RELATIONSHIP BETWEEN CIM AND XMI

Short Description

This document analyses the representation of CIM in XMI, an interchange format which can amongst others exchange information between different UML modeling tools in order to provide a basis for automatic deployment of changes in the UML representation of CIM models to the relevant management agents and providers. As XML is becoming popular as standard for encoding CIM notation, the so called CIM-XML, it is analyzed as the target of the intended automatic adaptation.

Keywords

Common Information Model (CIM), XML, UML, XMI, SLD, SAP, MagicDraw, Encoding Specification, Management Standards, CIM-XML, Rational Rose

Goals

www.cm-tm.uka.de
C&M (Prof. Abeck)

(1) Understand the SAP landscape management scenario

(2) Have a grasp of the different standards used (CIM, CIM-XML, XMI,UML)

(3) Acknowledge the main findings and problems which are described in the last chapter

Information 1: RELATIONSHIP BETWEEN CIM AND XMI – Goals

(1) SAP has different hardware and software products, which are managed by an SLD Server. This server is part of the SAP landscape management approach and based on CIM. This document analyzes how the CIM information in the SLD can be changed by working with UML class diagrams.

(2) The models relevant for understanding this scenario are briefly reviewed. From the user point of view UML as primary modeling standard and XMI as corresponding file format are described. For the actual transfer of the information in the SLD Server, CIM as the basis of the SLD, especially CIM-XML as the exchange format, are analyzed.

(3) In chapter 3 a manual transfer of information from the diagram to the CIM-XML notation is performed. In the same chapter, differences between the specific UML tool and the general XMI standard are highlighted. Based on these results and general problems with CIM represented in UML the last chapter points out what barriers the desired converter has to overcome. The final converter should be able to take information from many UML modeling tools and deploy this information through the SLD Server.

Table of Contents

Information 2: RELATIONSHIP BETWEEN CIM AND XMI - Content Overview

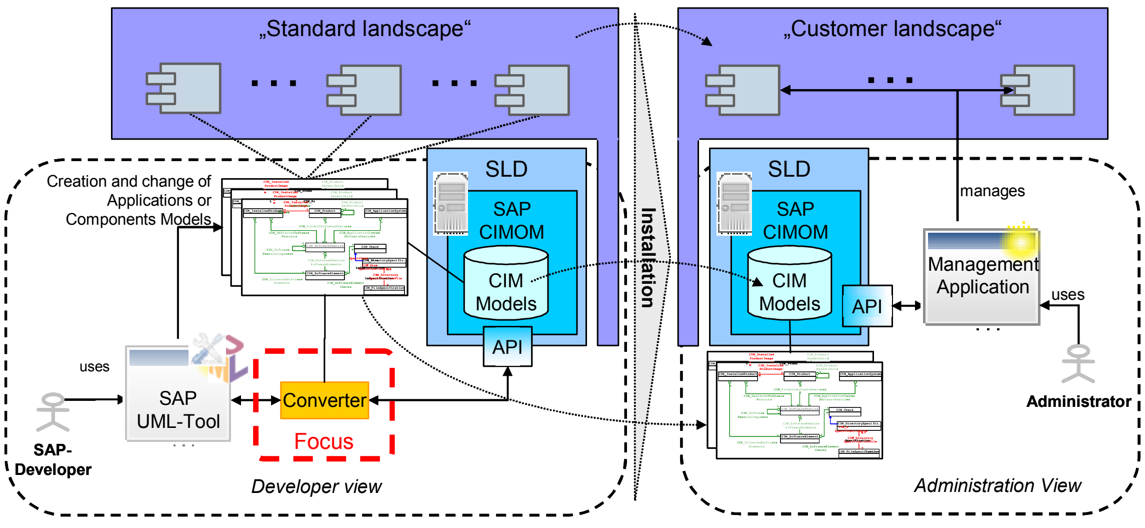
(1) In the first chapter the SAP landscape management scenario is explained as it is the motivation for this document.

(2) The most relevant standards and terms are reviewed shortly, as a full explanation of each term would go beyond the scope of this document.

(3) An UML diagram is shown, as it could occur in the target framework. Relevant parts of its encoding in XMI are shown and compared to the desired notation of the same aspects in CIM-XML. At the end the specific Magic Draw XMI representation is analyzed to derive how standardized it is.

(4) As a first step differences between the actual MagicDraw XMI representation and the XMI standard are shown. After this some general conclusions about the desired automatic transformation from UML models to the SAP landscape management servers are given.

1 SCENARIO



Information 3: SCENARIO – Overview

(Standard Landscape, Customer Landscape) SAP is managing its soft- and hardware in so called landscapes, the original environment from SAP is called standard landscape. Once this general landscape becomes installed in the environment of a customer, this landscape is referred to as customer landscape. SAP is currently aiming to renew and facilitate the management of these landscapes, without exchanging the existing components like the SLD Server, which is described in the next point.

(SLD) System Landscape Directory, manages the hard- and software components of the SAP environment [SAP-SLD]. Technically, the SLD Server is based on the WBEM management standard, and has a management interface. The desired converter should deliver the information created in UML to the management interface of the SLD Server [SAP-SLD]. In the context of this overview, it suffices to know that the inner components of the SLD Server provide a possibility to store information.

(SAP UML tool) Unfortunately, there are many different UML tools; each of them is having a slightly different encoding of their diagrams based on XMI. For the time being, MagicDraw is considered as an example before differences to the standard XMI representation are analyzed. This will be done in chapter 3.

(Converter) This document attempts to generate a basis for the development of such a converter. This converter should be able to retrieve information from different graphical modeling tools and transform it in standardized XMI. This is, or at least will become the standard for information exchange in the SAP environment. From this standard, the information can then be integrated in the SLD Server through another, already existing converter (J-Converter) [Jo05]. For the rest of this document, the difference between the two converters (XMI converter, J-Converter) is irrelevant, and both of them are equally addressed by the term converter. This converter (both of them) is taking the XMI output from an UML tool, converting it, and passing it to the management interface of the SLD server. In order to do this, the converter obviously has to support the other direction as well, meaning to query the SLD server and generate an UML diagram with the retrieved information.

To summarize, the overall idea is, that developers at SAP should be able to manage their IT environment via the graphical UML notation. To realize this, a converter has to be build to which retrieves the information from different UML modeling tools and incorporates them in the SLD server. This document is attempting to supply a basis for the development of such a converter.

The actual implementation will be done in [AK06], the study thesis providing the framework in which this document is developed.

2 THEORETICAL FUNDAMENTALS

This chapter gives short introductions to the different models and encoding specifications used in this scenario. As detailed explanations would go beyond the scope of this document, this chapter can be used to refresh knowledge about the mentioned standards. Further information can be retrieved from the references in the appendix.

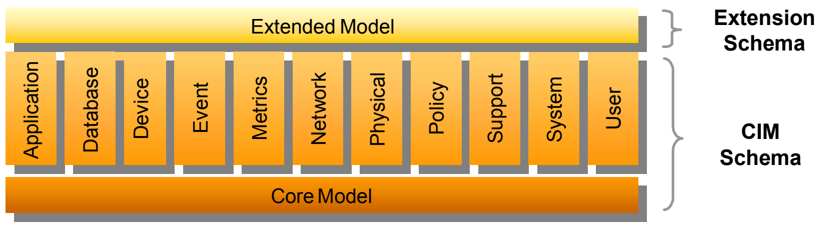
2.1 Common Information Model CIM

The following short introduction to key aspects of CIM is of course not able to explain the complex CIM management standard for readers new to it. Further information for these readers can be found in the CIM standard itself, see [DMTF], or in other references, i.e. [DMTF-CIM-T03] [C&M-I-MAN]. For a refreshing of the memento of already familiar readers, and to put the focus on certain aspects of CIM, the following two slides highlight some aspects of CIM.

(1) Generic model to describe all aspects relevant to manage a computer environemt

(2) Dependencies between different management areas can be traced

(3) Object oriented model (with familiar techniques like inheritance, abstraction and familiar elements like classes and objects)



Information 4: THEORETICAL FUNDAMENTALS - Common Information Model (CIM)

(1) Hardware, software, applications, networks, devices …. The specification of CIM is described in Managed Object Format (MOF), which is relatively easy to read. Due to the amount of standards this document already has to introduce, and as MOF is only of less importance for the rest of the document, an introduction to it is omitted here. Further information can for example be found in [DMTF-CIM-T03].

(2) Only one model has to be understood, all information can be found in one place [C&M-I-MAN].

(3) As all information is in one place, the user sees all influences at a device immediately and can therefore recognize dependencies which might otherwise become overlooked.

(4) The Reason why CIM is so powerful is that there are some very abstract classes (Core Model), which can be inherited to get more useful classes (12 Common Models above the Core

Model) and these more useful classes can finally be used to derive ready to work with classes (Extension Schema).

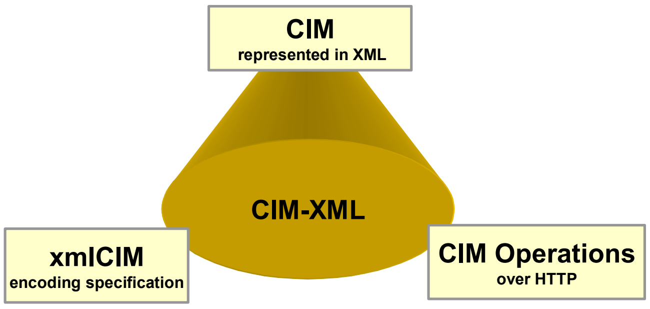
The next slide is introducing a transport protocol called CIM-XML, which is becoming prominent as the standard for exchanging information in CIM. Reasons for this are that CIM-XML is on the one hand based on the very popular XML as file format, and on the other hand on the widespread HTTP transport protocol. CIM-XML is introduced here, as it is the intended protocol for exchanging information between the desired converter and the SAP environment.

(1) Meta schema mapping of CIM to XML

(2) xmlCIM : XML grammar which can represent CIM elements (classes, qualifiers ..) for CIM-XML

(3) Note the confusing naming difference xmlCIM vs. CIM-XML

(4) Loosely valid : if element is not defined, the xmlCIM document can still be correct



Information 5: CIM-XML

(CIM-XML) As mentioned, CIM-XML is the DMTF standard transport protocol for exchanging information about CIM. The DMTF however expresses this slightly more sophisticated as CIM-XML is "one example of a WBEM protocol, CIM-XML includes the Representation of CIM using XML, written in Document Type Definition (DTD), and CIM Operations over HTTP, a transport mechanism for WBEM". From the standards mentioned in this chapter, CIM-XML is highly relevant, as it is the target format of the desired converter.

(xmlCIM) the actual encoding of CIM-XML. Examples for different elements of CIM can be seen in the next chapter.

(CIM Operations over HTTP) the irrelevant part of CIM-XML for the purpose of this document. For completeness, HTTP Operations include things like create, delete and update CIM instances.

(1) This means that there is only one schema in XML for CIM, and both classes and instances have to be (loosely) valid with respect to this schema.

(2) As shown in the graphic, xmlCIM is a part of CIM-XML. xmlCIM is the actual encoding of CIM elements in XML [DMTF-CIMXML2.2]. An example of this encoding is presented in Information 6: CIM-XML Example Code.

(3) Note the confusing naming difference between CIM-XML (the whole) and xmlCIM (part). CIM-XML is more than just the encoding, as it also includes CIM Operations and HTTP encapsulation [Da02].

(4) Loosely valid might become important for the desired converter. It means that a CIM-XML file is valid, if all contained elements are valid according to the CIM-XML XML schema, or are not specified by the XML schema. The advantage of this is that it can be used to enforce the use of certain elements. A simple example in an IT management scenario would be that every router has to have a status attribute, which indicates if the router is operable or not. A corresponding CIM-XML file would be rejected, if it doesn't contain the status attribute. The term loosely comes into play, as it doesn't matter how many other information the router is supplying in the CIM-XML file, the file is valid, if all necessary attributes are there.

For the sake of simplicity in the rest of the document only CIM-XML will be used to refer to the representation of CIM in XML. There are three reasons for the decision to use CIM-XML rather than xmlCIM. The first reason is that, xmlCIM is part of CIM-XML; however the author has experienced quite a lot of confusion caused by this naming schema. Secondly, this document is developed only as basis for a study thesis (of course yet unpublished, as this document is providing the basis), and there CIM-XML is used consistently. Finally, there is doubt whether such a confusing name schema will become standardized; currently most references to standards on the DMTF webpage are described with "Representation of CIM in XML".

Before it is preceded to the UML standard, the next information is depicting some example code how CIM-XML actually looks like. As the meaning of the code is discussed in detail in chapter 3, here only few information of the meaning of the code is given.

```
(1) <CLASS NAME =„ISWA_CoursewareRepository“>
    (1) <PROPERTY NAME=„RepositoryCounter“ TYPE=„uint16“>
    (2) <QUAILIFIER NAME=„Counter“ TYPE=„boolean“>
                  <VALUE>FALSE</VALUE></QUALIFIER>
    (3) </PROPERTY>
(2) </CLASS>
```

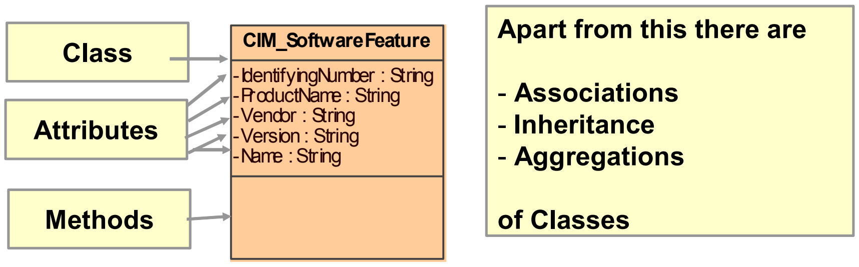
Information 6: CIM-XML Example Code

As noted, here only a short description of the code. The shown CIM-XML is representing a class, clearly indicated by the opening and closing class tag. This class has one property, or attribute called `RepositoryCounter`. As this property is (for humans pretty obvious) representing a counter, a so called qualifier is added, which is of the type counter, and exactly expressing this meaning. However, due to the qualifier, this meaning can now be comprehended by algorithms without human interaction.

2.2 UML

UML is intended to be the final front end to the user. The reason is that UML is a widespread visual modeling tool, with all the advantages visual modeling is providing, for example tracing of associations, grouping of elements and simple drag and drop operations. Probably UML is familiar to most readers, but to concretize it, some key information here.

- (1) Unified Modelling Language UML
  - (1) Industry standard language for graphical software modelling
- (2) XML Metadata Interchange XMI
  - (1) More details on the next slides
- (3) UML class diagram most relevant for this scenario



Information 7: UML and XMI

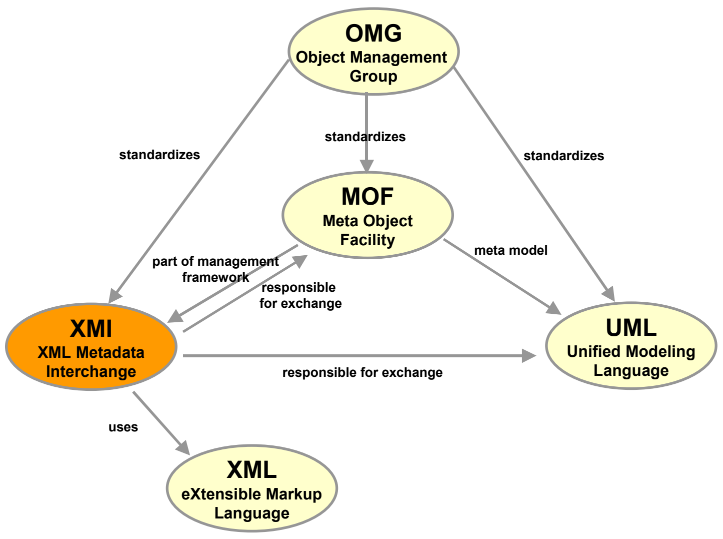
(1) UML is a very popular standard for graphical modeling; it defines many different diagram types. These diagram types are either structure diagrams, behavior or interaction diagrams. Structure diagrams are for example class, object and component diagrams, whereas use case, activity and state machine are behavior diagrams. As a last component, there are the sequence, communication, timing, and interaction overview diagram which form the interaction diagrams. Yet, for the purposes of this scenario only the class diagram is relevant.

(2) XMI is the widespread format for saving UML documents. As noted, XMI is based itself on XML. More details to XMI are given in the next subchapter. Examples of how UML looks like in the XMI file format can be seen in the next chapter.

(3) Brief review of the relevant parts of a class diagram. There are classes, which can have both attributes and methods. These classes are visualized by rectangles, with two separation lines. In the top area only the class name is given, in the middle area the attributes are declared, and in the bottom section methods would be declared, however, in this example, the class has no methods.

2.3 XMI

XML Metadata Interchange is a format for the exchange of the Meta-Object Facility (MOF) specification specified by the Object Management Group (OMG). The next slide is visualizing the connections which exist between all these organizations and standards, especially the connection between XMI and UML, which is of main interest here.



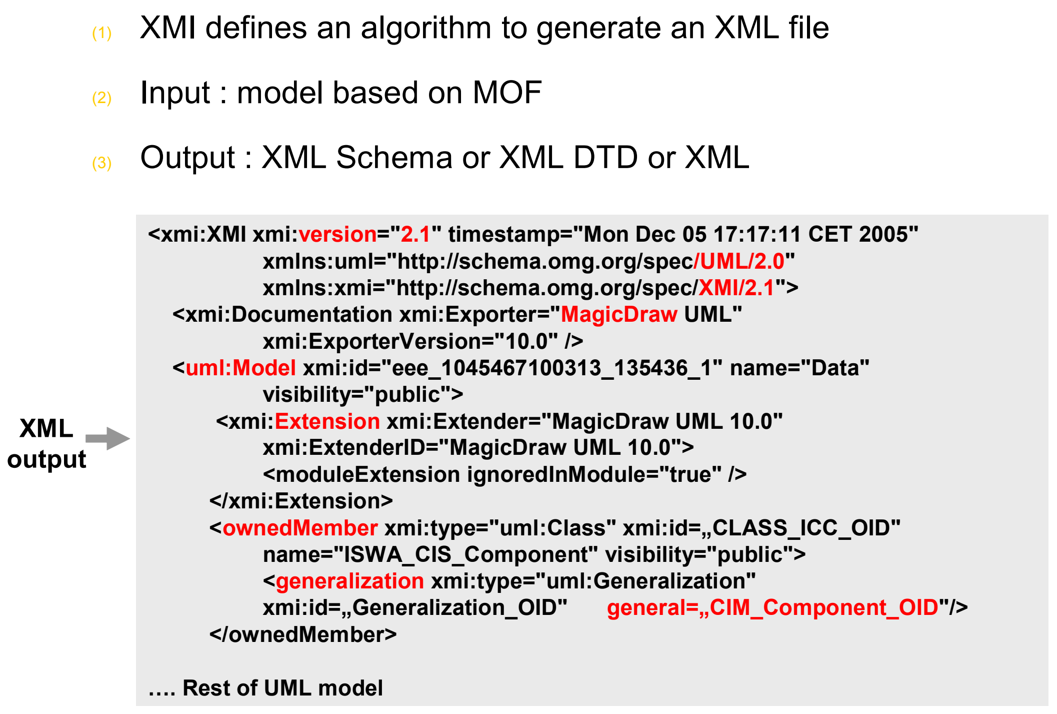
Information 8: XMI – Abstract Connections

(MOF) This specification will not become explained here, as it is firstly far too abstract to be relevant for the desired converter and secondly its naming is conflicting with the Management Object Format, which is equally shortened to MOF. However, the idea of the Meta-Object Facility is the important thing to remember. MOF is both a language to describe another meta model, and a framework for the management of the other meta model. To exemplify these highly abstract connections, the just discussed UML is such a described model, and XMI is part of the management framework for UML. Due to these issues, the XMI specification is actually part of the MOF specification, and responsible for the exchange of MOF meta models.

(XMI) XML Metadata Interchange is mapping on a technical level from MOF to XML data types. As explained above, it is responsible for the exchange of meta data. As UML can be described in MOF, information about UML can be exchanged through XMI.

(UML) Already mentioned in the last slide. It is shown here to highlight the connection to XMI. This connection is the relevant connection for the intended converter, as the information from the UML diagram can be transferred to XMI, from where it can be further processed.

After the intention of XMI has been clarified, as well as its connection to UML some more details about XMI itself. There is still one point which has to be mentioned. XMI is commonly referred to as interchange format, but what the XMI specification essentially describes is an Algorithm! [OMG_XMI2.0][Ma04], not tags for UML elements! Due to this, several vendors of UML modeling tools have different implementations of XMI, hence there is in general no way to propagate UML models from one tool to another. This issue will be dealt with at the end of chapter 3, and again in chapter 4. Before it is preceded with the concrete mapping from UML via XMI to CIM-XML, the next slide is showing a short example how XMI can look, and states again, that XMI is an algorithm.



(1) XMI defines an algorithm to generate an XML file

(2) Input : model based on MOF

(3) Output : XML Schema or XML DTD or XML

XML output

```
<xmi:XMI xmi:version="2.1" timestamp="Mon Dec 05 17:17:11 CET 2005"
         xmlns:uml="http://schema.omg.org/spec/UML/2.0"
         xmlns:xmi="http://schema.omg.org/spec/XMI/2.1">
  <xmi:Documentation xmi:Exporter="MagicDraw UML"
         xmi:ExporterVersion="10.0" />
  <uml:Model xmi:id="eee_1045467100313_135436_1" name="Data"
         visibility="public">
      <xmi:Extension xmi:Extender="MagicDraw UML 10.0"
         xmi:ExtenderID="MagicDraw UML 10.0">
         <moduleExtension ignoredInModule="true" />
     </xmi:Extension>
     <ownedMember xmi:type="uml:Class" xmi:id=„CLASS_ICC_OID"
         name="ISWA_CIS_Component" visibility="public">
         <generalization xmi:type="uml:Generalization"
         xmi:id=„Generalization_OID"    general=„CIM_Component_OID"/>
     </ownedMember>

.... Rest of UML model
```

Information 9: XMI – An Algorithm

(1) As mentioned, XMI is a responsible for the exchange of data between MOF models. It is quite astonishing; that XMI itself does not specify the tags shown on the slide to represent for example UML models, rather it is an algorithm that automatically derives appropriate tags from the meta model of UML.

(2) The input to the XMI algorithm is a meta model specified in MOF.

(3) The output of the XMI algorithm is an XML file. If the input was a meta model without data, the output is either an XML Schema, or an XML DTD. If the input was already a model which contained data, the output is a regular XML file.

The part with grey background is an example, how the output of the XMI algorithm can actually look like. The exact meaning of the elements is not important at this moment, but notice the elements highlighted in red. They contain the main information. At the beginning you see that this document is an XMI document, and that it was produced by the specification 2.1. The next highlighted part then shows, that this XMI document describes an UML model with version 2.0. The actual elements of the UML model are described after the tag `<uml:Model` .

3 MAPPING UML VIA XMI TO CIM-XML

This chapter is intended as a walkthrough of the different steps in the scenario, with a working converter. However, as this converter doesn't exist at the moment, it shows the different representations of the modeled information. The mapping between the representations is partly done manually, under assumptions how the converter could work, and of course, with respect to the restraints the different specification enforce on the information.

Before we start with the manual step by step, the next slide is giving a rough overview of what will be done in the different steps.

(1) GOAL : Examining the different representations of CIM Elements

(2) Checklist of CIM Elements which become examined

(3) Example CIM Model in UML from the c&m context

(4) MagicDraw XMI Representations of
    (1) CIM Classes, Hierarchies, Properties and Associations

(5) CIM-XML Representations of
    (1) CIM Classes, Hierarchies, Properties and Associations

(6) Differences between Magic Draw and the XMI Standard

Information 10: MAPPING UML VIA XMI TO CIM-XML - Overview

(1)(GOAL) The key idea in this chapter is to review the different representations of CIM in different formats. Here, three formats are analyzed. First, a UML diagram is presented, which is due to its visualization strengths the preferred format for human interaction. Secondly, XMI is analyzed as the standard format for saving UML diagrams. And finally, the data format of CIM-XML is discussed more in detail.

(2) An overview of the main modeling elements of CIM is given in the form of a checklist. The checklist is derived from the meta model of CIM or contains elements that are implicitly included in CIM due to its object oriented approach. The elements which will be covered by this document are printed in bold.

(3) Before we can look at representations of CIM in different formats, we need to have an example CIM model, which can then be transferred to the other two formats. As this document is developed in the context of the Internet Systems and Web Applications lecture at the University of Karlsruhe, the example CIM model is chosen from the standard scenario in this lecture, the c&m environment, as this will be common to most of the readers. For further information about the c&m environment, it is referred to the lecture [C&M].

(4) If we are modeling UML, we have to choose an UML modeling tool. Due to the fact that the widespread Microsoft solution for doing this does not by default allow saving UML models in XMI, another modeling solution was chosen. MagicDraw was selected, as it is both a powerful UML tool, and easily available at the time this document was designed. Because of the still improvable compatibility of different UML tools, in the last section of this chapter the differences between MagicDraw and the XMI Standard representation are discussed.

(5) CIM-XML as transport protocol was already included in the initial task description; hence it is not in the scope of this document to provide a detailed discussion why CIM-XML is used here. But to confute that this choice is arbitrary, some arguments why CIM-XML is a reasonable specification for this task. It was chosen because there is at the moment no accepted standard for the exchange of CIM information, but CIM-XML is promising to fulfill that role. Additionally, there are already many XML tools available, which give significant alleviation with the manual working with CIM-XML files.

(6) Finally, differences between Magic Draw and the XMI standard become analyzed. There is actually no real specification, how UML has to be saved in XMI, but more about this issue in the chapter itself.

3.1 Checklist of CIM Elements which become examined

(1) **Class**

(2) **Hierarchy**

(3) **Properties**

(4) Methods

(5) **Associations**

(6) **Aggregations**

(7) **Hierarchy of Associations**

(8) **Qualifiers**

(9) **Flavors**

(10) Trigger

(11) Schema

(12) Reference

(13) NamedElement

(14) Indication

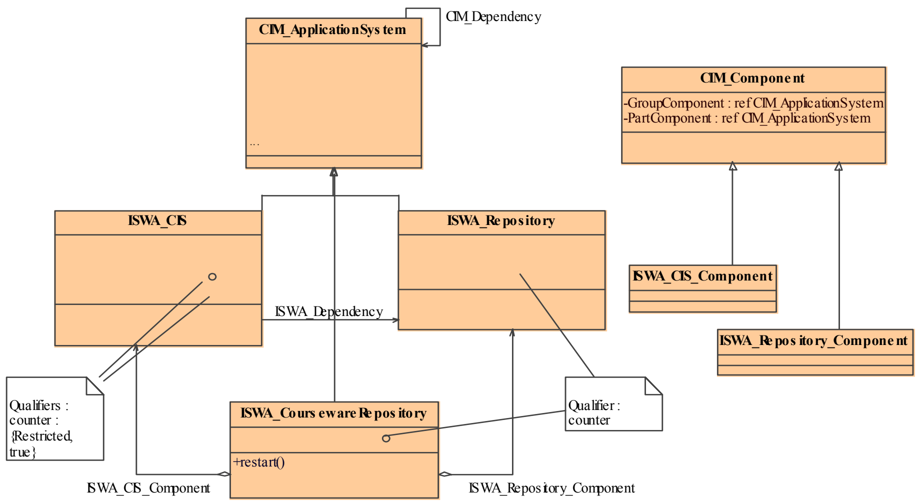
Information 11: Checklist of CIM Elements

This slide is a checklist of the different modeling elements in CIM. The elements are either taken directly out of the CIM meta model (class, property …), or are aspects relevant to developers which are implicitly included in CIM due to the object oriented approach of CIM (hierarchy, aggregations, …).The elements which will be of relevance in the next steps, are printed in bold. The rest of the elements are not discussed further, as they are either elements which are implicit in discussed elements (NamedElement, Reference, Schema) or are simply ignored, as they are not as intensively used as the chosen elements. Another reason is that the elements are modeled quite similar to the discussed elements; hence it suffices to show the concept on some of the elements.

In the next step the example CIM model is introduced as basis for the further analysis.

3.2 Example CIM Model in UML from the c&m context

The following slide shows a UML model how it could occur in the scenario. It is important to notice, that the UML model is only displaying management aspects. However, in this description some functional aspects of the shown applications are mentioned, to facilitate the tracing of how these components work together. There are three different applications in the c&m environment (not represented); their management infrastructure is derived from the **`CIM_ApplicationSystem`** class. The central application is the **`ISWA_CoursewareRepository`**, which is an aggregation of the other two application systems, the **`ISWA_Repository`** and the **`ISWA_CIS`**. CIS is standing for Content Information Service and is a Web Service providing functionality for the upload and provision of so called Living Documents. The **`ISWA_CoursewareRepository`** is considered closer from the perspective of management. In this document, it is less relevant, what the applications are actually doing, but more important, if they are doing their job, and if they are busy at the moment. These are important aspects, as for example the **`ISWA_CoursewareRepository`** is provided by an external company (ed.serv) to c&m. Of course, there are Service Level Agreements between c&m and ed.serv regarding the quality and quantity which has to be delivered by ed.serv. It is obvious, that ed.serv needs capabilities to monitor its systems for high traffic and to have controlling functionality to prevent a breakdown of the **`ISWA_CoursewareRepository`**. Otherwise, ed.serv would risk contract fines.


CIM_ApplicationSystem
CIM_Dependency
...
CIM_Component
-GroupComponent : ref CIM_ApplicationSystem
-PartComponent : ref CIM_ApplicationSystem
ISWA_CIS
ISWA_Repository
ISWA_CIS_Component
ISWA_Repository_Component
ISWA_Dependency
Qualifiers : counter : {Restricted, true}
ISWA_CoursewareRepository
+restart()
Qualifier : counter
ISWA_CIS_Component
ISWA_Repository_Component

Information 12: CIM Model in UML

`(CIM_*)` As the scenario is concerned about managing according to the CIM, classes have to be derived from abstract classes of the CIM schema. To see how this works, these abstract classes have been included with the prefix CIM_ in the UML diagram.

`(ISWA_*)` The actual classes which can be instanced are prefixed with **`ISWA_`**. In real world applications, ISWA should be a world wide unique identifier; here the ISWA acronym for Internet Systems and Web Applications has been used for the sake of simplicity.

(Comment) Specialties of CIM are qualifiers, which prefix elements, and are used to give more detailed information about their meaning. At the moment they are just noted in comments, this issue will be discussed more in depth in one of the next slides.

(Associations) Note that all relevant association types (inheritance, aggregation, simple association) have been included in this UML diagram. In the CIM data description, each association has to be modeled by a separate class. In the shown UML diagram, this has been neglected for the purpose of lucidity. The classes of the associations have been modeled where necessary, for example the three rightmost classes are classes which represent associations. In a "real" UML diagram, however, each association has to be represented by a class, as only by doing this the developer can handle the associations in the way intended by the CIM data description.

`(CIM_Component, ISWA_CIS_Component)` Special attention is drawn to these classes in the right part of the model, as these two classes are representing an association in the left part of the diagram. This construct is needed to model the hierarchy of associations.

Relevant aspects for a successful automatically mapping of UML to CIM-XML are analyzed in the next step, the representation of the UML diagram from this slide in MagicDraw XMI.

3.3 MagicDraw XMI Representations of CIM elements

This step is selecting different elements and showing their encoding in XMI. For reference to the UML diagram, the specific element from which the code is taken is given in the description of the slide after the trail. Before the trail the analyzed model element is given. Class is selected as the first element, and depicted in the next slide. Key parts of the code have been printed in red to highlight, which elements are connected to the class.

(1) <ownedMember xmi:type="uml:Class" xmi:id=„CLASS_ICR_OID" name="ISWA_CourseWareRepository" visibility="public">

- (1) <owned**Attribute** xmi:type="uml:Property" xmi:id=„Att_OID" name="**RepositoryCounter**" visibility="private" type=„uint16_OID"/>
- (2) <ownedAttribute xmi:type="uml:Property" xmi:id=„Att_OID" aggregation="shared" visibility="private" **association=„ISWA_Repository_Component_OID„** type=„CLASS_ISWA_Repository_OID"/>
- (3) <ownedAttribute xmi:type="uml:Property" xmi:id=„Att_OID" aggregation="shared" visibility="private" **association=„ISWA_CIS_COMPONENT_OID"** type=„CLASS_ISWA_CIS_OID"/>
- (4) <owned**Operation** xmi:type="uml:Operation" xmi:id=„Op_OID" concurrency="sequential" name="**restart**" visibility="public"/>
- (5) **Qualifier** (Comment) is missing,

(2) </ownedMember>

Information 13: XMI CIM Class – ISWA_CoursewareRepository (ICR)

(1)(2) Relevant for the class itself are only point (1) and (2). In general, a class is saved in an ownedMember tag as a UML class, with a unique ID, and a name. After the elements which are belonging to this class have been listed, the ownedMember tag is closed. The code between the **`owned member`** tags will be discussed separately in the next slides. However, it has been included here to show how the connection is made from the class to the elements it contains.

Next the representation of a generalization is shown on the example of the **`ISWA_CIS_Component`** class, which is very simple.

(1) <ownedMember xmi:type="uml:Class" xmi:id=„CLASS_ICC_OID" name="ISWA_CIS_Component" visibility="public">

- (1) <**generalization** xmi:type="uml:Generalization" xmi:id=„Generalization_OID" **general=„CIM_Component_OID**"/>

(2) </ownedMember>

Information 14: XMI CIM Hierarchy– ISWA_CIS_Component (ICC)

(1)(2) The generalization has to be part of a class.

(1.1) Here is shown how the inheritance is saved in Magic Draw XMI. A specific generalization tag is used, from type generalization, with a unique ID, and a general attribute, which is representing the ID of the more general class, **`CIM_Component`**. So it can be seen, that a generalization is basically just an attribute of the inherited class.

In the subsequent step, the realization of attributes is given.

```
(1)  <ownedAttribute xmi:type="uml:Property" xmi:id=„Att_OID"
     name="RepositoryCounter" visibility="private" type=„uint16_OID"/>

(2)  <ownedAttribute xmi:type="uml:Property" xmi:id=„Att_OID"
     name="AverageThroughputLastHour" visibility="private">
     (1)  <type xmi:type="uml:DataType"
          href="UML_Standard_Profile.xml|eee_1045467100323_385364_62">
     (2)  <xmi:Extension xmi:Extender="MagicDraw UML 10.0"
          xmi:ExtenderID="MagicDraw UML 10.0">
     (3)  <referenceExtension referentPath="UML Standard Profile::UML Standard
          Profile::datatypes::float" referentType="DataType"/>
     (4)  </xmi:Extension>
     (5)  </type>
(3)  </ownedAttribute>
```

Information 15: XMI CIM Properties – Different Classes

(1)(2) Essentially, there are two different ways how attributes are saved in Magic Draw XMI, depending on whether they are UML standard type or not. In both cases, the **`ownedAttribute`** tag is used, with unique ID and name. However, the type is fundamentally different. It is not shown in the slide, but this tag has to be part of a class, respectively an **`ownedMember`** tag.

(1) If the type of the attribute is not known to the UML modeling tool, it creates the **`attribute`** tag, where the type actually is a reference to another class, which is created automatically, when an attribute gets assigned such a value. These classes are invisible in the diagram, but visible in the XMI file. They are of no further meaning, as an attribute from type **`uint16`** is already containing all meaning relevant for further processing.

(2) If, however, the type of the attribute is know in the UML modeling tool, the code becomes slightly blown up, as can be seen in the slide by the example of the attribute from type float. But it is probably sufficient to extract the string after "**`data types::`** " and before " **`"`** ".

Now the transcription of associations is demonstrated. Note that the color highlighting has changed, in order to facilitate visual tracing of IDs.

```
(1)  <ownedAttribute xmi:type="uml:Property" xmi:id=„ATT_ICR_OID"
     aggregation="shared" visibility="private"
     association=„ISWA_Repository_Component_OID„
     type=„CLASS_ISWA_Repository_OID"/>
                                              Link information
                                              is found here

(2)  <ownedMember xmi:type="uml:Association"
     xmi:id=„ISWA_Repository_Component_OID"
     name="ISWA_Repository_Component" visibility="public">

  (1)  <memberEnd xmi:idref=„ATT_ICR_OID"/>
  (2)  <memberEnd xmi:idref=„ownedEnd_OID"/>
  (3)  <ownedEnd xmi:type="uml:Property" xmi:id=„ownedEND_OID"
       visibility="private" association=" ISWA_Repository_Component_OID "
       type=„CLASS_ICR_OID"/>

(3)  </ownedMember>
```

Information 16: XMI CIM Association – ISWA_Repository_Component

(1)(2)(3) Fundamentally, an association is made up of two parts, an attribute in one class referring to another class, which has the special type association. The separate association class contains quite a lot of code, though everything but the name of it can be ignored by the desired converter.

(1) This point describes the attribute, which is part of a common class. As seen before, it is an **ownedAttribute** tag, with unique ID and name. Yet, the type differs a little bit, as actually the attribute is from the type of the other class. This is already the information we need to link the association, meaning to extract the other end of this association. Additionally, there are two further attributes. "**aggregation**", which is displaying that this association is an aggregation, and "**association**" which is a reference to the separate association class.

(2) This is the separate class which is modeling the association; therefore it is of the type **association**. As mentioned in the description of the UML diagram, there has to be an association class for each association. It has three attributes, two **memberEnds** tags and one **ownedEnd** tag, but this information is not really needed. We already know (see (1)), to which class this association is linking, therefore it suffices to extract the name.

Following the representation of CIM qualifiers is sketched, but before this can be done, it is clarified why standard comments have been chosen to represent qualifiers in UML, because unfortunately there is no equivalent model element in UML for CIM qualifiers. Qualifiers are basically keywords which are added to classes, attributes, parameters, methods and references. Qualifiers do further define the element they are added to. In the scenario the counter qualifier is used to indicate that the specific attributes are counters. There are many different qualifiers, for example description, association and so on, and it is even possible to define new qualifiers.

(1) In general, unsolved problem how to model qualifiers in UML adequately.

(2) From three possibilities (visible comments, invisible comments, stereotypes) the visible comments have been chosen.

(3) Visible comments/notes are not saved in the Magic Draw UML Model, they are saved in the <xmi:Extension xmi:Extender="MagicDraw UML 10.0">

(4) Way to retrieve qualifiers, start from the comment in the extension model
   - (1) Search for keyword Qualifier
   - (2) Search for OID of the corresponding NOTE
   - (3) Retrieve NOTE anchor element
   - (4) Search linkFirstEnd OID of the note anchor, retrieve another element in the extension model
   - (5) Search the xmi:idref OID
   - (6) Finished, you have found the property the qualifier is describing

Information 17: XMI CIM qualifier – Property RepositoryCounter

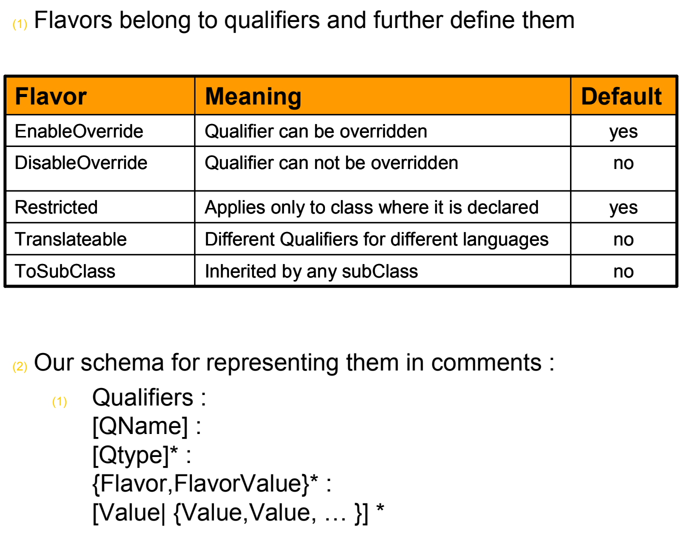
(1) As mentioned, there is no specific element in UML which captures the idea of qualifiers fully. Accordingly, a workaround has to be found, what to do exactly will be discussed in the next points.

(2) This document identified three different methods to model qualifiers in UML. First of all, there are the standard comments, which can be enhanced, or misused, depending on the point of view, with a pattern that allows describing qualifiers with them. Secondly, there are the invisible comments. They are in no way visible to the user, and have in the same way as the visible comments become enhanced or misused with a specific pattern. This kind of comments is called invisible in this document, as they are not visible if the whole UML model is shown. They can only be retrieved, or modified if the specific element which is described by such an invisible comment is double clicked. The last possibility is to use stereotypes, which basically are constructs like `<<counter>>` that become added to a class name, an attribute name, whatever you want.
Despite the third possibility seems to be the usual solution, here the approach with visible comments is chosen. This is due to the fact, that there is no possibility to create new qualifiers (meaning qualifiers that are not in the CIM schema) in the stereotype approach. This is possible in the CIM meta schema, though, and hence has to be possible in UML. From the two different kinds of comments, visible comments have been chosen, as no end user would want to work with a UML diagram, where half of the information is invisible.

(3) Despite the reasons why the visible comments have been chosen, there is a big drawback when using them in Magic Draw. They are not saved in the core of the UML model; rather they are saved in a proprietary part of the XMI file, where all the geometric information is stored. Tags in this proprietary part are prefixed with md, which is due to the fact that in this step by step walkthrough MagicDraw (namespace for MagicDraw specific XMI parts md) has been used. Further efforts have to be done to clarify how comments are handled in other UML tools, but for this chapter, the work is continued with what MagicDraw is providing.

(4) The drawback of visible comments is depicted in this point and its sub points, as the developer of the converter has to move hand over hand through the XMI code to align the comments with what they are actually describing. Though this is quite demanding for the developer, this should make no problems for the end user.

It has been said, that it should be possible to introduce new qualifiers directly in UML, unfortunately this is resulting in further effort, as if new qualifiers are allowed, there have to be flavors as well. What flavors are is shown in the next slide.



(1) Flavors belong to qualifiers and further define them

Flavor	Meaning	Default
EnableOverride	Qualifier can be overridden	yes
DisableOverride	Qualifier can not be overridden	no
Restricted	Applies only to class where it is declared	yes
Translateable	Different Qualifiers for different languages	no
ToSubClass	Inherited by any subClass	no

(2) Our schema for representing them in comments :

(1) Qualifiers :
[QName] :
[Qtype]* :
{Flavor,FlavorValue}* :
[Value| {Value,Value, … }] *

Information 18: XMI CIM flavor – Property RepositoryCounter

(1) Flavors are attributes of qualifiers, and define them further. What these flavors can look like is shown in the corresponding table. There are only the five types of flavors listed here allowed in CIM. For example, the `ToSubClass` flavor is indicating, that subclasses of this class are inheriting the specific qualifier. If you don't specify it, subclasses don't have this qualifier, as the default of this flavor is no.

(2) If qualifiers are represented in comments, there has to be a distinction between regular comments, and qualifier comments. Therefore a pattern has to be used, if a comment is describing a qualifier. The shown pattern is inspired by the Managed Object Format (MOF) specification of CIM itself, but it has an essential difference. In the CIM specification, the definition and the use of qualifiers is separated. In the pattern suggested here, this is not the case, to supply the user immidiately with all needed information. The identifying key is the `qualifiers` statement as the first word in the comment. After this, separated by colons, the name and the type (optional, if not specified by the CIM schema) of the qualifier is given, followed by flavors, if necessary (if not using a qualifier with defaults from the definition in the CIM schema, note that this default doesn't have to be the default given in the table of this slide), followed by the value of the qualifier.

The interested readers might have noticed the mismatch between this complicated pattern, and the qualifiers shown in UML diagram. This is due to the fact, that the last three parts of this pattern are optional, and may be ignored, if a standard flavor is used with its default flavors.

3.4 CIM-XML Representations of CIM elements

After the previous step has displayed how CIM elements are saved in XMI, now the representation in CIM-XML is shown. Luckily, as this is designed especially for CIM, the representations are much more readable and compact. So the next slide already depicts the representation of classes, properties and qualifiers (the definition of new qualifiers is different, though).

```
(1) <CLASS NAME =„ISWA_CoursewareRepository“>
      (1) <PROPERTY NAME=„RepositoryCounter“ TYPE=„uint16“>
      (2) <QUAILIFIER NAME=„Counter“ TYPE=„boolean“>
                   <VALUE>FALSE</VALUE></QUALIFIER>
      (3) </PROPERTY>
(2) </CLASS>

(3) <CLASS NAME =„ISWA_CIS_Component
        SUPERCLASS=„CIM_Component“>
(4) </CLASS>
```

(5) There are no attributes from the associations and aggregations in these classes.

Information 19: CIM-XML Class, Property & Qualifier

(1)(2) Classes are presented by the **`class`** tag, properties are represented by the **`property`** tag, and qualifiers are represented by the **`qualifier`** tag. The qualifier tag has to include an additional **`value`** tag, giving the value of the qualifier.

(3)(4) This class is an example for inheritance in CIM-XML. All which has to be done is to add a **`superclass`** attribute to the **`class`** definition tag.

(5) One very important aspect is missing, and partly causes the short code length. There is no reference to the associations. How they are modeled is shown in the next slide.

```
(1) <CLASS NAME=„ISWA_Repository_Component“
        SUPERCLASS=„CIM_Component“>

(2) </CLASS>
```

Information 20: CIM-XML Associations - ISWA_Repository_Component

(1) (2) Wrapping around the rest of the code is the class, which is representing the association. That this association is derived from another association is realized by just adding the attribute **`superclass`**.

(1.1) The first qualifier is indicating that this class is an association, hence the qualifier is called association. It is from the type **`boolean`**, and its value is **`true`**. The value of a boolean qualifier is of less importance, as it only restates: yes, this is an association.

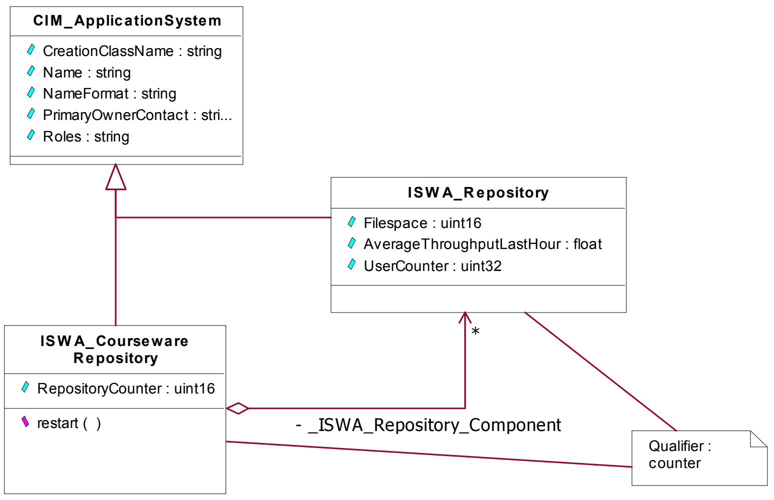
(1.2) The second qualifier is used, as this is not just a regular association, rather it is an **`aggregation`**. Hence the aggregation qualifier is used again from type boolean.

(1.3)(1.4) After it is now clear, that this class is an aggregation, the references are declared by the **`property.reference`** tag. The references are given names (**`PartComponent`**, **`GroupComponent`**), which in the general case can be chosen freely, yet not here, as this association is derived from another association. There is a **`property.reference`** tag for each end, including the ID of the class. Finally, these references are propagated from the **`superclass,`** hence they have the propagated value true.

3.5 Differences between MagicDraw and the XMI Standard

Up to now there is no established standard which defines exactly all details how an UML model is saved in XMI. Hence there are many different XMI formats, and none can claim to be the standard. The OMG has tried in 2001 to overcome this by specifying the diagram interchange format for UML, which is nothing else than their result if they apply the XMI algorithm to the UML meta model [OMG-DIF]. But the interchange format from 2001 had a significant weakness, as UML itself is not concerned with its graphical layout; the diagram interchange format specified nothing which could be used to exchange the actual layout of UML models from one tool to another. Mainly due to this, the suggested diagram interchange format didn't become accepted. With the new upcoming standard UML 2.0 (2005/2006), the OMG again tries to specify a new diagram interchange format. This time it specifies all aspects of UML, including the graphical representation [OMG-DIF]. Yet, this standard is still based on XMI 1.2, and actually it is still based on UML 1.4, but claimed to fit also without change on UML 2.0. The point here is that there is currently no standard how UML is completely specified in XMI.

Due to this, there can't be a real comparison to the XMI standard. As a solution, the XMI code of another UML tool is analyzed, and compared to Magic Draw XMI. The following slide shows the part of the previous UML we will focus here, and notes again, that this is currently the only solution to evaluate, how standardized XMI is today.


CIM_ApplicationSystem
CreationClassName : string
Name : string
NameFormat : string
PrimaryOwnerContact : stri...
Roles : string
ISWA_Repository
Filespace : uint16
AverageThroughputLastHour : float
UserCounter : uint32
ISWA_Courseware Repository
RepositoryCounter : uint16
restart ( )
*
- _ISWA_Repository_Component
Qualifier : counter

(1) As mentioned, there is no real standard how UML has to be specified in XMI

(2) Solution : Compare Magic Draw XMI to Rational Rose* XMI another widely spread UML modeling tool

*** The exact title of the software is Rational XDE Developer Plus Java**

Information 21: Part of the previous UML model in Rational Rose

(1) In the introduction to XMI, it was shown, that there is no exact specification how UML is saved in XMI. The solution of this document is given in the next point.

(2) The only way to evaluate how standardized Magic Draw XMI is is to compare it to XMI of other UML tools. As Rational Rose is another widespread UML tool, it is chosen here for the comparison. To be exact, the full title of the software is Rational XDE Developer Plus Java.

Before two very important parts of UML (classes and comments) are discussed more in detail, the next slide presents some of the first impression when these two kinds of XMI are compared.

(1) Very different

(2) Magic Draw UML classes can be found nearly immediately in the XMI file, as they are close to the root element in the hierarchy
xmi -> uml:Model ->ownedMember

(3) Rational Rose UML classes can be found in the following place in the XMI hierarchy
xmi -> XMI.content -> UML:Model -> UML:Namespace.ownedElement -> UML:Package -> UML:Class

(4) Rational Rose XMI doesn‘t save any information about the graphical representation in the XMI file

Information 22: XMI Differences – First Impression

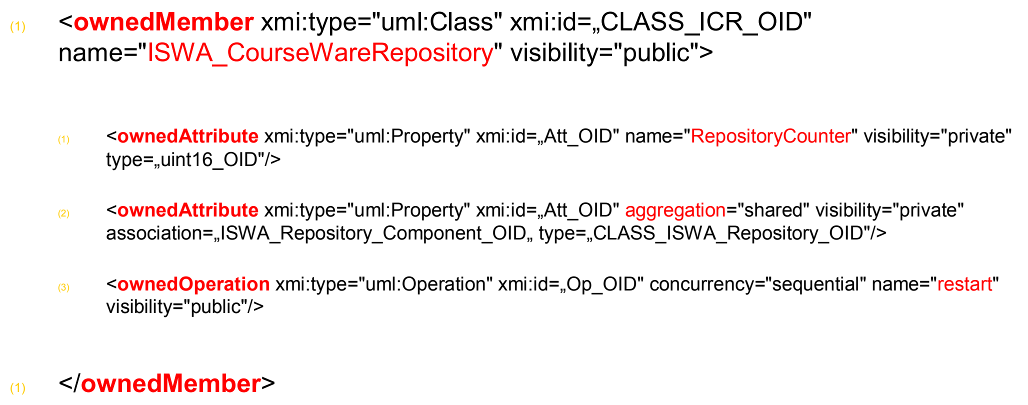
(1) The two XMI files make a rather different impression; the next points give reasons for this.

(2) When analyzing, why the files look so different, one of the first answers is that the main elements of an UML model can be found at rather different places in the XMI hierarchy. In Magic Draw XMI classes, or ownedMembers tags, are close to the root element.

(3) In Rational Rose, classes are at a very different place in the XMI hierarchy. Instead of being a third level element, they are a fifth level element. This proves that the XMI files are already completely different for very basic elements of UML.

(4) Another issue which is quite obvious is that the Rational Rose XMI file is much shorter than the Magic Draw XMI file. This is due to the fact, that Rational Rose doesn't save information about graphical layout in the XMI file. When reimporting a file exported by Rational Rose back into Rational Rose, all graphical information is lost. The structure is there, but the developer has manually to design his view again.

The next two slides are showing how an UML class is represented in the different XMI files.



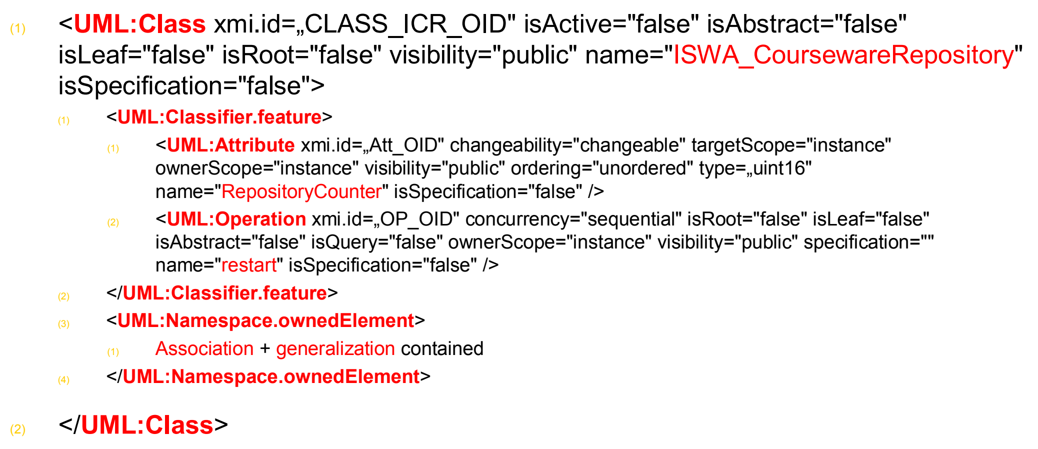
(1) <**ownedMember** xmi:type="uml:Class" xmi:id=„CLASS_ICR_OID" name="ISWA_CourseWareRepository" visibility="public">

- (1) <**ownedAttribute** xmi:type="uml:Property" xmi:id=„Att_OID" name="RepositoryCounter" visibility="private" type=„uint16_OID"/>
- (2) <**ownedAttribute** xmi:type="uml:Property" xmi:id=„Att_OID" aggregation="shared" visibility="private" association=„ISWA_Repository_Component_OID„ type=„CLASS_ISWA_Repository_OID"/>
- (3) <**ownedOperation** xmi:type="uml:Operation" xmi:id=„Op_OID" concurrency="sequential" name="restart" visibility="public"/>

(1) </**ownedMember**>

Information 23: XMI Class Magic Draw

(1)(2) The architecture of an UML class in Magic Draw XMI has already been discussed in chapter 3. This slide is only represented here for refreshing the fact, that in Magic Draw classes are `ownedMembers`, and their elements are `ownedAttributes` or `ownedOperations`. The successive slide shows in contrast how the same class is saved in Rational Rose XMI.



(1) <**UML:Class** xmi.id=„CLASS_ICR_OID" isActive="false" isAbstract="false" isLeaf="false" isRoot="false" visibility="public" name="ISWA_CoursewareRepository" isSpecification="false">

- (1) <**UML:Classifier.feature**>
  - (1) <**UML:Attribute** xmi.id=„Att_OID" changeability="changeable" targetScope="instance" ownerScope="instance" visibility="public" ordering="unordered" type=„uint16" name="RepositoryCounter" isSpecification="false" />
  - (2) <**UML:Operation** xmi.id=„OP_OID" concurrency="sequential" isRoot="false" isLeaf="false" isAbstract="false" isQuery="false" ownerScope="instance" visibility="public" specification="" name="restart" isSpecification="false" />
- (2) </**UML:Classifier.feature**>
- (3) <**UML:Namespace.ownedElement**>
  - (1) Association + generalization contained
- (4) </**UML:Namespace.ownedElement**>

(2) </**UML:Class**>

Information 24: XMI Class Rational Rose

(1) As it can be seen, classes are here saved in **`UML:Class`** tags, and not in ownedMember tags. Additionally, the tag has quite some attributes more as the Magic Draw tag, like isActive, isAbstract ….

(1.1)(1.2) Elements of a class in Rational Rose are grouped in two categories, either they belong to the **`UML:Classifier`** tag, or to the **`UML:Namespace.ownedElement`** tag. Attributes and operations belong to the classifier group. Their tag is **`UML:Attribute`**, respectively **`UML:Operation`**.

(1.3)(1.4) Associations and generalizations belong to the **`UML:Namespace.ownedElement`** tag. Their exact representation is omitted here for the purpose of readability.

A discussion of the differences between the representations can be found in chapter 4, but it is pretty obvious, that there are quite significant discrepancies. As a last point, it is analyzed how comments are saved in Rational Rose.

(1) Comments in Magic Draw are not saved in the UML:model and therefore hard to retrieve

(2) In Rational Rose, comments are directly saved in the UML:Attribute tag as shown on the previous slide as the following

(3) <UML:ModelElement.comment>

  (1) <UML:Comment body="**Qualifier : counter**" />

(4) </UML:ModelElement.comment>

Information 25: Comments in Rational Rose

(1) As mentioned, it is quite hard to retrieve a comment automatically from a Magic Draw XMI file. It is interesting to see, how Rational Rose is integrating comments in their XMI format.

(2) Happily, the retrieval of comments in Rational Rose is much easier than in Magic Draw, as comments are directly saved in the **`UML:Model`**, to be accurately a comment is saved directly in the tag which it is describing. The format of such a comment is given in the next points.

(3)(4) These points together show the structure how a comment looks like in Rational Rose XMI. Essentially, it consists out of an **`UML:ModelElement`** tag, which contains an **`UML:Comment`** tag with the content of the comment.

The last chapter now summarizes the main findings of this document, with a particular focus on the issues caused by the rather soft definition of XMI, which is the desired exchange format for the converter.

4 FINDINGS

4.1 Discussion of XMI issues

(1) XMI is the standard for saving UML, hence it is the only way to deliver information in a standized way from the CIM model in UML to the management interface of the SAP landscapes.

(2) But there are still many issues:
  - (1) XMI is no hard specification
  - (2) Different vendors have very different formats of XMI
  - (3) It has been evaluated, that the differences are significant, as the
    - (1) Structure of the whole XMI file
    - (2) Classes as core components
    - (3) Comments as provider for CIM specific properties
    - (4) … are essentially different!
  - (4) Some vendors include no information about graphical representation at all in the XMI file

Information 26: FINDINGS - XMI Issues

(1) The scenario for this document is based on the SAP landscape environment. With the aim to be as much standard conform as possible and interoperable with future applications, SAP needs to have a standardized exchange format. For UML, XMI is the standard for the exchange of information. Hence it is the first candidate for a possible synchronization between the user friendly UML model of CIM and a management server based on CIM. But there are still many barriers for this desired synchronization.

(2) This point contains relevant issues which emerge, when analyzing XMI functionality for the desired synchronization.

(2.1) First of all, the reason where most of the following problems root in is that XMI is a very soft specification. As derived in 2.3, XMI is rather an algorithm than a hard specification.

(2.2) Due to (2.1), each vendor can understand the XMI algorithm slightly different. As a result, there are various different XMI formats available, and many of them are not compatible.

(2.3) The previous items have presented, why there are different XMI formats, this item is highlighting that there are many aspects which can't be neglected. These aspects have been derived from the opposition of Magic Draw and Rational Rose XMI in 3.5. Firstly, the structure of the whole XMI file is essentially different. Secondly, the representation of each class is similarly different, and classes are central components of a UML class diagram. At last, the format of comments has been analyzed. As the two preceding items, the format of comments is highly different.

(2.4) Finally, there is the problem that Rational Rose doesn't include any information about graphical representation at all in the XMI file. This means that for Rational Rose and similar UML tools, there has to be an algorithm which automatically designs visual class diagram representation, if it is given the structure. Another possibility is that there must be an addition to the UML meta model, to incorporate the graphical representation in XMI itself. The last possibility would be, to make a human generate a visual representation, and store this somewhere else. Each time, the information is retrieved from the SAP landscape server, the retrieved information has to be merged with the existing graphical layout.

4.2 General Conclusion

(1) There are many different UML tools, that are currently not compatible to each other

(2) As long as the OMG or some other institution does not succed in establishing a hard specification for a XMI standard, the only solution is to code one converter for each UML tool that is used to modify CIM models of the SAP landscape environment

(3) The converter must provide the functionality to update graphical representations of the UML models with information retrieved from the SAP landscape server.

(4) Respectively to CIM and UML, apart from qualifiers and flavors everything can be specified in UML. These two exceptions can be modeled in UML for example as suggested here with comments

(5) However, as long as there is no integrated CIM plugin for an UML tool, it is rather uncomfortable to look up an appropriate superclass in the CIM standard for the initial modeling

Information 27: General Conclusion

(1) Alone [Je04] lists the capacities and compatibilities of over 100 different UML modeling tools.

(2) As long as there is no hard specification how UML has to be saved in XMI, including its graphical representation, there are problems when using different UML tools. The apparent solution is to develop one converter for each tool, and unfortunately there doesn't seem to be a better solution available. Obviously, there can be one converter, which is loading separate modules for each UML modeling tool, but that isn't solving the problem.

(3) As it has been shown, some UML tools don't save information about graphical representation in the XMI file. But even if they do, the converter had to fulfill the same task, as the information from the SAP landscape management server doesn't include any information about graphical layout. Apparently, the converter has to merge information from the server with a preexisting, probably hand made UML layout definition.

(4) Happily, most aspects of CIM can be modeled with elements of UML. Only qualifiers and their flavors have no specific modeling element in UML. The solution made in this document was to model them with UML comments. This solution has some disadvantages, as discussed in Information 17. Additionally, there is the problem that using comments for real modeling violates the idea of comments, and it would be nice to have specific element in UML to model qualifiers and flavors directly.

(5) If there is no specific tool for the design of CIM it is quite complicated to model CIM in UML, as the developer would have to memorize and/or look up all concepts defined in the Core Model and the relevant Common Models. Therefore a plug-in for existing UML tools is would alleviate the initial modeling of CIM models, by providing drag and drag abilities for classes, which can be accessed structured by the Common Models.

Abbreviations and Glossary

Abbreviation or Term	**Full Name and/or Term Description**
CIM	Common Information Model
	The DMTF Common Information Model (CIM) is a conceptual information model for describing computing and business entities in enterprise and Internet environments. It provides a consistent definition and structure of data, using object-oriented techniques. The CIM Schema establishes a common conceptual framework that describes the managed environment. A fundamental taxonomy of objects is defined - both with respect to classification and association, and with respect to a basic set of classes intended to establish a common framework [DMTF-CIM2.4].
DMTF	Distributed Management Task Force
	An open membership, not-for-profit consortium that produces and maintains computer industry specifications for interoperable enterprise applications.
J-Konverter	Tool which was created in the Diplomarbeit of Christop Joswig. There is no official name fort his tool, this name is only used in this work for easier referencing. This tool is taking XMI input and delivering it in to the SLD Server, and vice versa.
MagicDraw	A specific UML modeling tool, currently (Nov 2005) available in Version 10.
MOF	Managed Object Format
	The CIM Specification defines a language based on the Interface Definition Language (IDL) called Managed Object Format. The MOF syntax is a way to describe object definitions in textual form. [DMTF-CIM-T03]
	Or
	Meta-Object Facility
	Meta framework and language which supports the management of plttform independent meta models.
Rational Rose	Rational XDE Developer Plus Java
	A specific UML modeling tool, currently(Jan 2006) in version V2003.06.12
SAP	SAP (**S**ysteme, **A**nwendungen, **P**rodukte in der Datenverarbeitung)
	SAP is an acronym for "Systeme, Anwendungen, Produkte in der Datenverarbeitung", which means "Systems, Applications and Products in data processing". The company was founded in 1972 by five former IBM employees. Starting in 1972 with the provision of standard business management software applications, the company has evolved to a leading supplier of highly customized business management applications. As of 2005, SAP employs over 28,900 people in more than 50 countries. http://www.sap.com
SLD	System Landscape Directory
	The SLD of SAP NetWeaver is a repository for central storage of information about a variety of hardware and software components depending on each other.

UML	Unified Modeling Language Language to describe any objects (e.g., software systems or business units) in a semi-formal way using graphical elements.
WBEM	Web Based Enterprise Management Management Solution of the DMTF which is using standardized web technologies for management.
XMI	XML Metadata Interchange Exchange format for models between different modeling tools
XML	eXtensible Markup Language W3C recommendation for creating special-purpose markup languages; it is a simplified subset of SGML, capable of describing many different kinds of data.
CIM-XML	encoding specification of CIM which defines XML elements that can be used to represent CIM classes and Instances

Index

Information Slides

References

[AK06] Sébastien Anselment, Rudolf Krist: Entwicklung eines Werkzeugs zur Transformation von CIM-Modellen zwischen CIM-XMI-* und CIM-XML, Team-Studienarbeit, Universität Karlsruhe (TH), C&M (Prof. Abeck)

[C&M] Cooperation & Management: INTERNET-SYSTEME UND WEB-APPLIKATIONEN (ISWA) IM ÜBERBLICK, http://www.cm-tm.uka.de/iswa, Universität Karlsruhe, C&M (Prof. Abeck).

[C&M-I-MAN] ISWA Lecture IT Management
http://www.cm-tm.uka.de/547.php

[DMTF] Distributed Management Task Force
http://www.dmtf.org

[DMTF-CIM-T03] DMTF: Common Information Model (CIM) Tutorial 2003,
http://www.wbemsolutions.com/tutorials/CIM/cimtutorial.pdf

[DMTF-CIMXML2.2] Specification for the Representation of CIM in XML
http://www.dmtf.org/standards/wbem/DSP201.html

[Da02] Jim Davis, SUN Microsystems: Introduction to CIM-XML, Presentation at DMTF Developers' Conference, June 11 2002.

[Je04] Mario Jeckle, List with statements about the capacities of more than 100 UML modeling tools
http://www.jeckle.de/umltools.htm

[Jo05] Christof Joswig: UML Modeling for CIM, using XML Metadata Interchange, Diplomarbeit, Berufsakademie Mannheim, SAP Walldorf, Februar 2005.

[Ma04] Benoit Marchal, Working XML: UML, XMI, and code generation, Part 2, developer Works at the IBM internet page, IBM

http://www-128.ibm.com/developerworks/xml/library/x-wxxm24/

[OMG-XMI2.0] XML Metadata Interchange
http://www.omg.org

[OMG-MOF1.4] Meta Object_Facility
http://www.omg.org

[OMG-DIF] Diagram Interchange Format
Extension to the UML meta model with the goal to establish XMI as interchange standard between different UML tools.

http://www.omg.org/cgi-bin/doc?ptc/2005-06-04

[SAP-SLD] SAP-Library: Service Landscape Directory, http://help.sap.com/saphelp_erp2005/helpdata/en/31/f0ff69551e4f259fdad799a229363e/frameset.htm