Marcus Alanen | Ivan Porres

Model Interchange Using OMG Standards

Turku Centre *for* Computer Science

TUCS Technical Report

No 675, March 2004

Model Interchange Using OMG Standards

Marcus Alanen
Ivan Porres
TUCS Turku Centre for Computer Science
Åbo Akademi University, Department of Computer Science
Lemminkäisenkatu 14, FIN-20520 Turku, Finland
e-mail:{marcus.alanen,ivan.porres}@abo.fi

TUCS Technical Report
No 675, March 2004

Abstract

We discuss the need for a robust model interchange document format in order to realize the Model Driven Engineering vision. The OMG proposes XMI and XMI-DI as the standard document formats to store and exchange models between applications. However, there are still some open issues regarding the current version of these standards. In this article, we discuss some of these issues and conclude with a plea for a continued discussion on the future and improvement of the XMI and the XMI-DI model interchange standards.

TUCS Laboratory
Software Construction Laboratory

1 Introduction

Model Driven Engineering (MDE) [10] advocates the use of models to represent all the relevant information in a software development project. Software development is then carried out as a sequence of model development and transformation. MDE is the result of the recent development on computer and modeling languages, awareness of the need of software development methodologies and the constant need to tackle larger and more complex development projects.

We believe that MDE opens a window for new development methods and tools that are not available or are too expensive to implement in other approaches such as source-code driven development. However, MDE also presents new challenges that should be addressed before the approach can be used in practice. One of these challenges is how to represent models in a machine-independent format to allow model interchange between tools and systems. This exchange format should be well-documented, stable and supported by different tools from different vendors.

The Object Management Group (OMG), the same standard organization that maintains the UML standards, proposes the use of XMI [13, 16] and XMI-DI [17] to enable model interchange. One of the strong points of XMI is that it is based on XML [22]. XML has been successfully used to support many document and model representation standards. XML is well-documented, machine-independent and there exist plenty of tools supporting it. Thus, XMI documents should be portable and easy to parse.

It is also import to remark what XMI is not. XMI cannot be used to define the structure of a modeling language. This is the role of the Meta Object Facility (MOF) [14] standard. Actually, XMI can be used to serialize any MOF-based modeling language including, but not restricted to UML and MOF itself. The fact that XMI is independent of a modeling languages is at the same time one of its strong points and major weaknesses. First, two tools that use two different versions of the UML standard, for example UML 1.3 and UML 1.4 will not be able to exchange models even if they use XMI. Also, a UML model serialized as a XMI document will not contain any extra information that is not present in the UML model as such. Surprisingly, the UML and MOF metamodels do not contain information about the diagrammatic representation of models. A UML model may state that there is a class named "Person" in a model, but it cannot state that this class is represented in a diagram by a rectangle in a certain position, size and color. To remedy this situation the XMI-DI [17] standard has been proposed. XMI-DI is not a model interchange format but a metamodel to describe diagram information.

We should also remember XMI is neither an application programming interface to retrieve information from models, such as the Java Metadata Interface (JMI) [8], or the Eclipse EMF [4], nor a communications protocol to transport models between systems such as HTTP [5] or WebDAV [6]. This implies that there are no standard software components to create or retrieve XMI documents from a filesystem or multiuser repository since the API and communication mechanism for such components has not been standardized.

1.1 Is XMI Ready for MDE?

The question that we raise in this paper is how suitable the XMI and XMI-DI standards are for model interchange in practice. We will review different scenarios for model interchange, how suitable the current standards are to implement these scenarios and how different tools implement the standards. We will also propose different improvements for the standards.

The results presented in this article have been obtained using three different approaches. First, we have carefully studied the OMG documents that describe the XMI,

XMI-DI and MOF standards. These documents are in constant evolution and a new major revision is expected to be ready for the release of UML 2.0. When possible, we have used the latest versions of these documents, including drafts.

We have also performed practical experiments to test model interchange between different tools, including commercial UML tools such as Rational Rose and open source model repositories such as Eclipse EMF. Although the OMG standards are supposed to be the authoritative definition of the model interchange formats, we have observed some discrepancies on how different tools implement model interchange features.

Finally, we have implemented different research tools (SMW [18], Coral [2]) that include XMI support. We have also developed a prototype multiuser model repository [1] based on XMI. This work has allowed us to obtain first hand experience on all the issues that appear when implementing XMI in real applications.

We proceed as follows: in Section 2 we discuss the role of XMI and XMI-DI with relation to modeling tools while we present various usage scenarios for XMI and XMI-DI. Currently, many of these scenarios are not viable in practice. We explore some of the problems associated with XMI in Section 3 and XMI-DI in Section 4. Finally, we present some basic practical tests that were performed with common modeling tools and these results along with answers to the problems mentioned and ideas for future improvement are given in Section 5.

2 Using XMI

To understand how models can be interchange using the XMI standard we need to understand how models are organized according to a modeling language. According to the OMG modeling standards, a modeling language such as UML is defined as a model in the MOF language. Since a modeling language is defined as a model, that model is called a metamodel. The UML metamodel defines concepts such as Package and Class that can be used in UML models.

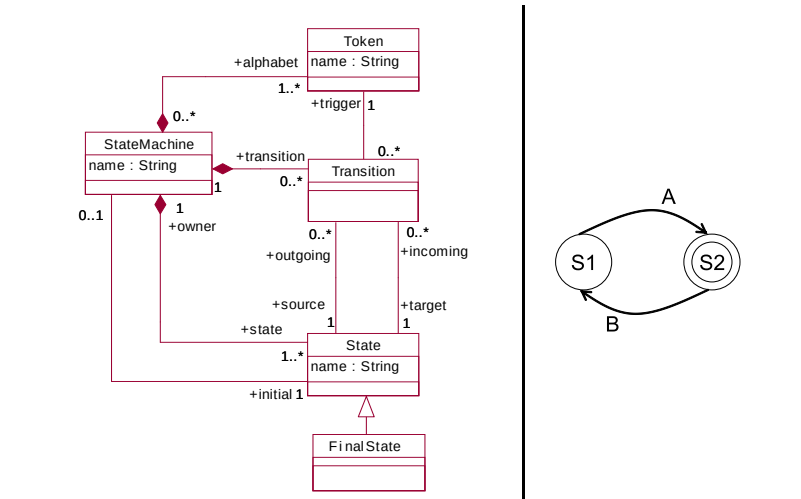
MOF is thus often called a metamodeling language. It defines important concepts to model relationships (or, rather, properties) between metamodel elements. For example, the fact that a Package can own several Classes is defined using the MOF concept of composition. Other MOF concepts are e.g. names properties and their multiplicity ranges and metamodel element (metaclass) names.

The advantage of this approach is that it is possible to create new modeling languages just by combining these basic concepts. Also, tools or standards based on MOF can be used in any MOF-based modeling language. We can demonstrate this approach by creating a MOF model for a simple modeling language to describe finite state machines. The metamodel for this language is shown on the left side of Figure 1. Each class in that model represents a concept in the FSM language while each association represents a relationship between concepts.

The bottom left side of the Figure 1 represents a particular model in the FSM language. Since the XMI standard defines a series of rules to serialize any MOF-based modeling language, we can also represent the example abstract model in XMI using these rules. The resulting XML document is shown in the right part of Fig. 1.

Based on this discussion, we can now define an XMI document as an XML document that describes one or more models that have been serialized according to a given metamodel and the rules proposed in [13]. An XMI document does not necessarily represent a UML model since it can be based on other metamodels and it is not necessarily a file since it can be stored and retrieved from e.g. a network stream or database.

We can see in the XMI document in Fig. 1 that a model element can refer to other elements using its **xmi:id**. This is necessary since the underlying structure of a model




```
<?xml version='1.0' encoding='UTF-8'?>
<xmi:XMI xmlns:xmi="http://schema.omg.org/spec/XMI/2.0"
         xmlns:xlink="http://www.w3.org/1999/Xlink"
         version='2.0' timestamp='Sun, 06 Mar 2005 18:21:02 +0200'>
    <documentation>. . .</documentation>
    <FSM:StateMachine xmlns:FSM="http://www.abo.fi/FSM/1.0"
                      xmi:id="e1" name="Example">
        <alphabet xmi:id="e2" name="A">
            <transition xmi:idref="e3" />
        </alphabet>
        <alphabet xmi:id="e4" name="B">
            <transition xmi:idref="e5" />
        </alphabet>
        <state xmi:id="e6" name="S1">
            <incoming xmi:idref="e5" />
            <outgoing xmi:idref="e3" />
        </state>
        <state xmi:type="AcceptingState" xmi:id="e7" name="S2">
            <incoming xmi:idref="e3" />
            <outgoing xmi:idref="e5" />
        </state>
        <transition xmi:id="e3" source="e6" target="e7" trigger="e2" />
        <transition xmi:id="e5" source="e7" target="e6" trigger="e4" />
    </FSM:StateMachine>
</xmi:XMI>
```

Figure 1: Top left: FSM metamodel; Top right: Example FSM Model ; Bottom: Example FSM Model as an XMI document

is a graph while a XML document is a tree. It is also possible to link across XMI documents, i.e., a model element may refer to elements that reside in a different document. This feature enables us to use XMI in larger projects.

In the context of a modeling tool, an XMI filter is a tool component that can create and retrieve XMI documents based on one or more given metamodels. Since a metamodel is based on a model, it can also be represented as an XMI document. We should note that although it is possible to create a DTD or XML Schema [25, 26] to define the structure of an XMI document this is not strictly necessary, since an XMI filter can obtain all the necessary information about the structure of an XMI document from a metamodel. Actually, a metamodel contains additional information that cannot be expressed in a DTD. An example of this are relations which are unordered such as a UML Package owning a set of UML Classes; in XML, the order is always important.

2.1 Scenarios for Model Interchange

In the rest of this section we identify some common, generic usage scenarios for XMI and XMI-DI. We have experience with all of the scenarios, but interoperability problems with XMI and especially the mostly unsupported XMI-DI mean that these scenarios should be seen as future goals for the XMI standards.

2.1.1 Migration From One CASE Tool To Another

Our first scenario describes the need to move all our models from one CASE tool to another, probably from a different vendor. To avoid vendor lock-in, we use XMI to provide seamless, robust and consistent ways to migrate from one CASE tool to another. We plan to migrate the models once and not use the current CASE tool any more.

This is the simplest scenario for XMI. In this case, the size of the XMI files or the speed of the XMI export and import process is not too important. Also we may tolerate small defects in the migration, if they can easily be found and corrected in the new CASE tool.

2.1.2 Model Interchange within a Desktop

Another scenario for model interchange is to use inter-application communication mechanisms such as cut-and-paste and drag-and-drop to share models or fragments of a model from one application to another. This feature is present in many commercial UML tools. For example, it is possible to select a part of a UML class diagram, copy it to the clipboard and paste it into a PowerPoint presentation. We should note that in this case, the modeling tool is not placing a model to the clipboard but a picture of a model in a graphical format such as a Windows Metafile. However, nothing prevents the use of a standard model interchange format such as XMI between applications running within the same desktop.

In this scenario we will interchange just a small part of a model. Therefore, or XMI documents will probably contain links to external documents.

2.1.3 A Model Driven Engineering Tool Set

In this scenario, different tools are used to create and maintain the models. Each tool may be specialized in a certain task, such as requirements engineering, analysis modeling, or model implementation. The tools are not necessarily used in sequence.

Since we use many different tools, the XMI documents representing our project may include elements that are not supported by every tool. However, all the tools

should respect and leave intact those unsupported elements, i.e., they should be able to load and save the elements even though they do not understand anything about them.

2.1.4 A Multiuser Model Repository

In this scenario, we plan to store all the models in a central repository, easily accessible by different developers using heterogeneous tools. XMI is used as the exchange format between the repository and the developers' tools; we ignore in this paper the transport protocol used. This is the most demanding scenario for XMI. The XMI documents may include version information or represent differences between models. Also, we might only interchange parts of a model.

Our anecdotal experience as users of different case tools suggest that even the simplest scenario, migration from one tool to another, can be problematic. In most cases, it is better to use Rational Rose's proprietary file format (Rose Petal) than the standard XMI file format. In the next section we discuss some of the reasons behind this situation.

3 XMI Drawbacks and Solutions

In this section we study some of the inherent problems that exist in the XMI standard. Many of these problems are a consequence of the rather fast development of the standard, with multiple versions appearing in a relatively short time. Also, the initial versions of XMI (and UML) were probably created taking into account only the first scenario.

3.1 Too Many Versions, Options and Optimizations

One of the main impediments to use XMI in practice is the large number of different versions of the XMI standard and UML metamodels. At the moment of writing this text, there are four versions of XMI (1.0, 1.1, 1.2 and 2.0) and seven versions of UML (1.0 to 1.5 and 2.0). This means that a document containing a UML model can actually be serialized in 28 different combinations of XMI and UML versions. The diversity of different XMI implementations has already been studied in [9]. There, Jiang and Systï£¡ devised a method to explore differences between XMI formats using DTDs.

The fast evolution of XMI and UML gives little room for implementors to try it out in practice. We consider that the solution to this problem is to include in each new version of XMI or UML, a mechanism to transform old documents into the new version. In the case of XMI, this could be achieved at the XML level by including an XSLT [21] transformation document with the new version of the standard. In the case of a new version of UML, the transformation could be defined using the MOF-based QVT language [12].

Some of the new features introduced in the latest version of XMI are questionable. A new concept related to XMI appears in the appendix of the UML 2.0 Infrastructure [15], where XMI files can be made smaller in size by removing some redundant information. However, the problem is that they seem of little use. These extensions are specific to UML and thus not applicable to other modeling languages, defined by the user or by the OMG. We have performed anecdotal experiments that suggest that these optimizations provide a minimal benefit, whereas standard compression tools, such as gzip, can obtain compression ratios of up to 95% on XMI files. A "binary XML" format has even been discussed [20] by the W3C, which would obviate the need for any application-specific (i.e. XMI) compression solutions. Also, these UML optimizations may actually lose information. The appendix states that "...there is an object in the model which contains redundant information and can be reconstructed from the values

of derived attributes in other objects." This is true only to a certain extent, since some information such as the UUIDs (which will be discussed later) of the objects are not retained. The rationale of the gain these optimizations bring is thus questionable, as they add more variability to the XMI format.

3.2 Metamodel Identification

To be able to process a model, a tool must know its metamodel. However, XMI lacks of a reliable mechanism to identify which metamodels are used in a document. The problem is complicated since XMI 1.x and XMI 2.0 use to completely different approaches.

The **XMI.metamodel** element can be used in XMI 1.x to identify the metamodel used in a document (Pages 3-12 and 3-20 of [13]). An example of this approach is as follows:

```
<XMI.header>
  <XMI.metamodel name="UML" version="1.3" href="UML.xml"/>
```

In XMI 2.0, the **XMI.metamodel** element has been removed. Instead, each metamodel is assigned one or more XML namespaces. A namespace specification may be any arbitrary string, that according to the standard "provides a permanent global name for the resource. An example is **http://schema.omg.org/spec/UML/1.4**. There is no requirement or expectation by the XML Namespace specification that the logical URI be resolved or dereferenced during processing of XML documents". In page 1-16 of [13] we can see the following example:

```
 <xmi:XMI version="2.0"
   xmlns:UML="http://schema.omg.org/spec/UML/1.4"
   xmlns:xmi="http://schema.omg.org/spec/XMI/2.0">
```

The new mechanism is more correct from an XML point of view, and certainly enables one to mix models of different metamodels with ease. However, it is inconvenient as there is no published mapping between namespaces and existing versions of UML and other modeling languages. Even if that mapping exists, it is merely an opaque string to a tool. No information can be deduced from it, and thus a generic modeling tool cannot work with unknown metamodels; it must know, in advance, about the namespace mapping and the metamodel. Our solution in the Coral modeling tool is a user-editable table with mappings between namespaces URIs and actual metamodels. However, this solution requires manual work by the user and is not viable in the long term.

Another problem to create robust XMI import component is that old XMI documents may combine namespaces with header information. One commercial tool produces the following header when exporting a model to XMI:

```
 <XMI xmi.version='1.1' xmlns:UML='//org.omg/UML/1.3'>
   <XMI.header>
     <XMI.metamodel xmi.name='UML' xmi.version='1.4'/>
```

The namespace declaration may suggest that the document contains a UML 1.3 model, while the header states that it is a UML 1.4 model.

The problem of metamodel identification is an important issue and its final consequence is that it is not possible to load new metamodels based on their description from the namespace URI. Therefore the current definition of namespace declarations in XMI cannot be used as such in a metamodel-based modeling tool that should support any user-defined modeling language.

3.3 Element Identification

Element identification is the process of obtaining a unique reference to a model element that distinguishes it from the rest of model elements that compose all the artifacts

required in a project.

A unique identifier for each model element allows us to differentiate between two instances of the same element and two elements that are similar. This distinction is fundamental to implement model management operations like comparing two models, merging two models into one or duplicating (parts of) a model. Unique identifiers can also be used to create traceability links between the artifacts in two different models, that may be expressed in two different languages.

The need for unique element identification was already tackled in XMI 1.x. This standard provides two attributes to uniquely identify an element: the **xmi.id** and **xmi.uuid** attributes. The **xmi.id** attribute is used to reference elements inside an XMI document. They are unique arbitrary strings only in the context of a given document and they may change in upcoming revisions of a document. For some inexplicable reason some tools still use **xmi.id** for referencing elements in different files, even though it is very clearly a brittle idea. On the other hand a Universally Unique Identifier (UUID) [3] is assumed to be a globally unique long string and persistent across serializations. UUIDs may be assigned the first time that an element is exported to an XMI document. Later, any standard-compliant open tool that imports the XMI document should not change or remove the assigned UUIDs.

UUID strings fulfill all the requirements to uniquely identify elements for the usage scenarios described previously: model management and transformation traceability. Unfortunately, many of the existing UML tools do not generate or preserve UUID strings and cannot be used rigorously to create links between different documents. This is further examined in a small survey in Section 5.

Of special mention is also the **xmi.label** attribute. It can be used to give elements an arbitrary name by the designer. But nowhere is it explained why it should be used instead of an **uuid** or **id** attribute—so why use it?

A recent addition to the XML family of standards is the xml:id Version 1.0 Candidate Recommendation [28] by the W3C which is very similar to **xmi.id**. Indeed when XML provides similar or superior alternatives for element identification and element linking, we believe they should be used in lieu of the XMI standards. This allows the usage of general-purpose XML tools for navigation in models, but would also mean that XMI must be constantly updated to match the new standards by W3C. This also means that the XMI 2.x standards should better incorporate the usage of XLink [23], XPath [27] and XPointer [24] instead of inventing their own syntax.

4 XMI-DI Drawbacks and Solutions

As we have discussed in the introduction, the UML and MOF metamodels do not contain a description of the diagrammatic information that can appear in a model. This includes the position, size, color, and fonts of the visual elements shown in diagrams. This omission has been tackled by a new proposal for diagram interchange named XMI-DI.

XMI-DI is not an extension to XMI or a different way to serialize models into streams. It is just another modeling language, similar to the FSM language shown in Section 2, but explicitly designed for enabling us to display it on a two-dimensional canvas such as a monitor or paper. The three main concepts in XMI-DI are the **Diagram**, the **GraphNode** and **GraphEdge** that can be combined to represent diagrams such as those from UML. XMI-DI is not limited to UML, and so it can be used to represent diagrams for models of other languages. Figure 2 shows how a diagram can be constructed using basic XMI-DI primitives.

Although we have several improvement suggestions for the XMI-DI specification language itself, they are in most cases quite insignificant in comparison with the ma-

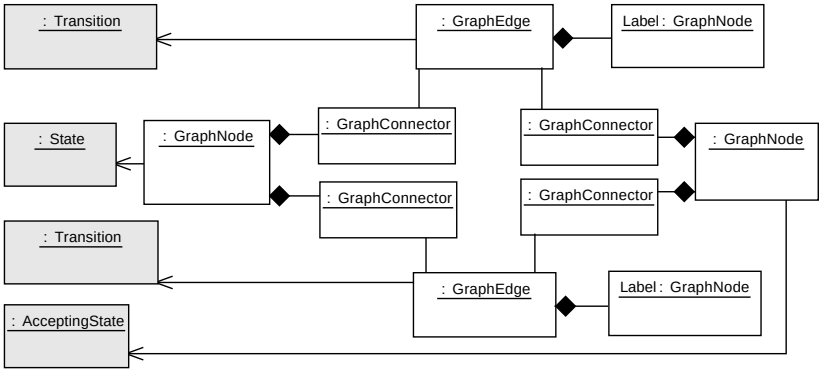


Figure 2: Example FSM diagram as an XMI-DI object model, representing the abstract model from Figure 1. An object name represents a simple GraphElement; otherwise, a directed arrow represents the relation from the GraphElement to the abstract model element (in gray).

jor areas requiring improvement. The are two main issues related with XMI-DI as a diagram interchange formats for models. The main problem is that given an XMI-DI diagram, the language is not rich enough to describe the graphical presentation of models: The information about how to render concrete XMI-DI nodes and edges is not present in the language. An XMI-DI document can state that there is a diagram with a UML Use Case in it, but it cannot state that it is rendered as an ellipse and not as a rectangle.

Another major omission is that the XMI-DI does not contain information about how to create a diagram from an abstract model. For example, given a UML class model, XMI-DI cannot describe how we can create the corresponding GraphNodes and GraphEdges that represent all classes, associations, etc. Also, XMI-DI cannot express layout constraints. Some UML diagrams, such as sequence diagrams, have a layout constrained by the semantic model.

The consequence is that XMI-DI in its current form should be extended or complemented with other languages in order to be usable in metamodel-based tools that can process models in many modeling languages and diagrammatic notations. We consider that there is an immediate need for a mapping language from abstract models to XMI-DI constructs, a rendering language and a layout constraints language. We do not mean to criticize the XMI-DI effort in itself; it is a good start and we have shown that it is possible to use it in the form it is defined today, but for even easier adoption it should be extended further.

5 Present and Future of Model Interchange Using OMG Standards

It is not an easy task to evaluate the current state of the XMI and XMI-DI standards since this evaluation is affected by two factors: the de jure standards as defined by the OMG and the de facto implementation of these standards in modeling tools.

We have implemented our own experimental modeling tool called Coral that supports the XMI and XMI-DI standards. The result is a tool that can interchange models with several well-behaved commercial tools. More significantly, it can exchange models and diagrams with Gentleware's Poseidon using XMI-DI. According to sources from Gentleware, Coral is the first independent tool that supports XMI-DI besides Po-

seidon. An example of this interchange can be seen in Figure 3. When tools cannot interchange model data, we have provided them with a picture of the selected diagram parts (e.g. for the kpaint drawing program) or just the XMI stream as plain text (e.g. with the kate editor), as can be seen in Figure 4. It is a pity that there still is no standard MIME type [11] for XMI streams, and so tools must exchange plain-text or generic XML fragments.

Our implementation of XMI and XMI-DI in Coral has revealed a great deal of information about the strengths and weaknesses of these standards. The two main drawbacks of XMI-DI became appallingly obvious: there is no definition of any rules on how to map UML models into XMI-DI and there is no definition of how to render different GraphNodes and GraphEdges. This information has to be hardcoded into the tool.

Besides developing our own XMI tool component, we have also performed a simple study of the XMI documents generated by six commercial CASE tools. There are many tools in the market that support UML in one way or another. The tools studied were chosen because they are popular, they can import and export XMI documents and the vendor offers a trial or free version available on the Internet. We have created a simple model with each of the tools and exported it to an XMI file. Then we have examined the generated file in a text editor to observe the header of the document to determine the XMI and UML version of the document. The objective of this study is to determine the effort of creating a new component that can import existing XMI files. If we would like to develop a new tool that can load and transform models generated by the studied tools, the XMI import filter of that tool should support all the XMI and UML versions used by the CASE tools.

A second experiment was performed to determine if a CASE tool preserves UUID strings. First, we have created a simple model and saved it as XMI. Then we have edited the XMI file with a text editor and added a UUID identifier to the Model element, the main element in a UML model. For example, in a XMI 1.x document we will add the following string shown in bold face:

```
<UML:Model xmi.id='1' name='Example Model'
  xmi.uuid = '123' isRoot='false' isLeaf='false'
  isAbstract='false' isSpecification='false'>
```

Then we have loaded the modified XMI document in the CASE tool. The modified XMI document should be imported without problems. Afterward, we exported the model again to an XMI document and opened with a text editor. The Model element should contain the same UUID as introduced by us. If the CASE tool modified the UUID string or removed it completely then it does not preserve UUID strings.

The results from a small sample of tools are not too encouraging. UUIDs are discarded by 5 tools out of 6. Also, each tool seems to use a different combination of XMI and UML version. Of special mention is Together 6.0, that can generate 7 different kinds of XMI documents from the same model, using different XMI versions (1 or 1.1), UML metamodels (1.1, 1.3 or 1.4) or extensions introduced by different XMI components (Unisys, IBM, or plain OMG).

When analyzing the various versions of the XMI, one easily notices how XMI 2.0 has taken a very different road from previous versions by aiming for better XML and XML namespace [19] compliance. Indeed, the change is so dramatic that there seems to be little reason to support XMI 1.x in new tools. Unfortunately, there are many older tools in the market than only support XMI 1.x so in many cases it is necessary to support backwards compatibility reasons.

In light of the arguments given in this paper, we primarily suggest a compliance test suite for checking XMI compatibility. It should consist of a set of files with successively more advanced XMI features. The current compliance points in appendix A of [13] are quite coarse-grained, and it is not always possible for vendors to know

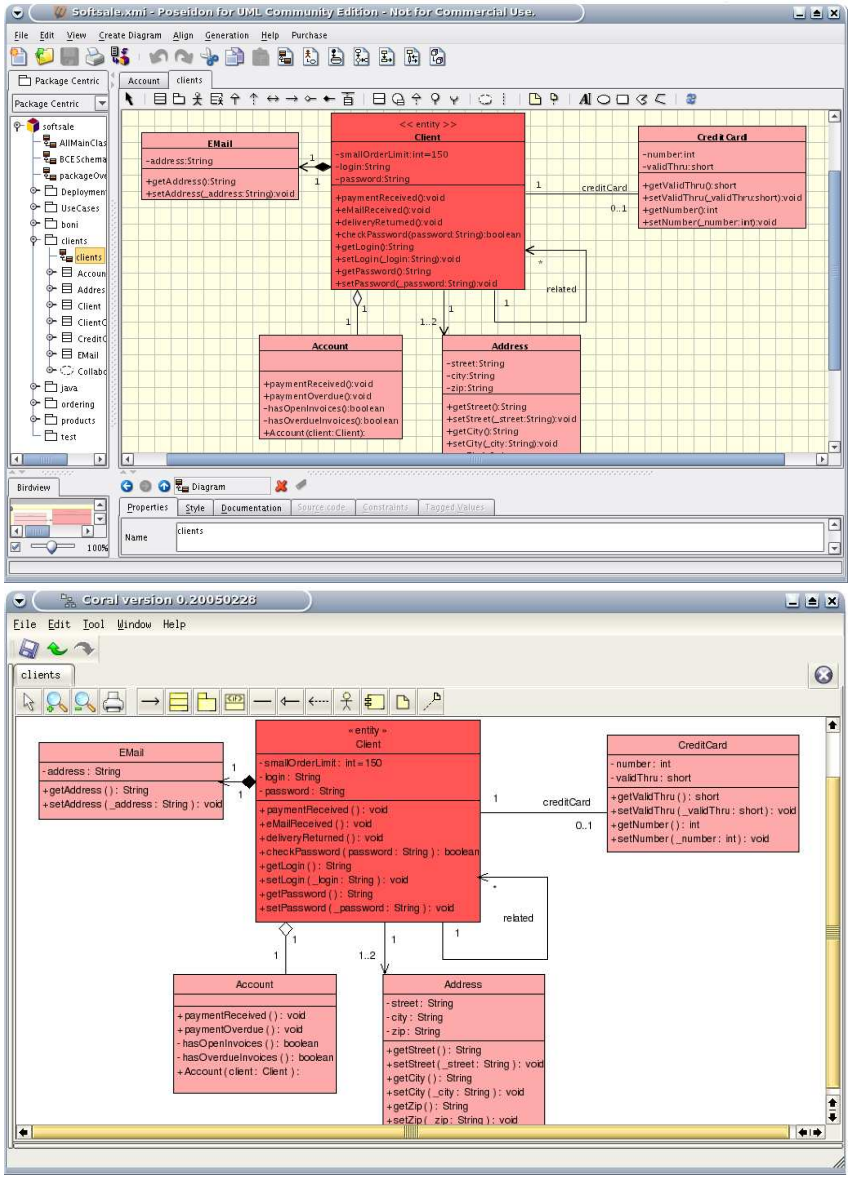


Figure 3: A diagram shown in Poseidon 3.0 (left) and Coral (right). The model was exchanged using the XMI 1.2, UML 1.4 and XMI-DI 2.0 standards.

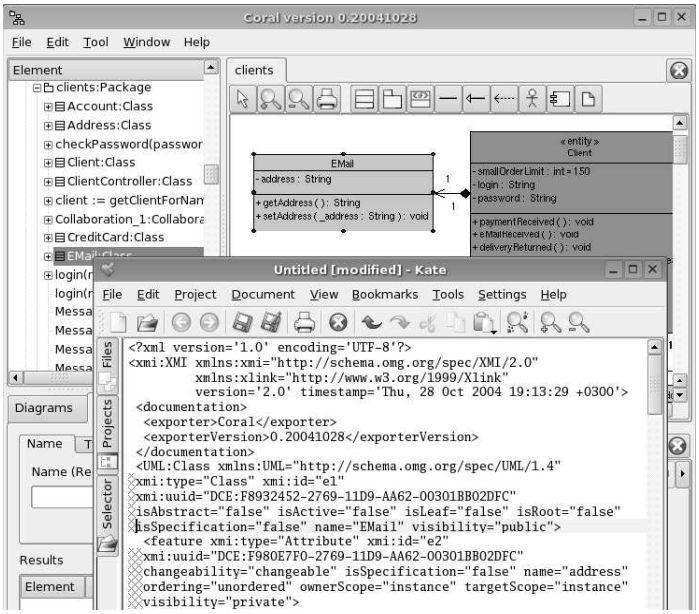


Figure 4: Cut-and-paste using XMI.

Tool	Exporter	XMI Version	UML Version	Supports UUID	XMI-DI
Coral	Coral	1.2,2.0	1.x	yes	yes
EMF UML2	(no header)	2.0	2.0	no	no
Magic Draw 7.1	Unisys.JCR.2	1	1.3	yes	no
Poseidon 3.0	Netbeans	1.2	1.4	no	yes
Rational Rose	Unisys.JCR.1	1.1	1.3	no	no
Together 6.0	TogetherSoft	1, 1.1	1.1,1.3,1.4	no	no
Visual Paradigm 3	(no header)	1.1	1.4	no	no
Visual UML 3.24	Visual UML	1	1.3	no	no

whether or not their tool provides the support they claim. Detecting successful loading could be done by evaluating OCL constraints on the loaded models as suggested by Stefan Haustein [7], although this requires an OCL or similar interpreter for automation. The constraints would verify that all the model elements, their interconnections, and the information stored in them are correct. This alone would mean that customers can check the official XMI compliance test scores and tremendously benefit from the interchange format. "Full compliance" as so often touted by vendors would in fact be split into several compliance levels of XMI features, such as interfile linking, support for **XMI.extensions** and multiple models using multiple metamodels in one file. We expect that a compliance suit will improve the implementation of XMI standard in modeling tools. However, we consider that the XMI standard should also be improved in these directions:

- A standard mechanism for element identification. This includes a deprecation of **xmi.label** and enforcement of the **xmi.uuid** identification mechanisms. This means that if a tool imports an element that contains a UUID, the same UUID should appear in any future serialization of that element. Similarly a tool should not require **xmi.id** preservation.
- A standard mechanism to retrieve what metamodel is used in a certain model, including future metamodels. This will allow the creation of generic metamodel-based modeling tools.
- Each new version of XMI should include the definition of a XSLT transformation to convert documents created using the previous version. Once these transformations has been defined for all versions of XMI, the XMI 1.x standards should be deprecated due to their lack of XML compliance.

From the point of view of XMI-DI we consider that the standard should be extended to include:

- A language to define mappings from the abstract syntax of a modeling language to its concrete syntax. This language should be used to create a mapping from UML 1.x or 2.0 to the XMI-DI language.
- A language to define how to render specific nodes and edges. This could be made as an extension to the mapping language.
- An official compliance test suite similar to the one proposed at the XMI level.

We think that the burden of creating compliance test suites and evolution mappings can be placed on the tool vendors that propose new revisions of the standards. The OMG should require that each new proposal for the XMI standards should include the corresponding test and mappings. The result would be that end users have a guaranteed and standard transition path for their models.

6 Conclusions

We consider that in order to apply MDE in practice, model interchange between tools should be as frequent and simple as, for example, source code exchange between text editors, compilers and build tools. In this article we have discussed the benefits of drawbacks of the existing standards for model interchange and we have also shown that it is possible to interchange models and diagrams between tools using only the XMI and XMI-DI standards. We find it important to stress the tasks of the standards discussed in this paper: XMI provides a general-purpose content format for arbitrary

model data, whereas XMI-DI cleanly separates the abstract models from their concrete syntax.

However, we have also discussed important drawbacks in these standards. We do not present these drawbacks as a criticism to the work performed by the OMG and its contributors but as an opportunity to improve the state of the practice.

The current abysmal state of interchange in practice using XMI and XMI-DI cannot be explained alone by complications in the standards themselves. We know from our own experience that the standards are usable as such and that it is possible to implement a working XMI filter with a relatively small effort. Still, basic interoperability is lacking between tools from different vendors.

References

[1] Marcus Alanen. A Meta Object Facility-Based Model Repository With Version Capabilities, Optimistic Locking and Conflict Resolution. Master's thesis, Åbo Akademi University, November 2002.

[2] Marcus Alanen and Ivan Porres. Coral: A Metamodel Kernel for Transformation Engines. In D. H. Akerhurst, editor, *Proceedings of the Second European Workshop on Model Driven Architecture (MDA)*, number 17, pages 165–170, Canterbury, Kent CT2 7NF, UK, Sep 2004. University of Kent.

[3] CAE Specification. DCE 1.1: Remote Procedure Call, 1997. Available at `http://www.opengroup.org/onlinepubs/9629399/toc.htm`.

[4] EMF Development team. Eclipse Modeling Framework. www.eclipse.org/emf.

[5] R. Fielding, J. Gettys, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol — HTTP/1.1, RFC 2616, June 1999. Available at `http://www.ietf.org/rfc/rfc2616.txt`.

[6] Y. Goland, E. Whitehead, A. Faizi, S. Carter, and D. Jensen. HTTP Extensions for Distributed Authoring — WEBDAV, RFC 2518, February 1999. Available at `http://www.ietf.org/rfc/rfc2518.txt`.

[7] Stefan Haustein, February 2004. Discussion on the mailing-list pumllist@cs.york.ac.uk.

[8] S. Iyengar et al. Java Metadata Interface (JMI) Specification API 1.0. Available at `http://java.sun.com/`, June 2002.

[9] J. Jiang and T. Systä. Exploring Differences in Exchange Formats – Tool Support and Case Studies. In *Seventh European Conference on Software Manteinance and Reengineering*. IEEE Computer Society, March 2003.

[10] Stuart Kent. Model Driven Engineering. In *Proc. of IFM International Formal Methods 2002*, volume 2335 of *LNCS*. Springer-Verlag, 2002.

[11] M. Murata, S. St. Laurent, and D. Kohn. XML Media Types — RFC 3023, January 2001. Available at `http://www.ietf.org/rfc/rfc3023.txt`.

[12] OMG. MOF 2.0 Query / Views / Transformations RFP. OMG Document ad/02-04-10. Available at www.omg.org, 2002.

[13] OMG. XML Metadata Interchange, version 1.2, January 2002. Available at `http://www.omg.org/`.

[14] OMG. MOF 2.0 Core Final Adopted Specification, October 2003. Document ptc/03-10-04, available at http://www.omg.org/.

[15] OMG. OMG Unified Modeling Language Infrastructure Specification, version 2.0, September 2003. Document ptc/03-09-15, available at http://www.omg.org/.

[16] OMG. OMG XML Metadata Interchange (XMI) Specification. OMG Document 03-05-02. Available at www.omg.org, 2003.

[17] OMG. Unified Modeling Language: Diagram Interchange version 2.0, July 2003. OMG document ptc/03-07-03. Available at http://www.omg.org.

[18] Ivan Porres. A Toolkit for Model Manipulation. *Software and Systems Modeling*, 2(4), December 2003.

[19] W3C. Namespaces in XML, January 1999. Available at http://www.w3.org/.

[20] W3C. WAP Binary XML Content Format, June 1999. Available at http://www.w3.org/.

[21] W3C. XSL Transformations (XSLT) Version 2.0, November 1999. Available at http://www.w3.org/TR/xslt20/.

[22] W3C. Extensible Markup Language (XML) 1.0 (Second Edition), October 2000. Available at http://www.w3.org/.

[23] W3C. XML Linking Language (XLink) Version 1.0, June 2001. Available at http://www.w3.org/TR/xlink/.

[24] W3C. XPointer Framework, March 2003. Available at http://www.w3.org/TR/xptr-framework/.

[25] W3C. XML Schema Part 1: Structures Second Edition, October 2004. Available at http://www.w3.org/.

[26] W3C. XML Schema Part 2: Datatypes Second Edition, October 2004. Available at http://www.w3.org/.

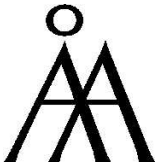
[27] W3C. XML Path Language (XPath) Version 1.0, February 2005. Available at http://www.w3.org/TR/xslt20/.

[28] W3C. xml:id Version 1.0 W3C Candidate Recommendation 8 February 2005, February 2005. Available at http://www.w3.org/.

Turku
Centre *for*
Computer
Science

Lemminkäisenkatu 14 A, 20520 Turku, Finland | www.tucs.fi

University of Turku
- Department of Information Technology
- Department of Mathematics



Åbo Akademi University
- Department of Computer Science
- Institute for Advanced Management Systems Research

Turku School of Economics and Business Administration
- Institute of Information Systems Sciences

ISBN 952-12-1520-8
ISSN 1239-1891