*** *and* ***OutputArcs*** *as shown by the corresponding multiplicities.*

```
<!ELEMENT PN.Place.bound (#PCDATA | XMI.reference)* >
<!ELEMENT PN.Place.name (#PCDATA | XMI.reference)* >
<!ELEMENT PN.Place.tokens (#PCDATA | XMI.reference)* >
<!ELEMENT PN.Place.inArc (PN.InputArc)* >
<!ELEMENT PN.Place.outArc (PN.OutputArc)* >
```

Class (continued) Finally, after having defined all the attributes, we described how the pre–defined components can build up the definition of the Class itself. At this point, all the attributes of its superclasses have to be collected and inserted before the own attributes of the Class are declared. The Attributes are followed by a special element (`XMI.Extension`) and the definitions of roles (References, Associations and Compositions).

An additional (uniform) `ATTLIST` declaration for the Class can also be found as a final step.

```
5:  ClassElementDef ::=
          '<!ELEMENT' ClassName ClassContents '>'
          '<!ATTLIST' ClassName ClassAttListItems '>'
```

The following examples demonstrate the encoding process for either complete or incomplete models.

Example 7. *The definition of Class* ***'Place'*** *in Figure 2.1 for* ***complete XMI models****. Class Attributes are listed first (****PN.Place.bound, PN.Place.name, PN.Place.tokens****) followed by a mandatory* ***XMI.extension**** *attribute. Finally, the Role names of References are printed (****PN.Place.inArc*, PN.Place.outArc*****).*

```
<!ELEMENT PN.Place (PN.Place.bound, PN.Place.name,
                    PN.Place.tokens, XMI.extension*,
                    PN.Place.inArc*, PN.Place.outArc*)? >
<!ATTLIST PN.Place
            %XMI.element.att;
            %XMI.link.att;
```

Example 8. *The definition of Class* ***'Place'*** *for* ***incomplete XMI models****. Class Attributes are listed first (****PN.Place.bound, PN.Place.name, PN.Place.tokens****) with a "?" multiplicity, followed by a mandatory* ***XMI.extension**** *attribute. Finally, the Role names of References are printed (****PN.Place.inArc*, PN.Place.outArc*****).*

```
<!ELEMENT PN.Place (PN.Place.bound?, PN.Place.name?,
                    PN.Place.tokens?, XMI.extension*,
                    PN.Place.inArc*, PN.Place.outArc*)? >
<!ATTLIST PN.Place
            %XMI.element.att;
            %XMI.link.att;
```

After having defined all the contained Classes of a Package, the definition of the Package can also be added. This definition resembles to an ordinary Package definition with the well–known `ELEMENT` and `ATTLIST` tags.

Example 9. *The definition of the Package 'PN' indicates that PN may contain an arbitrary amount of Classes **PN.Place**, **PN.InputArc**, **PN.Transition**, **PN.OutputArc**, **PN.Arc** and **PN.Subnet**.*

```
<!ELEMENT PN ((PN.Place | PN.InputArc | PN.Transition | PN.OutputArc |
             PN.Arc | PN.Subnet)*) >
<!ATTLIST PN
           %XMI.element.att;
           %XMI.link.att;
>
```

The complete **Petri Net DTD** is listed in Appendix B.1.

2.4 XMI Document Generation

Most metamodels are characterized by a composition hierarchy. Modelling elements of some type are composed of other modelling elements. A UML model for instance is composed of classes, use cases, packages, etc. This sort of composition is defined in metamodels using the MOF's composite form of Association. This composition must be strict containment – an element cannot be contained by multiple composition in order to keep the strict tree structure of XML.

In order to support models and model fragments as compositions, XMI provides XML document production by object containment. In other words, all the attributes and references of an object are grouped inside the tag containing the definition of the object.

Our running example (in Figure 2.2) describes a simple **Petri Net model**: a subnet of a transition and places (one of them is embedded into another subnet).

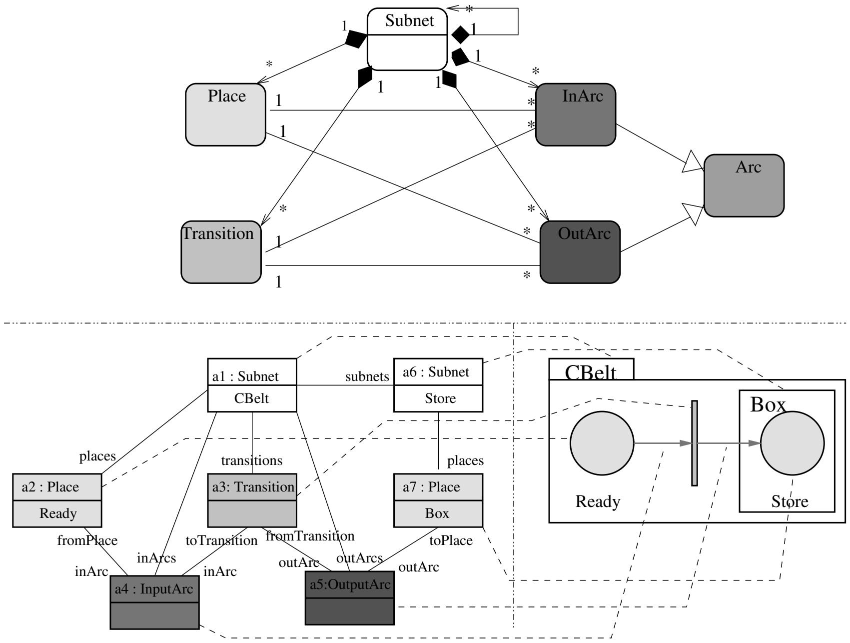

Figure 2.2: Objects forming the sample Petri Net model

The fact that the Petri Net model is a well–formed model of the Petri Net metamodel (as previously defined in Figure 2.1, only the Classes and Associations are depicted this time in Figure 2.2) is grey–scale encoded.

The metamodel classes at M2 level (i.e. `Place, Transition` etc.), which are also instances of M3 level Classes, are painted to different shades of grey. The original Petri Net objects are also linked to their corresponding model elements by dashed lines (beside shading). For instance, the two `Place` objects (`Ready` and `Box`) are light grey since their class `Place` is light grey as well and the isomorphism is also indicated by two reference lines.

2.4.1 Basic Principles of Document Generation

XMI documents mainly consist of two parts: a header and a content part. Naturally, only the content part has a real importance from a modelling point of view.

Header and content parts are enclosed in the root tag of all XMI documents providing information about the currently used XMI version, which is the `XMI` element itself.

Example 10. *The root element of an XMI document is a tag called* ***`XMI`****. One of its attributes is called* ***`xmi.version`***

```
<XMI xmi.version='1.0'>
</XMI>
```

The document generation is proceeded in a hierarchical way, starting from the root object (topmost container object). Each object and attribute contained by another object appear between the **begin** and **end tag** of the container instance. **Reference links** (like associations and unlike compositions) between objects are expressed by special unique object identities.

XML prologue Each generated XML document begins with a **prologue (header tag)** and the standard enclosing XML element's start tag. A possible header tag may be similar to the following:

Example 11. *A sample header tag of an XMI document consists of an* ***`XMI.header`*** *tag. Possibly, it may contain an* ***`XMI.metamodel`*** *element with an* ***`xmi.name`*** *and an* ***`xmi.version`*** *attributes.*

```
<XMI.header>
  <XMI.metamodel xmi.name = 'PetriNet' xmi.version = '0.1'/>
</XMI.header>
```

The importance of an XML prologue originates in the model management issues between different modelling tools of different vendors. A user may get information about the owner, the documentation, or the version of a model, and tool specific extensions of the metamodel standard are also often described here.

At the current phase of a creating common model interchange format of graph transformation systems, the prologue should be omitted probably (to improve clarity), however, we decided to include it in the report in every complete DTD for XMI compatibility reasons.

Object identifiers in XMI Model elements must often have a unique identifier, similarly to primary keys used in databases. There are three attributes in XMI (defined in the mandatory part of XMI DTDs described in Appendix B.1) aiming to provide such a unique descriptor for each XML element. The values of these attributes are used later for creating references to other XML elements. The three attributes differ from each other in the scope of uniqueness (locally or globally).

xmi.id XML semantics require the values of this attribute to be unique within an XML document; however, the value is not required to be globally unique. (In our running example, we mainly use this sort of identification.)

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

xmi.uuid The purpose of this attribute is to provide a (standardized) universally unique identifier (UUID) for an XML element. The values of this attribute should by globally unique strings prefixed be the type of identifier, for instance DCE:1234.

Object linking in XMI XMI requires the use of several XML attributes to enable XML elements to refer (i.e. to link) to other XML elements using the values of the object identifier attributes. The link attributes act as a union of three major linking mechanism, any one of which may be used at a time.

href A well–known inter–document referencing mechanism (from HTML documents).

xmi.idref This attribute allows an XML element to refer to another XML element within the same document. The value of this attribute should be (in an XMI document) the value of an existing **xmi.id** attribute.

xmi.uuidref This attribute provides a mechanism for referring to another XML element in any document all around the world by using a UUID specified in the **xmi.uuid** attribute of another XML element.

2.4.2 Skeleton of Document Generation

After having completed the header section of an XMI document we may focus on its contents embraced between an `xmi.content` tag. All the relevant contents are inside this predefined tag.

Example 12. *The XMI tag containing the user models is embraced in* ***`XMI.content`*** *element.*

```
<XMI.content>
</XMI.content>
```

Start and end tag of an object Document generation is based on element containment in order to keep the strict tree structure. The contained collection of elements is listed between the start and end tag of the container element.

Document generation for the actual model starts from the root object. For each object, including the root object, the element start tag is generated from the object's metaclass name.

Example 13. *The start tag of the sample Petri Net model of Figure 2.2 is the outermost container* ***`Subnet`*** *object, which is* ***`'a1'`*** *in our case.*

```
  <PN.Subnet xmi.id='a1'>
```

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

It must be noted that all names in XMI are fully qualified, based on the MOF description of their metamodel. The name of the item is formed by the sequence of containments and compositions, starting at the outermost package of the metamodel and separated by dots (for instance, `PN.Subnet` in our previous example) in order to help maintain the strict tree structure of XML.

Similarly to HTML, each start tag has to be completed with an end tag as well after all the necessary contents have been written out.

Example 14. *The corresponding end tag for the previous start tag (`PN.Subnet`).*

```
</PN.Subnet>
```

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

Next the attribute value is written out in XML. In our example, the name attribute is a simple string, hence no further preparations are needed at this level.

Finally, the attribute is completed with the corresponding end tag. The following example demonstrates the attribute generation for the root subnet of our running example (Figure 2.2).

Example 15. *Generating the attribute `name` for the object `Subnet` belonging to the package PN.*

```
<PN.Subnet.name>
  CBelt
</PN.Subnet.name>
```

Other attributes of the object are generated in a similar way. However, there is an exception in case of enumeration and boolean data types. Supposing that Subnet has the boolean attribute `isTrue` its value is written using the predefined `xmi.value` tag to increase XML parser validation.

Example 16. *Generating a fictive Boolean (enumeration) attribute for the object `Subnet` of package PN.*

```
<PN.Subnet.isTrue xmi.value='true'/>
```

Object references The attribute generation is followed by the Subnet object's references. A reference (in the sense of the MOF Model) provides the object's navigability to linked objects. XMI considers references to be of two different types and treats them differently.

- *A **normal reference**: an object linked to another via a link defined in the metamodel as having an aggregation other than composite.*
- *A **composite reference**: an object linked to another via a link defined in the metamodel as a composite association, with the composite end corresponding to link end of the composite object.*

It can be considered as a rule of thumb that in XMI, all of the normal references should be written out before the composite references.

As our example does not contain any normal references at this point (at the root object), it will be continued by a composite reference block.

Example 17. *Continuing our running example of Figure 2.2, Class* ***`Subnet`*** *has a composite reference* ***`places`*** *to the* ***`Place`*** *objects contained. Thus the complete definition of* ***`Place`*** *object* ***'a2'*** *is embedded in the* ***`Subnet`*** *element.*

On the other hand, ***`Place`*** *object* ***'a2'*** *(in addition to its attribute* ***`name`****) has a normal reference relating itself to* ***`InputArc`*** ***'a4'****. This reference is indicated by a contained* ***`InputArc`*** *element with a single* ***`xmi.idref`*** *attribute.*

```
<PN.Subnet.places>
  <PN.Place xmi.id='a2'>
    <PN.Place.name>
      Ready
    </PN.Place.name>
    <PN.Place.inArc>
      <PN.InputArc xmi.idref='a4'/>
    </PN.Place.inArc>
  </PN.Place>
</PN.Subnet.places>
```

The `Ready` Place contained by the `CBelt` Subnet is composed of an attribute (`name`) and a normal reference (`inArc`) to an InputArc (as shown in Figure 2.2). Normal references are written out by using the `xmi.idref` attributes for linking objects. Please note that the InputArc object itself is not written at this point, which emphasize a major difference between normal and composite references.

The generation process continues to generate all the composite elements of the root Subnet object in a similar way. The complete XMI document of our running example (the simple Petri Net model of Figure 2.2) is kept together with the Petri Net DTD in Appendix B.

Conclusion As a conclusion, the XMI standard of OMG may become a crucial factor in the integration of different modelling tools of different vendors. XMI is based upon the four–layer MOF metamodelling concept, strictly distinguishing between metamodels and models (instances).

Furthermore, regarding XMI documents as ordinary XML documents, user–created models can be distributed via the Internet. Document verification has become much more easier as well, using special Chomsky–like formal grammars, i.e. DTDs.

Chapter 3

A Model Interchange Format for Graph Transformation Systems

In the panel discussion at GRATRA 2000, G. Taentzer expressed [11] that a common interchange format is badly needed to support model interchange between graph transformation tools.

This chapter provides our proposed **GraTra metamodel** (and the derived **GraTra DTD** in Appendix A) for such a format based on the metamodelling techniques of Chapter 1 and the foundations of XMI in Chapter 2.

3.1 Supported Graph Transformation Features

As the DTD generation is automated with respect to a given MOF metamodel description, only the underlying MOF–based GraTra metamodel is discussed in details in the current report. As being our first proposal in the field, the most common structures of graph transformation were attempted to be included in the metamodel. Therefore our GraTra metamodel focuses on the underlying data structure and the issues of visualization are not supported directly. As visualization is highly tool–dependent, it is better to describe it by the means of XMI extensions.

Furthermore, some of our constructs (e.g. relation, condition) are defined in a very high abstraction level, which necessitates further metamodel refinements in the future. The main reason for this "liberal modelling" originates in the main goal of the report, i.e. to introduce the concepts of MOF metamodelling and XMI documents for developing a common interchange format for graph transformation systems.

As a result, the GraTra metamodel is able to cope with the following:

- Graph structure
  - typed, and attributed graphs (types and attributes associated to model elements like nodes and edges),
  - hypergraphs (graphs with edges linking more than two nodes at a time),
  - hierarchical graphs (where an entire graph may be related to a node or edge on a higher abstraction level),
  - variables and instances of graphs (to distinguish between graph instances and graph schemata),
- Graph transformation approach

- traditional graph transformation rules (following their definition in [1])
- the triple graph grammar approach (e.g. [10]) used for bi–directional visual language translation
- our model transformation approach (see e.g. [12]) with rules structured as Left–Right–Source–Target sides (called *model transformation rules* or *LRST rules* in the sequel)

- Graph transformation (GT) system (defined in [1])
  - GT system (set of GT rules)
  - graph grammars (the extension of Chomsky grammars)
  - transformation units (a method for modular construction of rules and an extended control flow specification)

- Mathematical extensions (at very high abstraction level)
  - relations, functions,
  - conditions,

Please note that a common GraTra metamodel may be independent from the applied graph transformation method (e.g. single pushout, double pushout) as only the underlying data needs to be changed. This requirement naturally includes the *structure* of rules or the description of the system but excludes the *process* of a transformation step.

3.2 Defining the GraTra Metamodel

3.2.1 Packages in the GraTra Metamodel

As the MOF standard allows a well–structured, modular construction of metamodels, the elements of the GraTra metamodel are discussed in this chapter in their container contexts (i.e. their packages).

The GraTra metamodel is divided into six Packages following the main graph transformation concepts and notations.

1. **Core**: Containing the basic elements like Nodes, Edges, Attributes, etc. and their abstract superclasses
2. **Graph**: The definition of graphs, hypergraphs, hierarchical graphs.
3. **GTRule**: Graph transformation rules and their extensions for visual translation and model transformation.
4. **GTSystem**: Graph transformation systems and graph grammars,
5. **TransUnit**: The metamodel of transformation units,
6. **Math**: And last but not least, major mathematical meta–structures, like relations, functions and conditions.

Figure 3.1 summarizes these Packages together with their dependencies (please note that all the figures describing the GraTra metamodels are listed at the end of the section).

3.2.2 Core Metamodel Elements

Package Core (Figure 3.2) collects the main components that will be helpful to construct different sort of graphs. The model hierarchy starts from an abstract superclass (`ModelElement`). Each `ModelElement` may be a variable or an instance as indicated by the boolean attribute `isVariable`. (Emphasized elements with capital letters refer to the GraTra metamodel Classes and not to the MOF Model itself.)

Two subclasses are inherited from `ModelElement`: the abstract `TypedElement` and the `Type`, which can be instantiated. The relationship between them indicates that each ancestor of a `TypedElement` may have a `Type`.

Another three Classes are inherited from `TypedElement`, namely, `Attribute, Argument` and `GraphElement`, which is the superclass of `Nodes` and `Edges`. An `Attribute` have a `name` and a `value`, and, naturally, the inherited relationship with `Types`. `Argument` has a single attribute (`order`) for prescribing a specific order between `Arguments`.

3.2.3 Graph Metamodel Elements

As hypergraphs (with several source and target nodes) are a generalization of ordinary graphs, our GraTra metamodel uses hypergraphs as the default graph model (Figure 3.3) and ordinary graphs are specially constrained graph instances.

A `Graph` may contain several arbitrarily connected `Nodes` and `Edges`. `Nodes` can be reached from an `Edge` via the attributes `fromNode` and `toNode`. An `Edge` is constrained to have at least one source and target `Node` by the corresponding multiplicities. Both `Nodes` and `Edges` may have additional `Attributes`.

Hierarchical graphs (`HierGraph`, Figure 3.4) are `Graphs` where each `Node` may contain a `Graph`. The attribute `parent` of a `HierGraph` refers to the the container `Node` while the subgraph ("sub" in a hierarchical sense) can be reached by the optional `child` attribute of a `Node`.

Naturally, it is possible to extend the GraTra metamodel for the case when edges may contain graphs as well.

3.2.4 Mathematical Structures in GraTra Metamodel

Mathematical structures are also required in the GraTra metamodel since, for instance, graph transformation rules contain two relations and an application condition. For that purpose, relations on graphs (`Relation`) and `Conditions` are introduced in the `Math` Package.

Please note that the word Relation refers in the report to a set of concrete relation instances (e.g rows in a relational database table). This ambiguity is indicated by `Relation` and `RelInstance`. A `RelInstance` may have several `Arguments`, which has exactly one value (usually a `GraphElement`).

3.2.5 Graph Transformation Rule Elements

In this first release of GraTra metamodel, three types of graph transformation rules are handled (as indicated by inheritance in Figure 3.6). The original graph transformation rule (`GTRule`, Figure 3.7) is strictly based on its definition in [1].

Every transformation rule (`Rule`) may have a priority and can be executed in a parallel or sequential way. A graph transformation rule consists of a left–hand side graph (`LHS`), a right–hand side graph (`RHS`), an interface graph `interface`, and, additionally, two relations (`embRel`, `glueRel`) and an application condition (`appCond`).

Triple graph grammar rules (`TGGRule`, Figure 3.8) and our model transformation approach (`SGRule`, Figure 3.9) are also included as two possible extension directions of a core GraTra metamodel, however, these approaches can also be described by the traditional approach.

Triple graph grammar rules consist of a LHS graph, a RHS graph, a correspondence graph, and (beside the application conditions), two relations coupling LHS and RHS with the correspondence graph.

The model transformation approach is composed from four graphs (LHS–Source, LHS–Target, RHS–Source, RHS–Target) and two relations connecting the corresponding LHS–and–RHS, and the source–and–target graph elements.

3.2.6 Modelling Graph Transformation Systems

Both graph transformation systems (`GTSystem`) and graph grammars (`GraphGrammar`) are included in the GraTra metamodel (see Figure 3.10).

Graph transformation systems (`GTSystem`) are a collection of rules (`rules`) while graph grammars `GraphGrammar` are graph transformation systems (inheriting e.g. their rules) extended by an initial graph (`initial`) and terminal type labels (`terminal`).

3.2.7 Modelling Transformation Units

In order to support transformation rule structuring, the concepts of transformation units (`TransUnit`, Figure 3.11) are also defined in the GraTra metamodel, naturally, strictly following its definition in [1].

A transformation unit has a name (as an own Attribute) and may also contain an initial graph (`initial`) and a set of terminal graphs (`terminals`), any type of rules (`rules`), control conditions (`controlCond`) and further imported transformation units (`import`).

3.3 The Graphical GraTra Metamodel

The submodels of the MOF–based GraTra metamodel are listed below. As the graph transformation community is in the very first phase of standardization, our first proposal aiming to provide a general description of graph transformation deals with only the major concepts, therefore, further extensions and refinement will surely be needed (in corporation with the GraTra community).

- Core concepts (Figure 3.2)
- Basic graph concepts (Figure 3.3)
- Hierarchical graphs (Figure 3.4)
- Mathematical concepts (Figure 3.5)
- Different sort of graph transformation rules (Figure 3.6)
  - Graph transformation rules (Figure 3.7)
  - Triple graph grammars (Figure 3.8)
  - Our visual translation approach (Figure 3.9)
- Graph transformation systems (Figure 3.10)

- Transformation units (Figure 3.11)

Please note that relations with black diamonds are composite relations. Multiplicities are always indicated in order to ensure that the automatically generated GraTra DTD is conform with the GraTra metamodel.

A GraTra metamodel element has its corresponding XMI element in the DTD. The list of its attributes is composed of own metamodel Attributes (like e.g. `isVariable` in `ModelElement`) and the Role names of all navigable associations (e.g. `fromNode` in `Edge`).

Finally, Appendix A contains the generated DTD.

3.4 Conclusion

In the current report, a general model interchange format was introduced for graph transformation systems. Our proposal was based on the novel standard of the Object Management Group, namely, the XML Metadata Interchange, which was developed to provide an easy interchange of metadata between modelling tools and repositories in distributed environments.

An XMI–based model interchange format (serving as an XML–based format for graph transformation tools) has several advantages.

- Only a slight knowledge of XML is required for the common format designers as the GraTra DTD is automatically derived from a MOF metamodel of graph transformation.
- The GraTra metamodel is designed by a class diagram–like subset of the Unified Modelling Language (UML), called Meta Object Facility (MOF). Thus, designers formalize their thoughts in the style of the widely–used standard of object–oriented software design.
- A graphical MOF diagram is more comprehensive ("legible") than a purely textual DTD in a cooperative phase of design.
- MOF provide a well–structured design nesting the similar constructs into Packages.
- XMI is strongly related to CORBA hence providing a common programming interface as well.

On the other hand, we must admit that our XMI–based solution may have a couple of disadvantages as well in contrast with a pure XML solution.

- An XMI DTD may contain XML elements that are hardly ever used (e.g. Corba types).
- Long names of XMI elements with several dots are difficult to handle manually.
- More rigorous validity checks may be performed with respect to a pure XML DTD (especially, in case of basic data types).

As a conclusion, the MOF standard may become the standard model description of complex metamodels in various academic and industrial fields. Several sophisticated tools already exist that are capable of producing the corresponding XMI DTD automatically, which reduces the time spent on creating or updating an XML DTD significantly. As a result, the design process of XMI DTDs is characterized by a well–defined semi–formal method (in an analogy with entity–relationship diagrams that rule the design of relational databases).

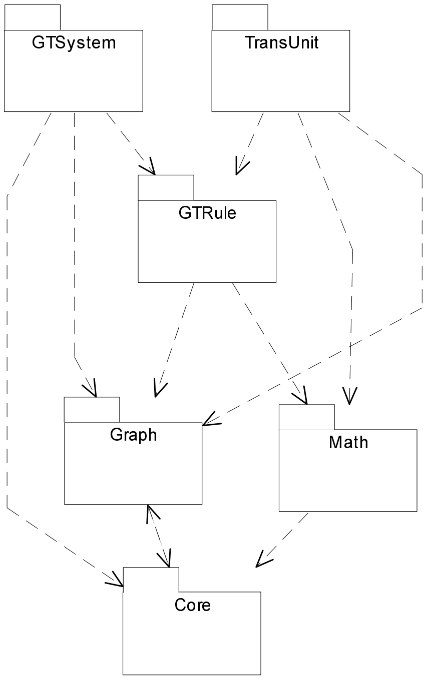

Figure 3.1: Graph Transformation package overview

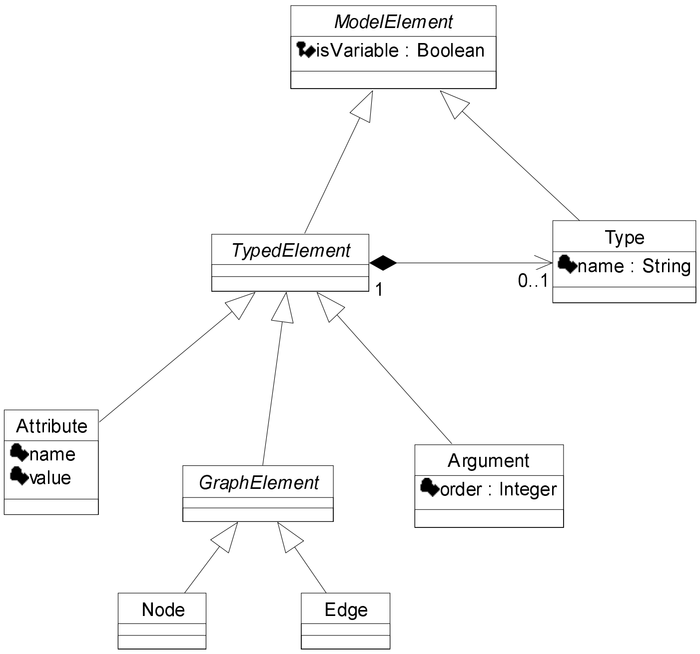

Figure 3.2: Core graph transformation concepts

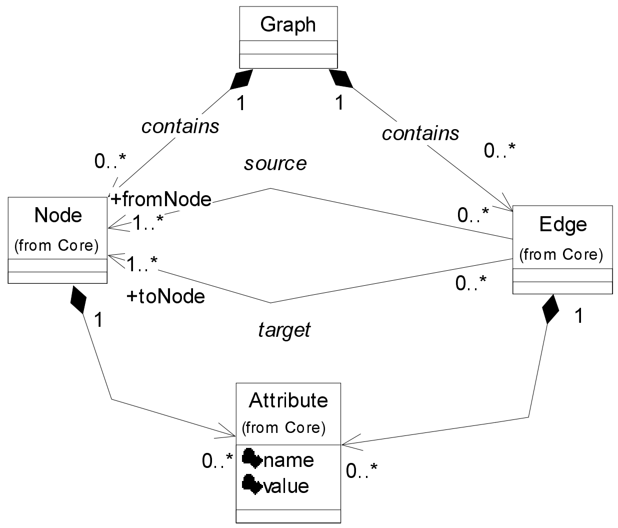

Figure 3.3: Basic graph concept

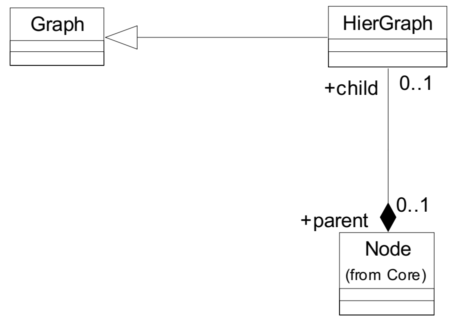

Figure 3.4: Modelling hierarchical graphs

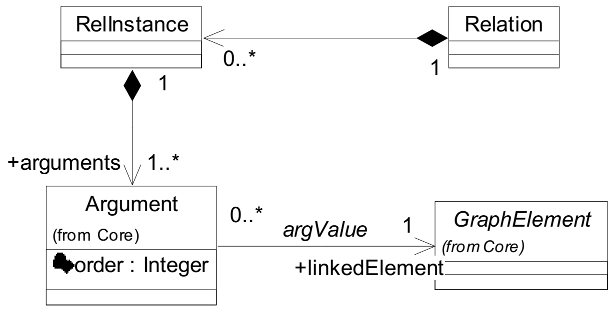

Figure 3.5: Mathematical concepts

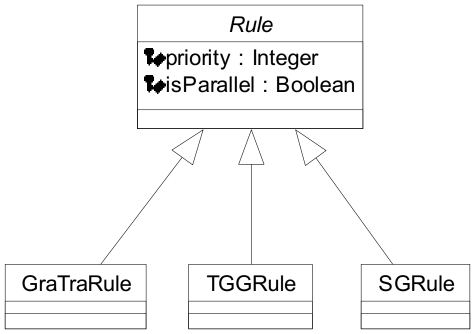

Figure 3.6: Modelling different sort of graph transformation rules

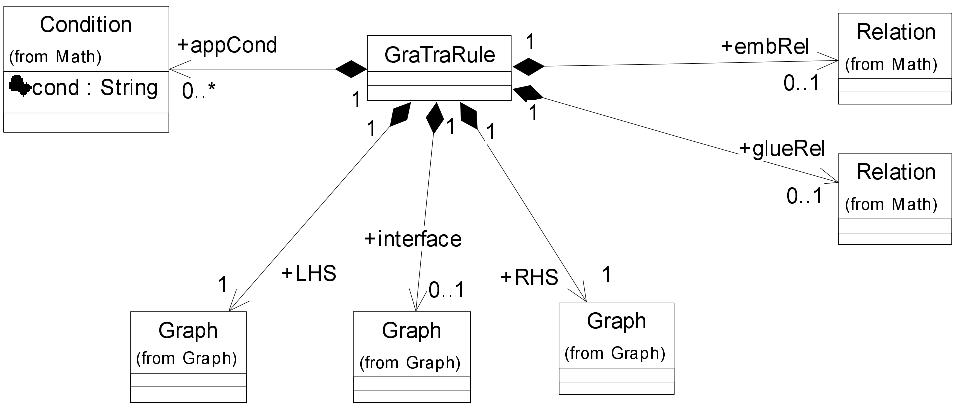

Figure 3.7: Modelling graph transformation rules

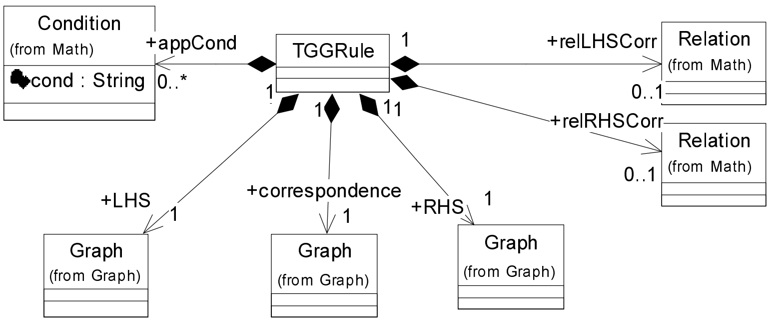

Figure 3.8: Modelling triple graph grammars

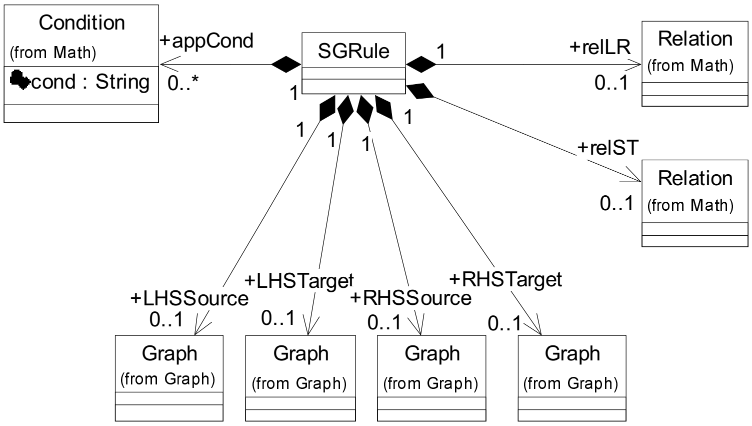

Figure 3.9: Modelling the model transformation approach

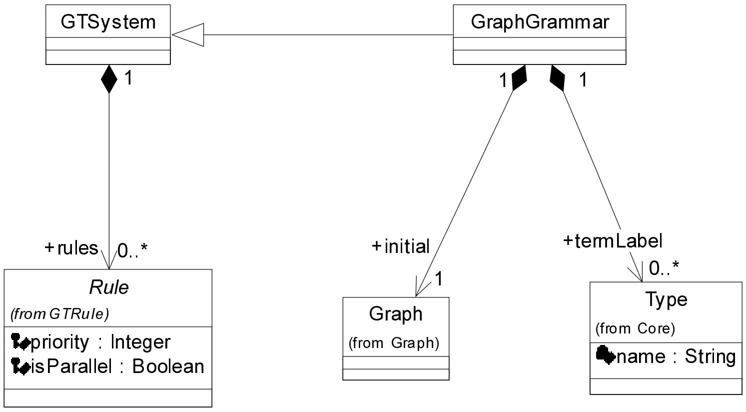

Figure 3.10: Modelling graph transformation systems

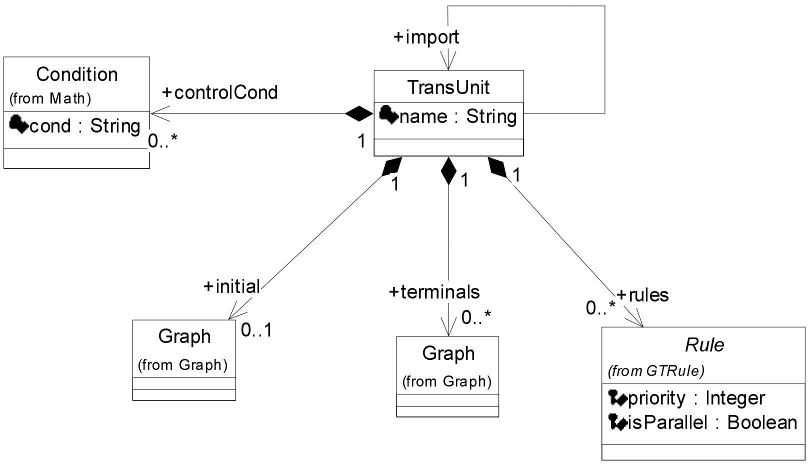

Figure 3.11: Modelling transformation units

Bibliography

[1] M. Andries, G. Engels, A. Habel, B. Hoffmann, H.-J. Kreowski, S. Kuske, D. Plump, A. Schürr, and G. Taentzer. Graph transformation for specification and programming. *Science of Computer Programming*, (34):1–54, 1999.

[2] APPLIGRAPH. An ESPRIT Working Group for the Applications of Graph Transformation. `http://www.informatik.uni-bremen.de/theorie/appligraph/`.

[3] A. Corradini, U. Montanari, F. Rossi, H. Ehrig, R. Heckel, and M. Löwe. *Handbook of Graph Grammars and Computing by Graph Transformations*, volume 1: Foundations, chapter Algebraic Approaches to Graph Transformation — Part I: Basic Concepts and Double Pushout Approach, pages 163–245. World Scientific, 1997.

[4] H. Ehrig, R. Heckel, M. Korff, M. Löwe, L. Ribeiro, A. Wagner, and A. Corradini. *Handbook of Graph Grammars and Computing by Graph Transformations*, volume 1: Foundations, chapter Algebraic Approaches to graph transformation — Part II: Single pushout approach and comparison with double pushout approach, pages 247–312. World Scientific, 1997.

[5] GETGRATS. A TMR Research Network for a General Theory of Graph Transformation Systems. `http://www.di.unipi.it/ andrea/getgrats/`.

[6] Object Management Group. *XML Metadata Interchange*, October 1998.
`http://www.omg.org`.

[7] Object Management Group. *Meta Object Facility Version 1.3*, September 1999.
`http://www.omg.org`.

[8] Object Management Group. *Object Constraint Language Specification Version 1.3*, June 1999.
`http://www.rational.com/uml`.

[9] Object Management Group. *UML Semantics Version 1.3*, June 1999.
`http://www.rational.com/uml`.

[10] A. Schürr. Specification of graph translators with triple graph grammars. Technical report, RWTH Aachen, Fachgruppe Informatik, Germany, 1994.

[11] G. Taentzer. Panel Statement – Theory and practice of graph transformation: From historical roots to visions in the new milleneum. In H. Ehrig and G. Taentzer, editors, *GRATRA 2000, Joint APPLIGRAPH and GETGRATS Workshop on Graph Transformation Systems*, pages 275–276, March 2000.

[12] D. Varró. Automatic transformation of UML models. Master's thesis, Budapest University of Technology and Economics, 2000. `http://domino.inf.mit.bme.hu/biblio.nsf`.

Appendix A

GraTra DTD

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

<!-- XMI Automatic DTD Generation       -->
<!-- Date: Sat Apr 22 19:12:39 CEST 2000 -->
<!-- Metamodel: GraTra1             -->

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

<!-- _______________________________________________________________ -->
<!--                                                                 -->
<!-- METAMODEL:  GraTra1 (own definitions)                           -->
<!-- _______________________________________________________________ -->

<!-- _______________________________________________________________ -->
<!--                                                                 -->
<!-- METAMODEL PACKAGE:  GT                                          -->
<!-- _______________________________________________________________ -->

<!-- _______________________________________________________________ -->
<!--                                                                 -->
<!-- METAMODEL PACKAGE:  GTRule                                      -->
<!-- _______________________________________________________________ -->

<!ELEMENT GT.GTRule.GraTraRule.LHS (GT.Graph.Graph |
                                    GT.Graph.HierGraph)? >
<!ELEMENT GT.GTRule.GraTraRule.RHS (GT.Graph.Graph |
                                    GT.Graph.HierGraph)? >
<!ELEMENT GT.GTRule.GraTraRule.interface (GT.Graph.Graph |
                                          GT.Graph.HierGraph)? >
<!ELEMENT GT.GTRule.GraTraRule.appCond (GT.Math.Condition)* >
<!ELEMENT GT.GTRule.GraTraRule.embRel (GT.Math.Relation)? >
<!ELEMENT GT.GTRule.GraTraRule.glueRel (GT.Math.Relation)? >

<!ELEMENT GT.GTRule.TGGRule.LHS (GT.Graph.Graph | GT.Graph.HierGraph)? >
<!ELEMENT GT.GTRule.TGGRule.correspondence (GT.Graph.Graph |
                                            GT.Graph.HierGraph)? >
<!ELEMENT GT.GTRule.TGGRule.RHS (GT.Graph.Graph | GT.Graph.HierGraph)? >
<!ELEMENT GT.GTRule.TGGRule.appCond (GT.Math.Condition)* >
<!ELEMENT GT.GTRule.TGGRule.relLHSCorr (GT.Math.Relation)? >
<!ELEMENT GT.GTRule.TGGRule.relRHSCorr (GT.Math.Relation)? >

<!ELEMENT GT.GTRule.SGRule.LHSSource (GT.Graph.Graph |
                                      GT.Graph.HierGraph)? >
<!ELEMENT GT.GTRule.SGRule.LHSTarget (GT.Graph.Graph |
                                      GT.Graph.HierGraph)? >
<!ELEMENT GT.GTRule.SGRule.RHSSource (GT.Graph.Graph |
                                      GT.Graph.HierGraph)? >
<!ELEMENT GT.GTRule.SGRule.RHSTarget (GT.Graph.Graph |
                                      GT.Graph.HierGraph)? >
<!ELEMENT GT.GTRule.SGRule.appCond (GT.Math.Condition)* >
<!ELEMENT GT.GTRule.SGRule.relLR (GT.Math.Relation)? >
<!ELEMENT GT.GTRule.SGRule.relST (GT.Math.Relation)? >

<!-- _______________________________________________________________ -->
<!--                                                                 -->
<!-- METAMODEL CLASS:  GraTraRule                                    -->
<!-- _______________________________________________________________ -->

<!ELEMENT GT.GTRule.GraTraRule (GT.GTRule.Rule.priority?,
                                GT.GTRule.Rule.isParallel?,
                                XMI.extension*,
                                GT.GTRule.GraTraRule.LHS?,
                                GT.GTRule.GraTraRule.RHS?,
                                GT.GTRule.GraTraRule.interface?,
                                GT.GTRule.GraTraRule.appCond*,
                                GT.GTRule.GraTraRule.embRel?,
                                GT.GTRule.GraTraRule.glueRel?)? >
<!ATTLIST GT.GTRule.GraTraRule
```

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

<!-- ______________________________________________________________ -->
<!--                                                                -->
<!-- METAMODEL CLASS:  TGGRule                                      -->
<!-- ______________________________________________________________ -->

<!ELEMENT GT.GTRule.TGGRule (GT.GTRule.Rule.priority?,
                             GT.GTRule.Rule.isParallel?, XMI.extension*,
                             GT.GTRule.TGGRule.LHS?,
                             GT.GTRule.TGGRule.correspondence?,
                             GT.GTRule.TGGRule.RHS?,
                             GT.GTRule.TGGRule.appCond*,
                             GT.GTRule.TGGRule.relLHSCorr?,
                             GT.GTRule.TGGRule.relRHSCorr?)? >
<!ATTLIST GT.GTRule.TGGRule
            %XMI.element.att;
            %XMI.link.att;
>

<!-- ______________________________________________________________ -->
<!--                                                                -->
<!-- METAMODEL CLASS:  Rule                                         -->
<!-- ______________________________________________________________ -->

<!ELEMENT GT.GTRule.Rule.priority (#PCDATA | XMI.reference)* >
<!ELEMENT GT.GTRule.Rule.isParallel (#PCDATA | XMI.reference)* >
<!ELEMENT GT.GTRule.Rule (GT.GTRule.Rule.priority?,
                          GT.GTRule.Rule.isParallel?, XMI.extension*)? >
<!ATTLIST GT.GTRule.Rule
            %XMI.element.att;
            %XMI.link.att;
>

<!-- ______________________________________________________________ -->
<!--                                                                -->
<!-- METAMODEL CLASS:  SGRule                                       -->
<!-- ______________________________________________________________ -->

<!ELEMENT GT.GTRule.SGRule (GT.GTRule.Rule.priority?,
                            GT.GTRule.Rule.isParallel?, XMI.extension*,
                            GT.GTRule.SGRule.LHSSource?,
                            GT.GTRule.SGRule.LHSTarget?,
                            GT.GTRule.SGRule.RHSSource?,
                            GT.GTRule.SGRule.RHSTarget?,
                            GT.GTRule.SGRule.appCond*,
                            GT.GTRule.SGRule.relLR?,
                            GT.GTRule.SGRule.relST?)? >
<!ATTLIST GT.GTRule.SGRule
            %XMI.element.att;
            %XMI.link.att;
>

<!ELEMENT GT.GTRule ((GT.GTRule.GraTraRule | GT.GTRule.TGGRule |
                     GT.GTRule.Rule | GT.GTRule.SGRule)*) >
<!ATTLIST GT.GTRule
            %XMI.element.att;
            %XMI.link.att;
>

<!-- ______________________________________________________________ -->
<!--                                                                -->
<!-- METAMODEL PACKAGE:  Graph                                      -->
<!-- ______________________________________________________________ -->

<!ELEMENT GT.Graph.Graph.node (GT.Core.Node)* >
```

```
<!ELEMENT GT.Graph.Graph.edge (GT.Core.Edge)* >
<!ELEMENT GT.Core.Node.attribute (GT.Core.Attribute)* >
<!ELEMENT GT.Core.Edge.attribute (GT.Core.Attribute)* >
<!ELEMENT GT.Core.Node.child (GT.Graph.HierGraph)? >

<!-- ______________________________________________________________ -->
<!--                                                                -->
<!-- METAMODEL CLASS:  Graph                                        -->
<!-- ______________________________________________________________ -->

<!ELEMENT GT.Graph.Graph (XMI.extension*, GT.Graph.Graph.node*,
                          GT.Graph.Graph.edge*)? >
<!ATTLIST GT.Graph.Graph
            %XMI.element.att;
            %XMI.link.att;
>

<!-- ______________________________________________________________ -->
<!--                                                                -->
<!-- METAMODEL CLASS:  HierGraph                                    -->
<!-- ______________________________________________________________ -->

<!ELEMENT GT.Graph.HierGraph.parent (GT.Core.Node)? >
<!ELEMENT GT.Graph.HierGraph (XMI.extension*,
                              GT.Graph.HierGraph.parent?,
                              GT.Graph.Graph.node*,
                              GT.Graph.Graph.edge*)? >
<!ATTLIST GT.Graph.HierGraph
            %XMI.element.att;
            %XMI.link.att;
>

<!ELEMENT GT.Graph ((GT.Graph.Graph | GT.Graph.HierGraph)*) >
<!ATTLIST GT.Graph
            %XMI.element.att;
            %XMI.link.att;
>

<!-- ______________________________________________________________ -->
<!--                                                                -->
<!-- METAMODEL PACKAGE:  Core                                       -->
<!-- ______________________________________________________________ -->

<!ELEMENT GT.Core.TypedElement.type (GT.Core.Type)? >

<!-- ______________________________________________________________ -->
<!--                                                                -->
<!-- METAMODEL CLASS:  TypedElement                                 -->
<!-- ______________________________________________________________ -->

<!ELEMENT GT.Core.TypedElement (GT.Core.ModelElement.isVariable?,
                                XMI.extension*,
                                GT.Core.TypedElement.type?)? >
<!ATTLIST GT.Core.TypedElement
            %XMI.element.att;
            %XMI.link.att;
>

<!-- ______________________________________________________________ -->
<!--                                                                -->
<!-- METAMODEL CLASS:  Node                                         -->
<!-- ______________________________________________________________ -->

<!ELEMENT GT.Core.Node (GT.Core.ModelElement.isVariable?,
                        XMI.extension*, GT.Core.TypedElement.type?,
                        GT.Core.Node.child?, GT.Core.Node.attribute*)? >
<!ATTLIST GT.Core.Node
            %XMI.element.att;
```

```
            %XMI.link.att;
>

<!-- _______________________________________________________________ -->
<!--                                                                 -->
<!-- METAMODEL CLASS:  Edge                                          -->
<!-- _______________________________________________________________ -->

<!ELEMENT GT.Core.Edge.toNode (GT.Core.Node)* >
<!ELEMENT GT.Core.Edge.fromNode (GT.Core.Node)* >
<!ELEMENT GT.Core.Edge (GT.Core.ModelElement.isVariable?,
                        XMI.extension*, GT.Core.Edge.toNode*,
                        GT.Core.Edge.fromNode*,
                        GT.Core.TypedElement.type?,
                        GT.Core.Edge.attribute*)? >
<!ATTLIST GT.Core.Edge
            %XMI.element.att;
            %XMI.link.att;
>

<!-- _______________________________________________________________ -->
<!--                                                                 -->
<!-- METAMODEL CLASS:  Attribute                                     -->
<!-- _______________________________________________________________ -->

<!ELEMENT GT.Core.Attribute.name  >
<!ELEMENT GT.Core.Attribute.value  >
<!ELEMENT GT.Core.Attribute (GT.Core.ModelElement.isVariable?,
                             GT.Core.Attribute.name?,
                             GT.Core.Attribute.value?, XMI.extension*,
                             GT.Core.TypedElement.type?)? >
<!ATTLIST GT.Core.Attribute
            %XMI.element.att;
            %XMI.link.att;
>

<!-- _______________________________________________________________ -->
<!--                                                                 -->
<!-- METAMODEL CLASS:  Type                                          -->
<!-- _______________________________________________________________ -->

<!ELEMENT GT.Core.Type.name (#PCDATA | XMI.reference)* >
<!ELEMENT GT.Core.Type (GT.Core.ModelElement.isVariable?,
                        GT.Core.Type.name?, XMI.extension*)? >
<!ATTLIST GT.Core.Type
            %XMI.element.att;
            %XMI.link.att;
>

<!-- _______________________________________________________________ -->
<!--                                                                 -->
<!-- METAMODEL CLASS:  ModelElement                                  -->
<!-- _______________________________________________________________ -->

<!ELEMENT GT.Core.ModelElement.isVariable (#PCDATA | XMI.reference)* >
<!ELEMENT GT.Core.ModelElement (GT.Core.ModelElement.isVariable?,
                                XMI.extension*)? >
<!ATTLIST GT.Core.ModelElement
            %XMI.element.att;
            %XMI.link.att;
>

<!-- _______________________________________________________________ -->
<!--                                                                 -->
<!-- METAMODEL CLASS:  Argument                                      -->
<!-- _______________________________________________________________ -->

<!ELEMENT GT.Core.Argument.order (#PCDATA | XMI.reference)* >
```

```
<!ELEMENT GT.Core.Argument.linkedElement (GT.Core.GraphElement |
                                          GT.Core.Node | GT.Core.Edge)? >
<!ELEMENT GT.Core.Argument (GT.Core.ModelElement.isVariable?,
                            GT.Core.Argument.order?, XMI.extension*,
                            GT.Core.Argument.linkedElement?,
                            GT.Core.TypedElement.type?)? >
<!ATTLIST GT.Core.Argument
            %XMI.element.att;
            %XMI.link.att;
>

<!-- ______________________________________________________________ -->
<!--                                                                -->
<!-- METAMODEL CLASS:  GraphElement                                 -->
<!-- ______________________________________________________________ -->

<!ELEMENT GT.Core.GraphElement (GT.Core.ModelElement.isVariable?,
                                XMI.extension*,
                                GT.Core.TypedElement.type?)? >
<!ATTLIST GT.Core.GraphElement
            %XMI.element.att;
            %XMI.link.att;
>

<!ELEMENT GT.Core ((GT.Core.TypedElement | GT.Core.Node | GT.Core.Edge |
                   GT.Core.Attribute | GT.Core.Type |
                   GT.Core.ModelElement | GT.Core.Argument |
                   GT.Core.GraphElement)*) >
<!ATTLIST GT.Core
            %XMI.element.att;
            %XMI.link.att;
>

<!-- ______________________________________________________________ -->
<!--                                                                -->
<!-- METAMODEL PACKAGE:  Math                                       -->
<!-- ______________________________________________________________ -->

<!ELEMENT GT.Math.RelInstance.arguments (GT.Core.Argument)* >
<!ELEMENT GT.Math.Function.return (GT.Core.Argument)? >
<!ELEMENT GT.Math.Function.arguments (GT.Core.Argument)* >
<!ELEMENT GT.Math.Relation.relInstance (GT.Math.RelInstance)* >

<!-- ______________________________________________________________ -->
<!--                                                                -->
<!-- METAMODEL CLASS:  RelInstance                                  -->
<!-- ______________________________________________________________ -->

<!ELEMENT GT.Math.RelInstance (XMI.extension*,
                               GT.Math.RelInstance.arguments*)? >
<!ATTLIST GT.Math.RelInstance
            %XMI.element.att;
            %XMI.link.att;
>

<!-- ______________________________________________________________ -->
<!--                                                                -->
<!-- METAMODEL CLASS:  Function                                     -->
<!-- ______________________________________________________________ -->

<!ELEMENT GT.Math.Function (XMI.extension*, GT.Math.Function.return?,
                            GT.Math.Function.arguments*)? >
<!ATTLIST GT.Math.Function
            %XMI.element.att;
            %XMI.link.att;
>

<!-- ______________________________________________________________ -->
```

```
<!--                                                                -->
<!-- METAMODEL CLASS:  Condition                                    -->
<!-- _______________________________________________________________ -->

<!ELEMENT GT.Math.Condition.cond (#PCDATA | XMI.reference)* >
<!ELEMENT GT.Math.Condition (GT.Math.Condition.cond?, XMI.extension*)? >
<!ATTLIST GT.Math.Condition
            %XMI.element.att;
            %XMI.link.att;
>

<!-- _______________________________________________________________ -->
<!--                                                                -->
<!-- METAMODEL CLASS:  Relation                                     -->
<!-- _______________________________________________________________ -->

<!ELEMENT GT.Math.Relation (XMI.extension*,
                            GT.Math.Relation.relInstance*)? >
<!ATTLIST GT.Math.Relation
            %XMI.element.att;
            %XMI.link.att;
>

<!ELEMENT GT.Math ((GT.Math.RelInstance | GT.Math.Function |
                   GT.Math.Condition | GT.Math.Relation)*) >
<!ATTLIST GT.Math
            %XMI.element.att;
            %XMI.link.att;
>

<!-- _______________________________________________________________ -->
<!--                                                                -->
<!-- METAMODEL PACKAGE:  GTSystem                                   -->
<!-- _______________________________________________________________ -->

<!ELEMENT GT.GTSystem.GTSystem.rules (GT.GTRule.Rule | GT.GTRule.SGRule |
                                      GT.GTRule.TGGRule |
                                      GT.GTRule.GraTraRule)* >

<!ELEMENT GT.GTSystem.GraphGrammar.initial (GT.Graph.Graph |
                                            GT.Graph.HierGraph)? >

<!ELEMENT GT.GTSystem.GraphGrammar.termLabel (GT.Core.Type)* >

<!-- _______________________________________________________________ -->
<!--                                                                -->
<!-- METAMODEL CLASS:  GTSystem                                     -->
<!-- _______________________________________________________________ -->

<!ELEMENT GT.GTSystem.GTSystem (XMI.extension*,
                                GT.GTSystem.GTSystem.rules*)? >
<!ATTLIST GT.GTSystem.GTSystem
            %XMI.element.att;
            %XMI.link.att;
>

<!-- _______________________________________________________________ -->
<!--                                                                -->
<!-- METAMODEL CLASS:  GraphGrammar                                 -->
<!-- _______________________________________________________________ -->

<!ELEMENT GT.GTSystem.GraphGrammar (XMI.extension*,
                                    GT.GTSystem.GTSystem.rules*,
                                    GT.GTSystem.GraphGrammar.initial?,
                                    GT.GTSystem.GraphGrammar.termLabel*)?
                                     >
<!ATTLIST GT.GTSystem.GraphGrammar
            %XMI.element.att;
```

```
            %XMI.link.att;
>

<!ELEMENT GT.GTSystem ((GT.GTSystem.GTSystem |
                       GT.GTSystem.GraphGrammar)*) >
<!ATTLIST GT.GTSystem
            %XMI.element.att;
            %XMI.link.att;
>

<!-- _______________________________________________________________ -->
<!--                                                                 -->
<!-- METAMODEL PACKAGE:  TransUnit                                   -->
<!-- _______________________________________________________________ -->

<!ELEMENT GT.TransUnit.TransUnit.controlCond (GT.Math.Condition)* >
<!ELEMENT GT.TransUnit.TransUnit.initial (GT.Graph.Graph |
                                          GT.Graph.HierGraph)? >
<!ELEMENT GT.TransUnit.TransUnit.terminals (GT.Graph.Graph |
                                            GT.Graph.HierGraph)* >
<!ELEMENT GT.TransUnit.TransUnit.rules (GT.GTRule.Rule |
                                        GT.GTRule.SGRule |
                                        GT.GTRule.TGGRule |
                                        GT.GTRule.GraTraRule)* >

<!-- _______________________________________________________________ -->
<!--                                                                 -->
<!-- METAMODEL CLASS:  TransUnit                                     -->
<!-- _______________________________________________________________ -->

<!ELEMENT GT.TransUnit.TransUnit.name (#PCDATA | XMI.reference)* >
<!ELEMENT GT.TransUnit.TransUnit.import (GT.TransUnit.TransUnit)* >
<!ELEMENT GT.TransUnit.TransUnit (GT.TransUnit.TransUnit.name?,
                                  XMI.extension*,
                                  GT.TransUnit.TransUnit.import*,
                                  GT.TransUnit.TransUnit.controlCond*,
                                  GT.TransUnit.TransUnit.initial?,
                                  GT.TransUnit.TransUnit.terminals*,
                                  GT.TransUnit.TransUnit.rules*)? >
<!ATTLIST GT.TransUnit.TransUnit
            %XMI.element.att;
            %XMI.link.att;
>

<!ELEMENT GT.TransUnit ((GT.TransUnit.TransUnit)*) >
<!ATTLIST GT.TransUnit
            %XMI.element.att;
            %XMI.link.att;
>

<!ELEMENT GT ((GT.GTRule | GT.Graph | GT.Core | GT.Math | GT.GTSystem |
              GT.TransUnit)*) >
<!ATTLIST GT
            %XMI.element.att;
            %XMI.link.att;
>

<!ELEMENT GraTra1 ((GT)*) >
<!ATTLIST GraTra1
            %XMI.element.att;
            %XMI.link.att;
>
```

Appendix B

XMI–Based Petri Net Models

B.1 A Sample Petri Net DTD

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

<!-- XMI Automatic DTD Generation       -->
<!-- Date: Mon Apr 10 14:25:22 CEST 2000 -->
<!-- Metamodel: PetriNet             -->

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

<!-- _______________________________________________________________ -->
<!--                                                                 -->
<!-- METAMODEL:  PetriNet (private definitions)                      -->
<!-- _______________________________________________________________ -->

<!ELEMENT PN.Subnet.places (PN.Place)* >
<!ELEMENT PN.Subnet.inputArcs (PN.InputArc)* >
```

```
<!ELEMENT PN.Subnet.transitions (PN.Transition)* >
<!ELEMENT PN.Subnet.outputArcs (PN.OutputArc)* >
<!ELEMENT PN.Subnet.subnets (PN.Subnet)* >

<!-- ______________________________________________________________ -->
<!--                                                                -->
<!-- METAMODEL PACKAGE:  PN                                         -->
<!-- ______________________________________________________________ -->


<!-- ______________________________________________________________ -->
<!--                                                                -->
<!-- METAMODEL CLASS:  Place                                        -->
<!-- ______________________________________________________________ -->

<!ELEMENT PN.Place.bound (#PCDATA | XMI.reference)* >
<!ELEMENT PN.Place.name (#PCDATA | XMI.reference)* >
<!ELEMENT PN.Place.tokens (#PCDATA | XMI.reference)* >
<!ELEMENT PN.Place.inArc (PN.InputArc)* >
<!ELEMENT PN.Place.outArc (PN.OutputArc)* >

<!ELEMENT PN.Place (PN.Place.bound?, PN.Place.name?, PN.Place.tokens?,
                    XMI.extension*, PN.Place.inArc*, PN.Place.outArc*)? >
<!ATTLIST PN.Place
            %XMI.element.att;
            %XMI.link.att;
>

<!-- ______________________________________________________________ -->
<!--                                                                -->
<!-- METAMODEL CLASS:  InputArc                                     -->
<!-- ______________________________________________________________ -->

<!ELEMENT PN.InputArc.fromPlace (PN.Place)? >
<!ELEMENT PN.InputArc.toTransition (PN.Transition)? >
<!ELEMENT PN.InputArc (PN.Arc.weight?, XMI.extension*,
                       PN.InputArc.fromPlace?,
                       PN.InputArc.toTransition?)? >
<!ATTLIST PN.InputArc
            %XMI.element.att;
            %XMI.link.att;
>

<!-- ______________________________________________________________ -->
<!--                                                                -->
<!-- METAMODEL CLASS:  Transition                                   -->
<!-- ______________________________________________________________ -->

<!ELEMENT PN.Transition.name (#PCDATA | XMI.reference)* >
<!ELEMENT PN.Transition.guard (#PCDATA | XMI.reference)* >
<!ELEMENT PN.Transition.random_variable (#PCDATA | XMI.reference)* >
<!ELEMENT PN.Transition.memory_policy (#PCDATA | XMI.reference)* >
<!ELEMENT PN.Transition.priority (#PCDATA | XMI.reference)* >
<!ELEMENT PN.Transition.inArc (PN.InputArc)* >
<!ELEMENT PN.Transition.outArc (PN.OutputArc)* >

<!ELEMENT PN.Transition (PN.Transition.name?, PN.Transition.guard?,
                         PN.Transition.random_variable?,
                         PN.Transition.memory_policy?,
```

```
                         PN.Transition.priority?, XMI.extension*,
                         PN.Transition.inArc*, PN.Transition.outArc*)? >
<!ATTLIST PN.Transition
            %XMI.element.att;
            %XMI.link.att;
>

<!-- _______________________________________________________________ -->
<!--                                                                 -->
<!-- METAMODEL CLASS:  OutputArc                                     -->
<!-- _______________________________________________________________ -->

<!ELEMENT PN.OutputArc.fromTransition (PN.Transition)? >
<!ELEMENT PN.OutputArc.toPlace (PN.Place)? >

<!ELEMENT PN.OutputArc (PN.Arc.weight?, XMI.extension*,
                        PN.OutputArc.fromTransition?,
                        PN.OutputArc.toPlace?)? >
<!ATTLIST PN.OutputArc
            %XMI.element.att;
            %XMI.link.att;
>

<!-- _______________________________________________________________ -->
<!--                                                                 -->
<!-- METAMODEL CLASS:  Arc                                           -->
<!-- _______________________________________________________________ -->

<!ELEMENT PN.Arc.weight (#PCDATA | XMI.reference)* >

<!ELEMENT PN.Arc (PN.Arc.weight?, XMI.extension*)? >
<!ATTLIST PN.Arc
            %XMI.element.att;
            %XMI.link.att;
>

<!-- _______________________________________________________________ -->
<!--                                                                 -->
<!-- METAMODEL CLASS:  Subnet                                        -->
<!-- _______________________________________________________________ -->

<!ELEMENT PN.Subnet.name  >

<!ELEMENT PN.Subnet (PN.Subnet.name?, XMI.extension*,
                     PN.Subnet.inputArcs*, PN.Subnet.transitions*,
                     PN.Subnet.subnets*, PN.Subnet.places*,
                     PN.Subnet.outputArcs*)? >
<!ATTLIST PN.Subnet
            %XMI.element.att;
            %XMI.link.att;
>

<!ELEMENT PN ((PN.Place | PN.InputArc | PN.Transition | PN.OutputArc |
              PN.Arc | PN.Subnet)*) >
<!ATTLIST PN
            %XMI.element.att;
            %XMI.link.att;
>
```

```
<!ELEMENT PetriNet ((PN)*) >
<!ATTLIST PetriNet
            %XMI.element.att;
            %XMI.link.att;
>
```

B.2 A Sample Petri Net Model


```
<?xml version = ‘1.0’ encoding = ‘ISO88591’ ?>
<!DOCTYPE XMI SYSTEM ’file:PetriNet.dtd’ >
<XMI xmi.version=’1.0’>
  <XMI.header>
    <XMI.metamodel xmi.name = ‘PetriNet’ xmi.version = ‘0.1’/>
  </XMI.header>
  <xmi.content>
    <PN.Subnet xmi.id=’a1’>
      <PN.Subnet.name>
        CBelt
      </PN.Subnet.name>
      <PN.Subnet.places>
        <PN.Place xmi.id=’a2’>
          <PN.Place.name>
            Ready
          </PN.Place.name>
          <PN.Place.inArc>
            <PN.InputArc xmi.idref=’a4’/>
          </PN.Place.inArc>
        </PN.Place>
      </PN.Subnet.places>
      <PN.Subnet.transitions>
        <PN.Transition xmi.id=’a3’>
          <PN.Transition.inArc>
            <PN.InputArc xmi.idref=’a4’/>
          </PN.Transition.inArc>
          <PN.Transition.outArc>
            <PN.OutputArc xmi.idref=’a5’/>
          </PN.Transition.outArc>
        </PN.Transition>
      </PN.Subnet.transitions>
      <PN.Subnet.inArcs>
        <PN.InputArc xmi.id=’a4’>
          <PN.InputArc.fromPlace>
            <PN.Place xmi.idref=’a2’/>
          </PN.InputArc.fromPlace>
          <PN.InputArc.toTransition>
            <PN.Transition xmi.idref=’a3’/>
          </PN.InputArc.toTransition>
        </PN.InputArc>
      </PN.Subnet.inArcs>
      <PN.Subnet.outArcs>
        <PN.OutArc xmi.id=’a5’>
          <PN.OutputArc.toPlace>
            <PN.Place xmi.idref=’a7’/>
          </PN.OutputArc.toPlace>
          <PN.OutputArc.fromTransition>
            <PN.Transition xmi.idref=’a3’/>
          </PN.OutputArc.fromTransition>
        </PN.OutArc>
      </PN.Subnet.outArcs>
      <PN.Subnet.subnets>
        <PN.Subnet xmi.id=’a6’>
          <PN.Subnet.name>
            Box
          </PN.Subnet.name>
          <PN.Subnet.places>
```

```
            <PN.Place xmi.id = 'a7'>
              <PN.Place.name>
                Ready
              </PN.Place.name>
              <PN.Place.outArc>
                <PN.OutputArc xmi.idref='a5'/>
              </PN.Place.outArc>
            </PN.Place>
          </PN.Subnet.places>
        </PN.Subnet>
      </PN.Subnet.subnets>
    </PN.Subnet>
  </xmi.content>
</XMI>
```