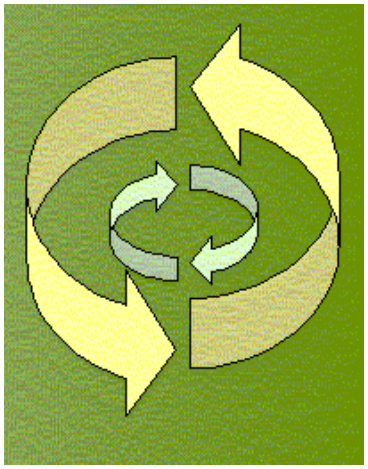
UMT

UML Model Transformation Tool

Overview and user guide documentation



UMT documentation

UMT Overview and user guide

Date:	22/03/2004
Author	Jon Oldevik
Affiliation:	SINTEF
Contact:	jon.oldevik@sintef.no
Version:	0.8

Table of contents

1. UML MODEL TRANSFORMATION TOOL (UMT) 4

1.1. OVERVIEW 4
1.2. HIGH-LEVEL ARCHITECTURE 4
1.3. UMT TOOL-CHAIN 6
1.3.1. XMI integration 7
1.3.2. XMI Light 7
1.4. TRANSFORMER DEVELOPMENT 9
1.4.1. XSLT transformer 10
1.4.2. Java transformer 12
1.5. UMT USAGE 12
1.5.1. Starting the UMT GUI 12
1.5.2. Working with projects 13
1.5.2.1. Creating a project 13
1.5.2.2. Opening a project 14
1.5.2.3. Deleting a project 15
1.5.2.4. View/edit project properties 15
1.5.3. After opening a project 16
1.5.4. Forward transformation 17
1.5.5. The Result tab 18
1.5.6. Reverse transformation 19
1.5.7. Settings 19
1.5.7.1. Transformations 19
1.5.7.2. Profiles 21
1.5.8. Checking a model's consistency to a profile 22
1.5.8.1. Preferences 23
1.6. COMMAND-LINE TRANSFORMATIONS 23
1.7. ACTIVITY GRAPHS IN UMT 23
1.8. UMT SUMMARY 24

2. APPENDIX (XMI LIGHT SCHEMA) 25

3. APPENDIX (UMT JAVA EMITTER EXAMPLE) 30

Table of figures

FIGURE 1 UMT HIGH-LEVEL USE CASES 4
FIGURE 2 UMT HIGH-LEVEL ARCHITECTURE (PLATFORM-INDEPENDENT) 5
FIGURE 3 UMT HIGH-LEVEL ARCHITECTURE (PLATFORM-SPECIFIC) 6
FIGURE 4 UMT TOOL CHAIN 6
FIGURE 5 XMI LIGHT META MODEL 8
FIGURE 6 UML TICKET EXAMPLE 9
FIGURE 7 NEWLY STARTED UMT 13
FIGURE 8 CREATING A NEW PROJECT 14
FIGURE 9 CREATE NEW PROJECT DIALOG 14
FIGURE 10 PROJECT SELECTOR DIALOG 14
FIGURE 11 PROJECT PROPERTIES 15
FIGURE 12 UMT – WITH LOADED MODEL (INFORMATION VIEW) 16
FIGURE 13 UMT W. LOADED MODEL (XMI LIGHT VIEW) 17
FIGURE 14 TRANSFORMATION MENU 17
FIGURE 15 TRANSFORMATION MENU (2) 18
FIGURE 16 RESULT TAB 18
FIGURE 17 RESULT TAB (SEVERAL FILES) 19
FIGURE 18 REVERSE TRANSFORMATION 19
FIGURE 19 TRANSFORMATION EDITOR 20
FIGURE 20 ADDING A NEW TRANSFORMATION 21
FIGURE 21 ADD TRANSFORMATION DIALOG 21
FIGURE 22 PROFILE EDITOR 22
FIGURE 23 ADDING A PROFILE CONCEPT 22
FIGURE 24 CHANGING A CONCEPT 22
FIGURE 25 UMT ACTIVITY VIEW 24

1. UML Model Transformation Tool (UMT)

UMT (UML Model Transformation Tool) is a prototype developed within the scope of the DAIM, CAFÉ, COMBINE, and ACEGIS projects.

1.1. Overview

UMT is a tool to support model transformation and code generation based on UML models in the form of XMI.

XMI models are imported by the tool and converted to a simpler intermediate format, which is the basis for validation and generation towards different target platforms.

The intermediate format is an XML format, which in UMT is called XMI Light. The end-user of the tool doesn't need be concerned about this format. However, the developers of code generators (emitters) need to know its details. *(In this context, an emitter is a mechanism that generates output from a source model, e.g. code. An emitter is also called a code generator or a transformer.)*

The main usage scenario is to import a UML XMI model and generate code for the desired target platform. Figure 1 shows the high-level use cases that are most pertinent to a developer using UMT.

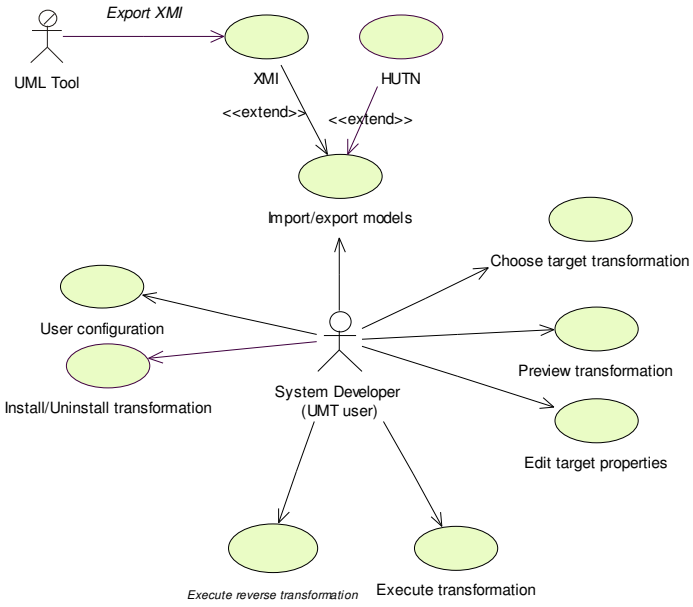


Figure 1 UMT high-level use cases

Figure 1 shows the high-level use cases for a UMT user. The user will typically import models in the form of XMI or XMI Light, which represent UML models from some UML tool. The user will have the choice of choosing target technology, edit some properties for the target technology, preview and then execute the transformation. This will produce output for the target technology. The user will also configure project- or user-specific settings and install/un-install transformations.

1.2. High-level architecture

Figure 2 and Figure 3 illustrates the high-level architecture of UMT. Figure 2 describes the high-level components independent of technology (platform-independent). Figure 2 shows a more technology-near architecture.

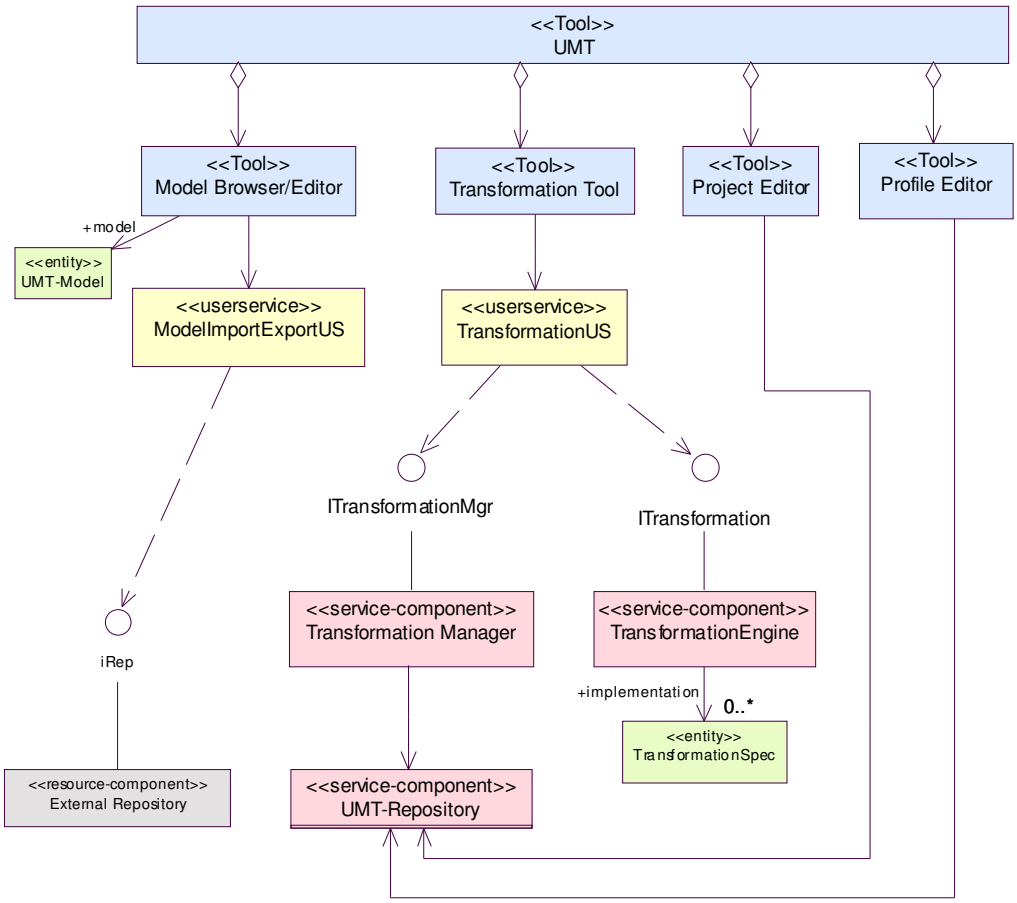


Figure 2 UMT high-level architecture (platform-independent)

The UMT tool consists of sub-tools that provide different end-user functionality. The model browser/editor provides a viewer towards a model and the capability of assigning different kind of properties to model elements, for example technology-specific properties. The project editor provides a context for working with source models. The profile editor provides a means of providing support for the concepts in product line architectures. The transformation tool provides the means for defining and modifying new transformations.

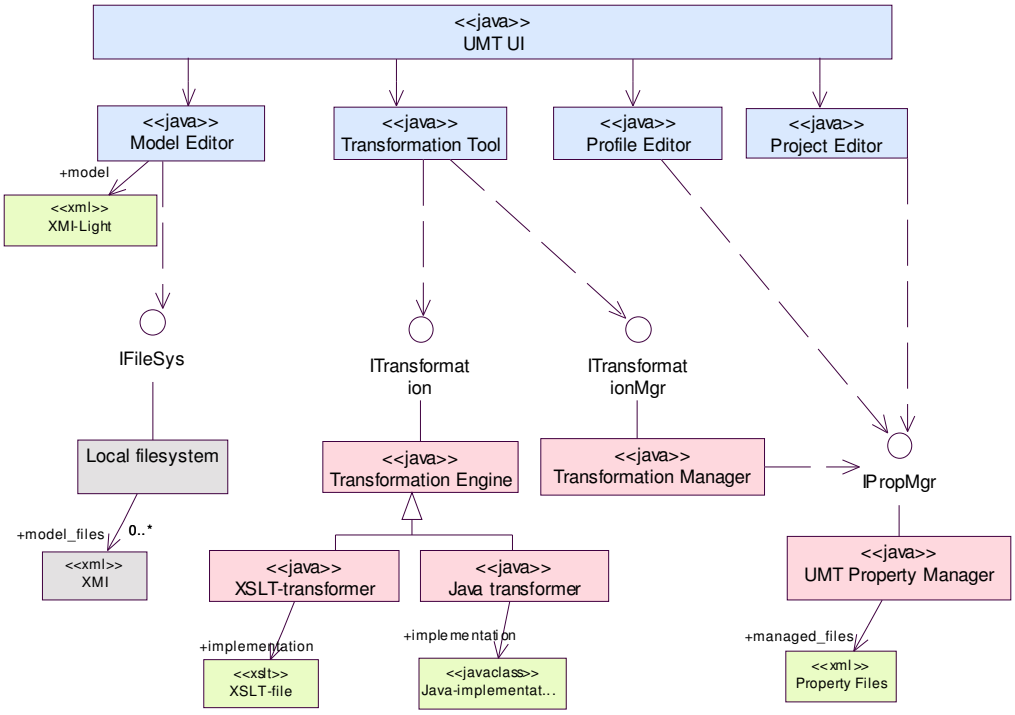


Figure 3 UMT high-level architecture (Platform-specific)

Figure 3 shows a platform-specific mapping of UMT's high-level architecture, where the different components are represented by a platform-specific view of their realisation. The <<java>> stereotype reflects a java implementation; the <<xml>> stereotype reflects an XML file; the <<xslt>> stereotype reflects an xslt file; the <<javaclass>> stereotype reflects a compiled java class.

1.3. UMT Tool-chain

UMT can be used as both a visual tool and as a command-line tool. The architecture of it's code generation component (transformation engine) is based on XSLT. It also supports other kinds of transformer implementation, e.g. java-based. Details of the transformation architecture are described in chapter 1.4.

An overview of the UMT tool-chain, where UMT is integrated with a UML modelling tool, is shown in Figure 4.

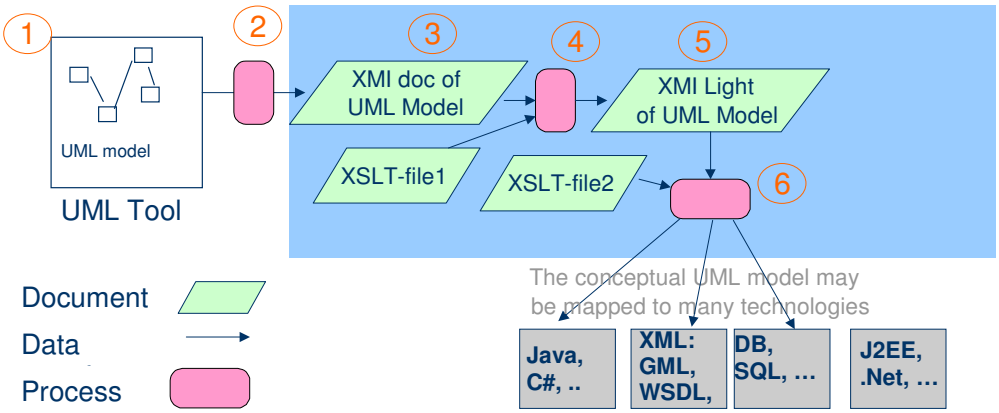


Figure 4 UMT tool chain

The process of using UMT together with a modelling tool is described in a simple way below.

(1) The source for using UMT is a UML model described in some UML tool capable of exporting XMI.

(2) XMI is used as an intermediate format, which is exported from the UML tool and imported by UMT. XMI will provide tool-independence with respect to the UML modelling tool.

(3) The XMI-representation of the model is imported by UMT.

(4) UMT transforms XMI to a simpler format, called XMI Light. This transformation is done in order to simplify later transformations. XMI Light is further described in chapter 1.3.2. UMT uses XSLT to transform XMI to XMI Light.

(5) XMI Light is the internal model representation used by UMT. This is the format that is transformed to different target technologies, like Java, C#, XML-based formats like WDSL, GML, database schemas, application server technologies, like J2EE, .Net.

(6) UMT transforms the XMI Light to desired target technology using a transformer (a transformation implementation). In UMT, this is normally XSLT, but may also be a Java-implementation.

XMI Light is transformed to the end-result, for instance J2EE, WSDL, GML, etc.

1.3.1. XMI integration

As shown in the previous chapter, UMT uses XMI as the UML integration format. The currently supported XMI is XMI version 1.0 for UML version 1.1/1.3 and XMI version 1.1 for UML 1.3 and 1.4.

1.3.2. XMI Light

XMI Light is the internal model used by UMT, i.e. it is an intermediate format used by the tool. It is also, however, a simple lexical view into the UML model.

UMT transforms XMI to the XMI Light format, which is used by the tool for browsing and editing. The UMT transformers use XMI Light as a source metamodel for doing code generation.

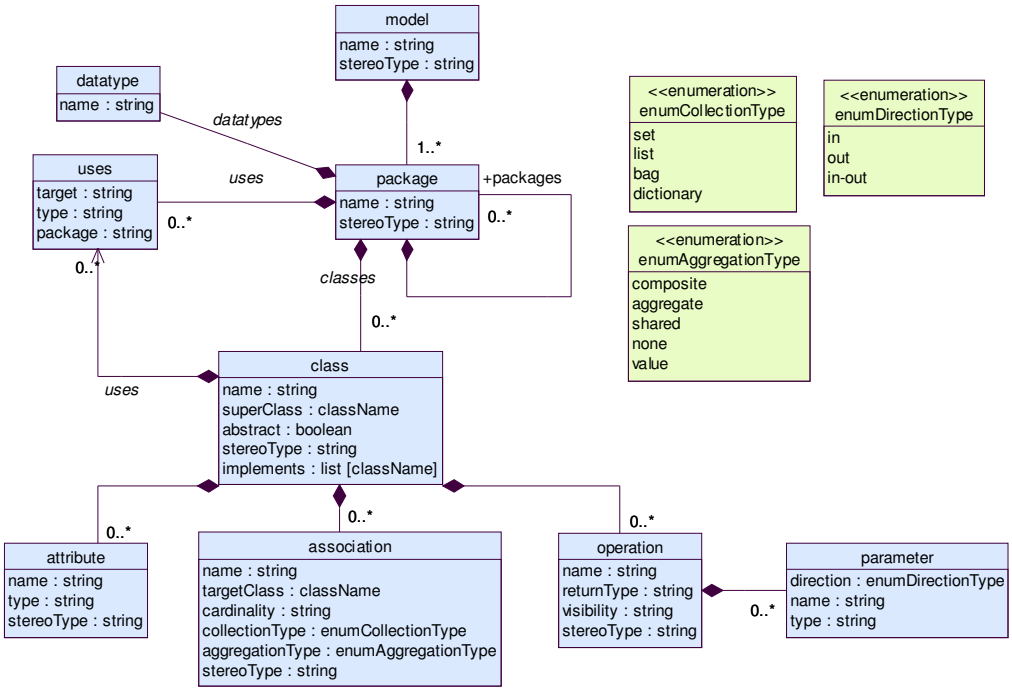


Figure 5 XMI Light meta model

The example below shows how the XML-based representation of the XMI Light looks. The details of the format are specified in an XML schema file (XMI Light.xsd). Figure 6 shows the UML source for the example.

```
<model xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:noNamespaceSchemaLocation="XMI Light.xsd">
   <package id="rootPack_ticket" name="ticket">
      <class abstract="false" id="class_TicketService" name="TicketService"
           stereoType="service">
         <implements id="class_ITicketService" />
         <implements id="class_IAuthorisaton" />
         <attribute cardinality="1..1" name="owner" type="dt_string" />
         <attribute cardinality="1..1" name="version" type="dt_string" />
         <association aggregationType="none" cardinality="0..*" collectionType="set"
            name="reservations" targetClass="class_Reservation" />
         <operation name="buyTicket" returnType="class_Ticket" />
       <operation name="pay">
            <parameter name="amount" type="dt_integer" />
         </operation>
         <taggedValue tag="persistence" value="transient" />
       </class>
      <class abstract="false" id="class_Ticket" name="Ticket" stereoType="entity">
         <attribute cardinality="1..1" name="price" type="dt_integer" />
         <attribute cardinality="1..1" name="issueDate" type="dt_date" />
         <attribute cardinality="1..1" name="ticket_id" type="dt_string" />
         <taggedValue tag="persistence" value="transient" />
       </class>
      <class abstract="false" id="class_Reservation" name="Reservation"
           stereoType="entity">
         <attribute cardinality="1..1" name="reservation_id" type="dt_string" />
         <association aggregationType="none" cardinality="1..*" collectionType="set"
            name="reservation_for" targetClass="class_Ticket" />
         <taggedValue tag="persistence" value="transient" />
       </class>
      <class abstract="false" id="class_ITicketService" name="ITicketService"
           stereoType="interface">
       <operation name="payReservation" returnType="class_Reservation">
            <parameter name="ticket_id" type="dt_string" />
         </operation>
         <operation name="getTickets" returnType="dt_list_Ticket_" />
       <operation name="reserveTicket">
```

```
          <parameter name="ticket_id" type="dt_string" />
        </operation>
        <taggedValue tag="persistence" value="transient" />
      </class>
    <class abstract="false" id="class_IAuthorisaton" name="IAuthorisaton"
         stereoType="interface">
       <operation name="login" returnType="dt_string">
          <parameter name="username" type="dt_string" />
          <parameter name="password" type="dt_string" />
        </operation>
        <taggedValue tag="persistence" value="transient" />
      </class>
      <datatype id="dt_string" name="string" />
      <datatype id="dt_integer" name="integer" />
      <datatype id="dt_date" name="date" />
      <datatype id="dt_list_Ticket_" name="list(Ticket)" />
    <package id="pack_TicketFactory" name="TicketFactory">
       <class abstract="false" id="class_TicketFactory" name="TicketFactory">
          <attribute cardinality="1..1" name="session_id" type="dt_string" />
          <operation name="createTicket" returnType="class_Ticket" />
          <taggedValue tag="persistence" value="transient" />
        </class>
      </package>
    <package id="pack_TicketShop" name="TicketShop">
        <uses target="pack_TicketFactory" />
       <class abstract="false" id="class_TShop" name="TShop">
          <attribute cardinality="1..1" name="shop" type="dt_string" />
          <taggedValue tag="persistence" value="transient" />
        </class>
      </package>
    </package>
  </model>
```

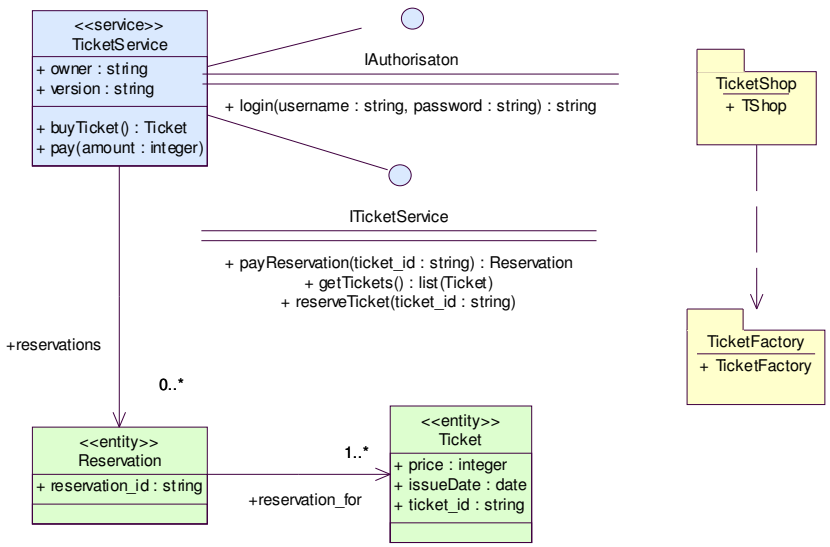


Figure 6 UML Ticket example

Figure 6 shows the UML model representation corresponding to the XMI Light example above.

The XML schema for XML Light is described in appendix 2.

1.4. Transformer development

The transformer architecture in UMT is based on a flexible model where different kind of transformer implementations can be plugged in.

Currently, two main different types of transformers can be plugged in: Java transformers and XSLT-transformers.

1.4.1. XSLT transformer

A UMT XSLT transformer is an XSLT Stylesheet that produces transformations based on XMI Light input models.

Currently, there are two kinds of XSLT transformers; single-file and multi-file.

Single-file XSLT transformers are aimed at producing one single output file from the source model, e.g. a WSDL or SQL file. The single-file transformation process is implemented just by an XSLT file and the result is that produced by that file.

Multi-file XSLT transformers are aimed at producing several output files from the source model, e.g. a set of java files. The multi-file transformation process is implemented by an XSLT file and pre-processed by a UMT Java Transformer class. The XSLT result defines an XML structure of packages and files, which is used to produce physical files. The processing of a multi-file transformation is currently implemented by the UMT class *'transformer.DefaultMultiFileTransformer'*.

Common to both types of transformers is that the XSLT relates to the XMI Light input.

A format pattern of a single-file XSLT:

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="xml" indent="yes"/>

  <xsl:template match="model">
      <xsl:apply-templates select="package"/>
  </xsl:template>

  <xsl:template match="package">
       J
       THIS IS PRODUCED AS OUTPUT FOR THE PACKAGE.
       J
      <xsl:apply-templates/>
  </xsl:template>

<!--
***     template match class
-->
<xsl:template match="class">
      J
      THIS IS PRODUCED AS OUTPUT FOR THE CLASS
      J
      <xsl:apply-templates select="attribute"/>
      <xsl:apply-templates select="association"/>
      <xsl:apply-templates select="operation"/>
</xsl:template>

……
……
```

A format pattern of a multi-file XSLT is shown below. The difference from the single-file patterns is that different file-outputs are separated within XML file tags. The explanation for the tags is described below the example.

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="xml" indent="yes"/>

  <xsl:template match="model">
      <xsl:apply-templates select="package"/>
  </xsl:template>

  <xsl:template match="package">
    <package>
      <xsl:attribute name="name"><xsl:value-of select="@name"/></xsl:attribute>
      <xsl:attribute name="stereotype"><xsl:value-of select="@stereoType"/>
   </xsl:attribute>
```

```
      <xsl:apply-templates select="class"/>
      <xsl:apply-templates select="package"/>
    </package>
  </xsl:template>

<!--
***     template match class
-->
<xsl:template match="class">
  <file>
    <xsl:attribute name="type">javaclass</xsl:attribute>
    <xsl:attribute name="filename"><xsl:value-of select="@name"/>
    </xsl:attribute>
    <xsl:attribute name="location">interfaces/<xsl:value-of select="../@name"/>
    </xsl:attribute>
    <xsl:attribute name="stereotype">interface</xsl:attribute>
    <xsl:attribute name="extension">java</xsl:attribute>

package interfaces;

import java.util.*;

public interface <xsl:value-of select="@name"/> {
      <xsl:apply-templates select="attribute"/>
      <xsl:apply-templates select="association"/>
      <xsl:apply-templates select="operation"/>
}
   </file>
</xsl:template>


<!--
*** template processAttributes
-->

<xsl:template match="attribute">
        public <xsl:call-template name="map2JavaType"><xsl:with-param
   name="orgtype" select="@type"/></xsl:call-template> get<xsl:call-template
   name="firstUpperCase"><xsl:with-param name="input"
   select="@name"/></xsl:call-template>();
        public void set<xsl:call-template name="firstUpperCase"><xsl:with-param
   name="input" select="@name"/></xsl:call-template>(<xsl:call-template
   name="map2JavaType"><xsl:with-param name="orgtype"
   select="@type"/></xsl:call-template><xsl:text>&#32;</xsl:text><xsl:value-of
   select="@name"/>);
</xsl:template>

<!--
****   template processAssociations
-->

<xsl:template match="association">
</xsl:template>

<!--
***   template processOperations
-->

<xsl:template match="operation">
      <xsl:apply-templates select="parameter"/>
</xsl:template>

<!--
***   template processParameter
-->

<xsl:template match="parameter">
 </xsl:template>
</xsl:stylesheet>
```

The multi-file properties are defined for each file by the XML file-element which can have the attributes as shown in the XML extract below.

```
  <file>
    <xsl:attribute name="type">javaclass</xsl:attribute>
```

```
    <xsl:attribute name="filename"><xsl:value-of select="@name"/>
    </xsl:attribute>
    <xsl:attribute name="location">interfaces/<xsl:value-of select="../@name"/>
    </xsl:attribute>
    <xsl:attribute name="stereotype">interface</xsl:attribute>
    <xsl:attribute name="extension">java</xsl:attribute>
     J
    HERE Goes the Content of the file
     J
 </file>
```

– The attribute *'type'* says what kind of target file type this is. Currently, only *type="xml"* or *type="xml-file"* will signify the processing result. A type of xml/xml-file will be attempted processed as an XML-file.

– The attribute *'filename'* signifies the result filename.

– The attribute *'location'* signifies the directory location of the file.

– The attribute *'stereotype'* signifies the stereotype that applies to the file item. This is currently not used.

– The attribute *'extension'* signifies the file extension for the output file.

1.4.2. Java transformer

A UMT Java emitter a Java-class that implements the UMT Transformer interface *'transformer.TransformationEngine'*, which is shown below.

```
public interface TransformerEngine
{
   public void setInputSource (java.io.Reader input);
   public void setTransformationImpl (Object impl);
   public void setOutputDir (String dir);
   public void doTransformation ();
   public void addTransformationResultListener
   (TransformationResultListener listener);
}
```

The simplest way of implementing this interface is to subtype the class *'transformer.AbstractTransformer'*, which implements all but the *'doTransformation'* method.

The default input that is sent in a *'setInputSource'* call is an instance of a model on XMI Light form. The transformation itself must be implemented in the *'doTransformation'* method. An example is given in the 'DefaultJavaTransformer' class shown in the appendix in chapter 3.

In order to add a new transformer (install it) in UMT, the UMT transformation configuration tool can be used. This is described in chapter 1.5.7.

1.5. UMT usage

1.5.1. Starting the UMT GUI

The UMT GUI is started by running the batch-file 'run.bat' located at the root of UMT installation directory. This will start the tool without in start up mode as shown in Figure 7.

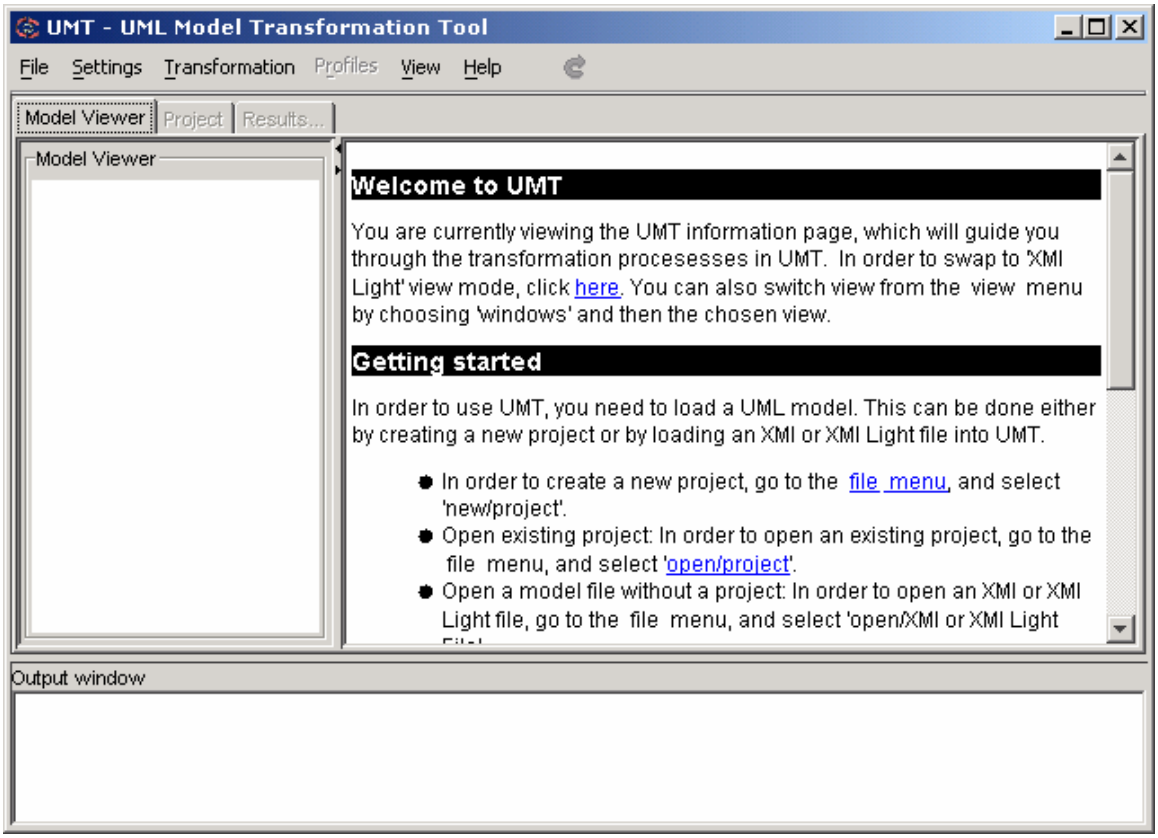


Figure 7 Newly started UMT

When UMT is started, you are ready to import XMI/UML XMI models. The best way of doing so is to associate a project with your model file. It is also possible to open model files without a project context.

The left side view default displays an ‘Information View’, which provides information about available actions in UMT. You can switch to XMI Light model view by choosing ‘View|Windows|Xmi Light view’ from the menu or clicking the hyperlink under ‘remove this view’ in the Information View. To change the default view to a XMI Light view, open the ‘Settings|Preferences’ menu and change the ‘umt.defaultview’ property.

1.5.2. Working with projects

Projects are the best way to work with your XMI models. It gives you a way to associate your model with a project context, which can be managed (e.g. control source generation output directory). When opening a project, the associated XMI/XMI Light file will be loaded.

1.5.2.1. Creating a project

There are two ways of creating a project in UMT, illustrated in Figure 8. From the file menu, you can choose ‘new|project’ or ‘open|XMI/XMI Light as new project’.

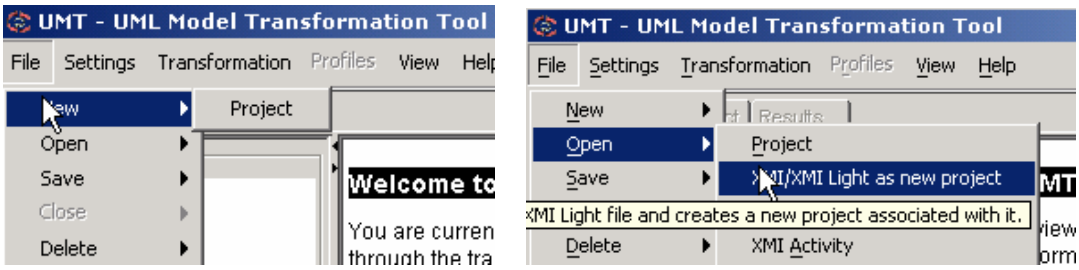


Figure 8 Creating a new project

These two options will both open the dialog shown in Figure 9. The latter will allow you start with finding the file you want to work with. The first will launch an empty create dialog.

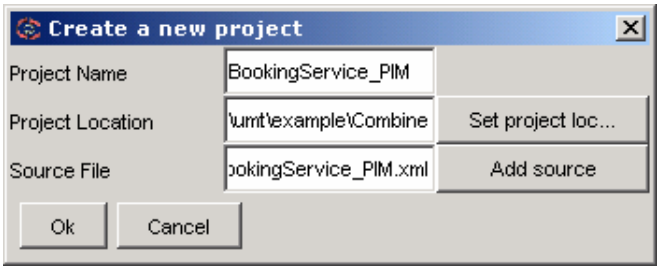


Figure 9 Create new project dialog

When you click the 'Ok' button, a project file will be created in *Project Location*, associated with the model file selected in *Source File*. If the selected project location already contains a project file, you will be prompted to overwrite or not. *(There can be only one project in a directory).*

The model file will be loaded, and you are ready to perform transformations.

1.5.2.2. Opening a project

To open an existing project, choose the 'File|Open|Project' from the menu. This will launch the project selector (Figure 10), which will allow you to select among the projects and open one of them. The selector dialog shows the names of the projects, and their location. (You may modify the value of the location property in this dialog. This property controls where UMT looks for the project file.)

Click the 'Ok' button or press the enter key to open the selected project.

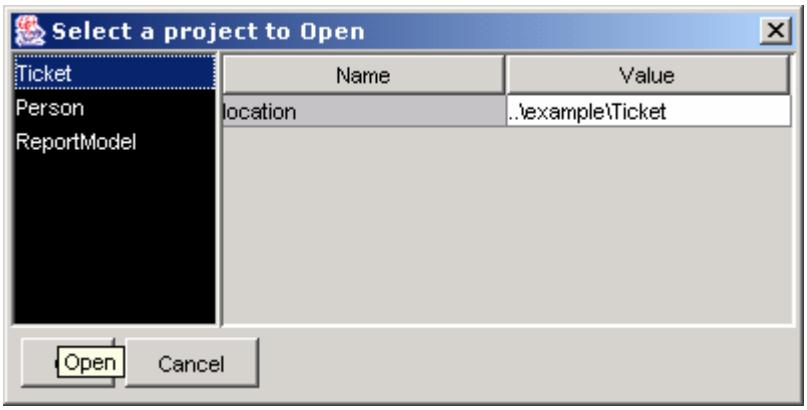


Figure 10 Project selector dialog

After pressing the 'Ok' button, the project's model file will be loaded.

1.5.2.3. Deleting a project

To delete an existing project, choose 'File|Delete|A Project' from the menu. this will launch the project selector to delete a project.

Also, if a project is opened, you can delete it by choosing 'File|Delete|Delete current project' from the menu.

1.5.2.4. View/edit project properties

Associated with each project is a set of project properties, some of which may be modified by the user.

The 'Project' tab gives a view of these properties and a means of changing them (Figure 11).

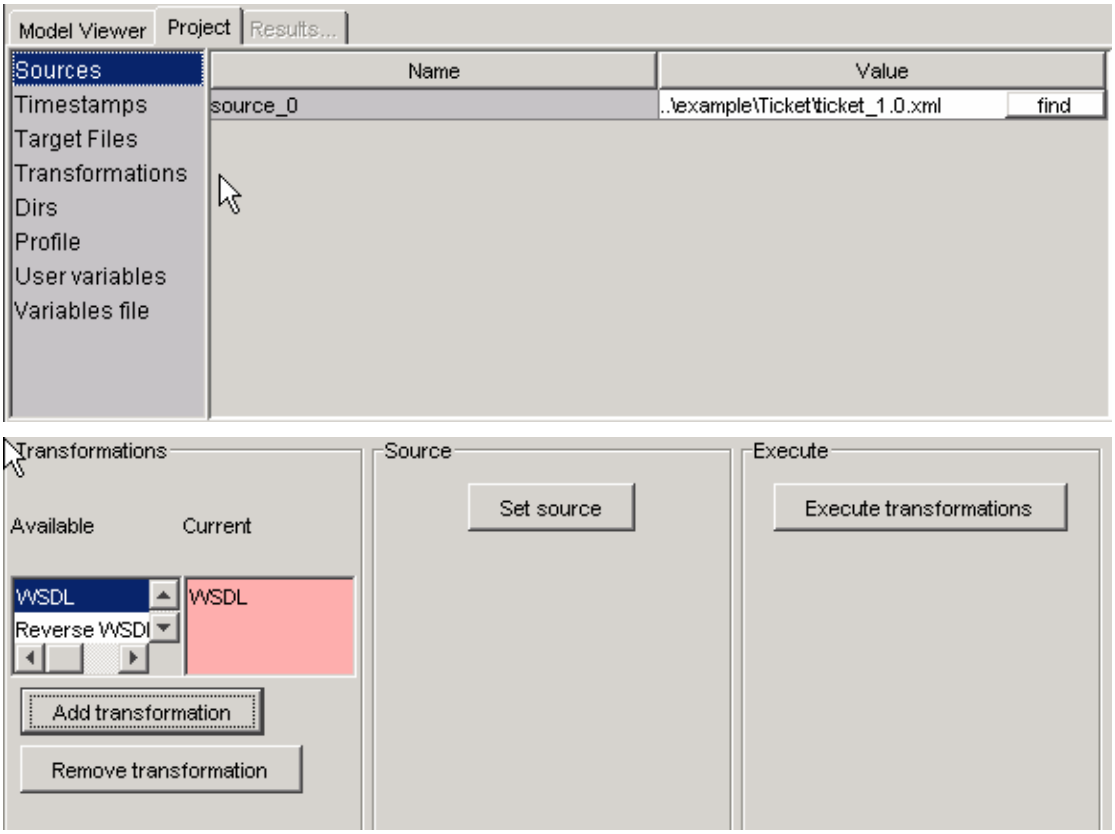


Figure 11 Project properties

The project properties are ordered in groups, which define a set of properties. The following describes the groups and the properties they contain:

Property group	Description	Properties
Sources	Refers to the source file of the project (operational)	*source_0=the source file path*
Timestamps	Different time stamps for the project (informational)	*Created, Last opened, Last saved, Previous transformation (name), Previous transformation time*
Target Files	The name of target files associated with this project (generated in this project) (not used)	
Transformations	The transformations	*The names of the associated*

	associated with this project (currently only informational – to be operational)	*transformations*
Dirs	Directory settings	*gendir=the directory to which source files are generated (for multifile generations)*
Profile	Associated profile information (informational)	*Profile name*
User Variables	User defined variables used for code generation	*PACKAGE_NAME* *PACKAGE_PATH*
Variables file	Output file for user variables	

1.5.3. After opening a project

When a project has been opened, the associated model will be loaded and become visible within the UMT model viewer. You are now ready to perform transformations.

Figure 12 and Figure 13 shows the UMT gui after loading a project/model with the Information view and XMI Light view, respectively.

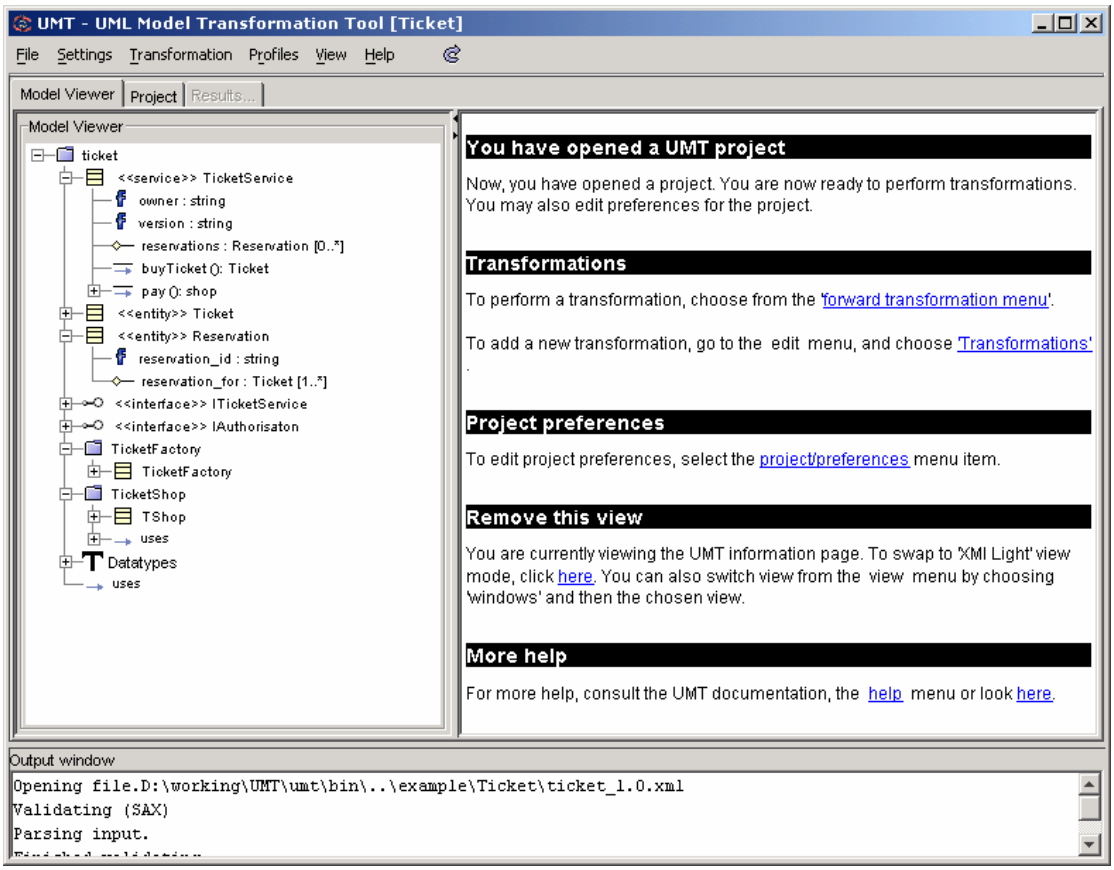


Figure 12 UMT – with loaded model (Information View)

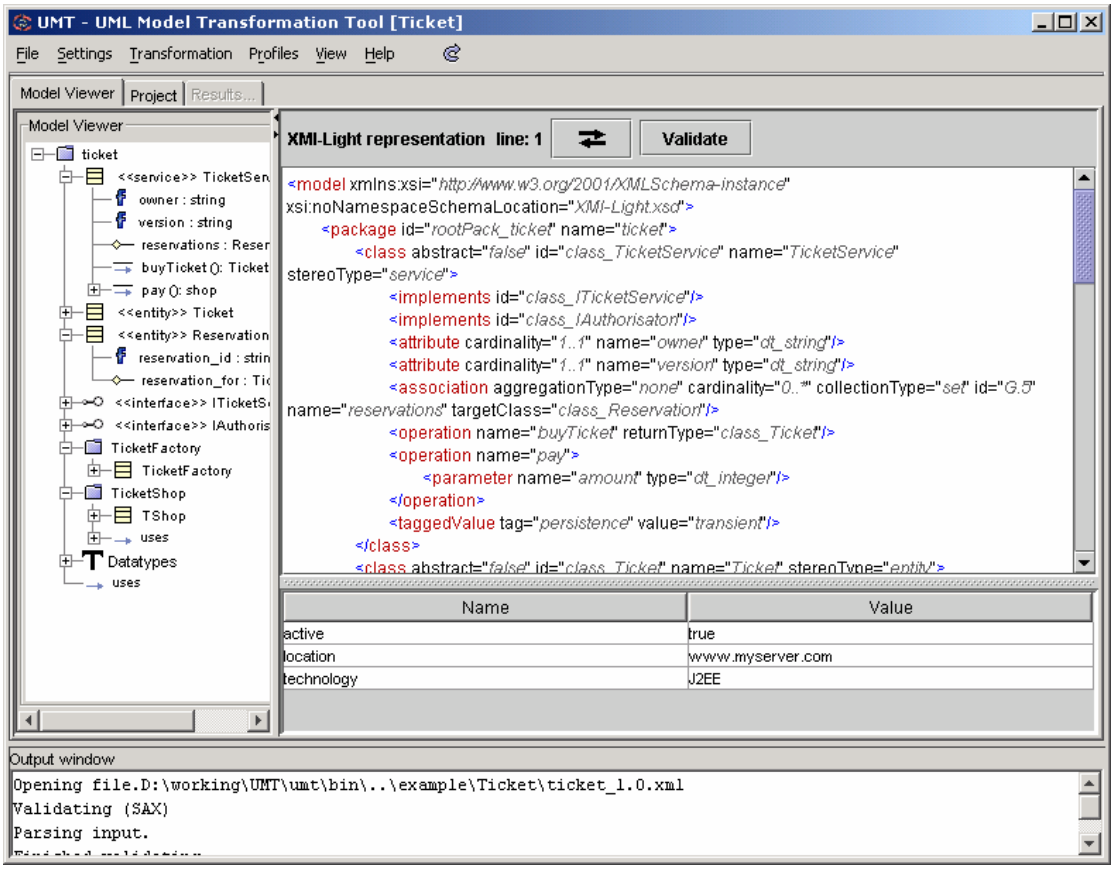


Figure 13 UMT w. loaded model (XMI Light view)

1.5.4. Forward transformation

When a project (or an independent model) has been loaded, you can execute transformations available in the 'Transformations|forward transformation' menu (Figure 14).

You may also mouse right-click on the model tree to launch the transformation menu (Figure 15).

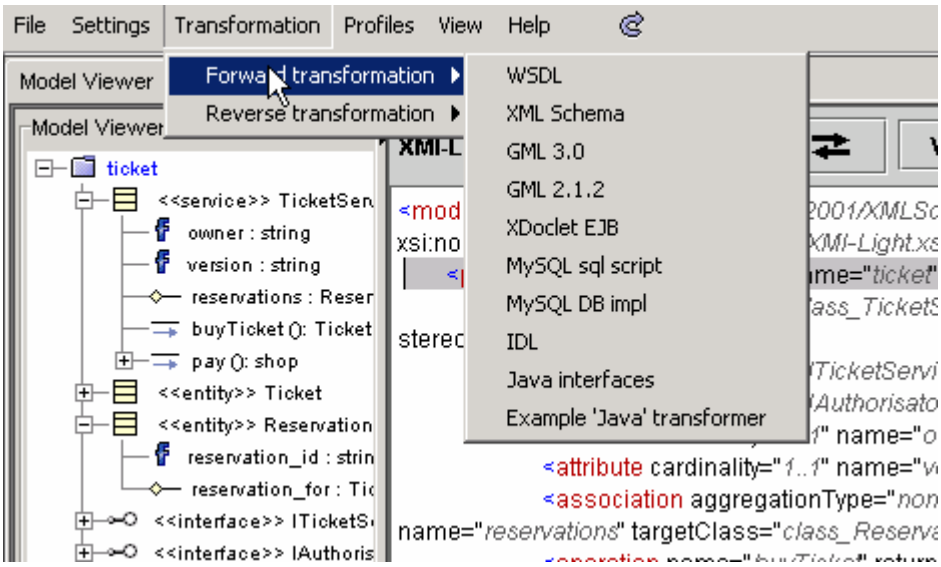


Figure 14 Transformation menu

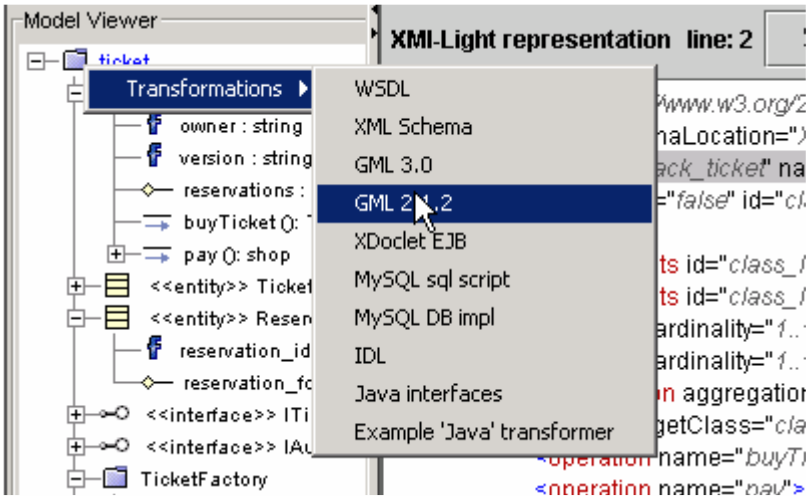


Figure 15 Transformation menu (2)

When choosing a transformation, the transformation engine will execute the associated transformation script/code. The result of a transformation is typically one or more source code files, e.g. one WSDL file or many Java files.

When the result is a single file output, the user will be prompted for a place to save the result. When it is many files, these will be placed relative to the 'gendir' property location.

1.5.5. The Result tab

When the transformation is done, you can view the result in the result tab. It shows the files generated and displays their contents (Figure 16 and Figure 17). If the result is not saved to a file, the result tab will still show the produced result, but there will be no associated file.

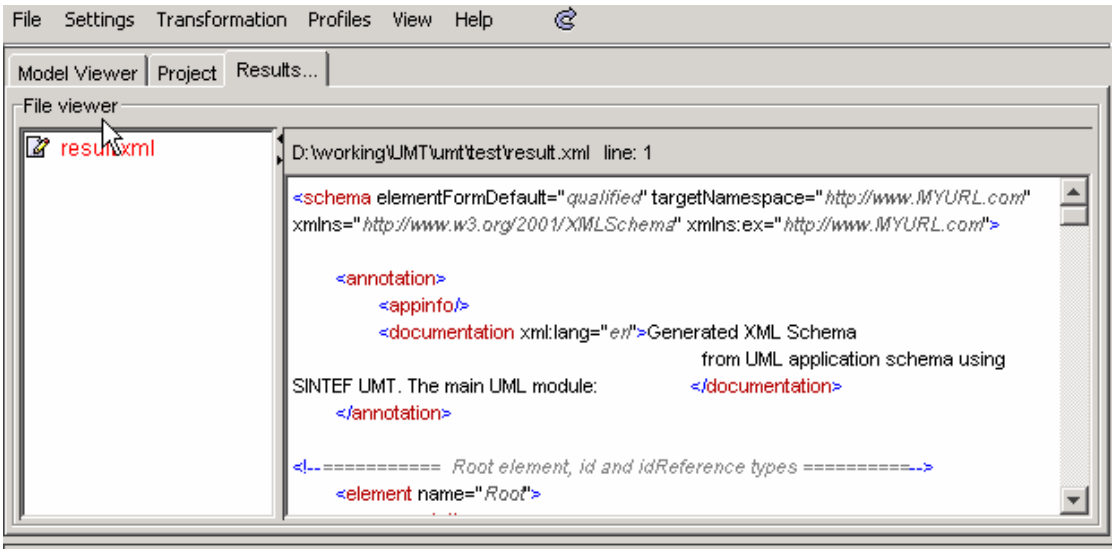


Figure 16 Result tab

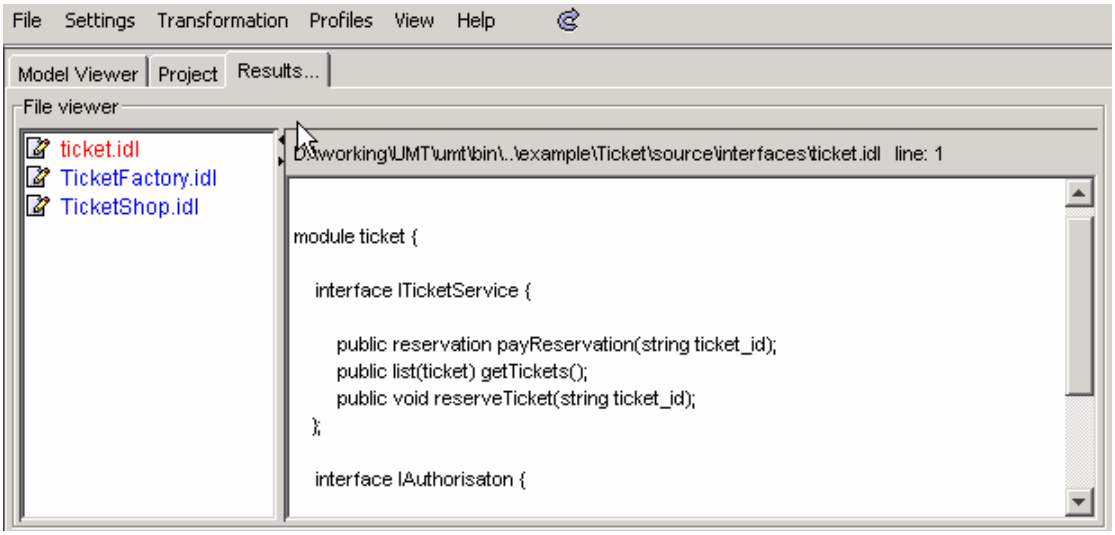


Figure 17 Result tab (several files)

To view a file wihtin the result tab, single click on the file name in the list view. To open the file in an external editor, double click on the file name. This will launch an external viewer/editor (default notepad, but this can be modified in the 'Setttings|Preferences' menu).

1.5.6. Reverse transformation

Reverse transformations are available with or without a project context. A reverse transformation loads a source file, e.g. WSDL and generates a model, which in turn can be used for transformations, or just stored as XMI or XMI Light.

Reverse transformations are defined in the same way as forward transformations, and become available in tool menu 'Transformations|Reverse transformation' (Figure 18).

Figure 18 Reverse transformation

An example of a reverse transformation is from WSDL to XMI Light.

1.5.7. Settings

1.5.7.1. Transformations

Transformations are added dynamically, through the 'Settings|Transformations' menu.

This will launch a transformation editor, which allows to add, remove and modify transformations (Figure 19).

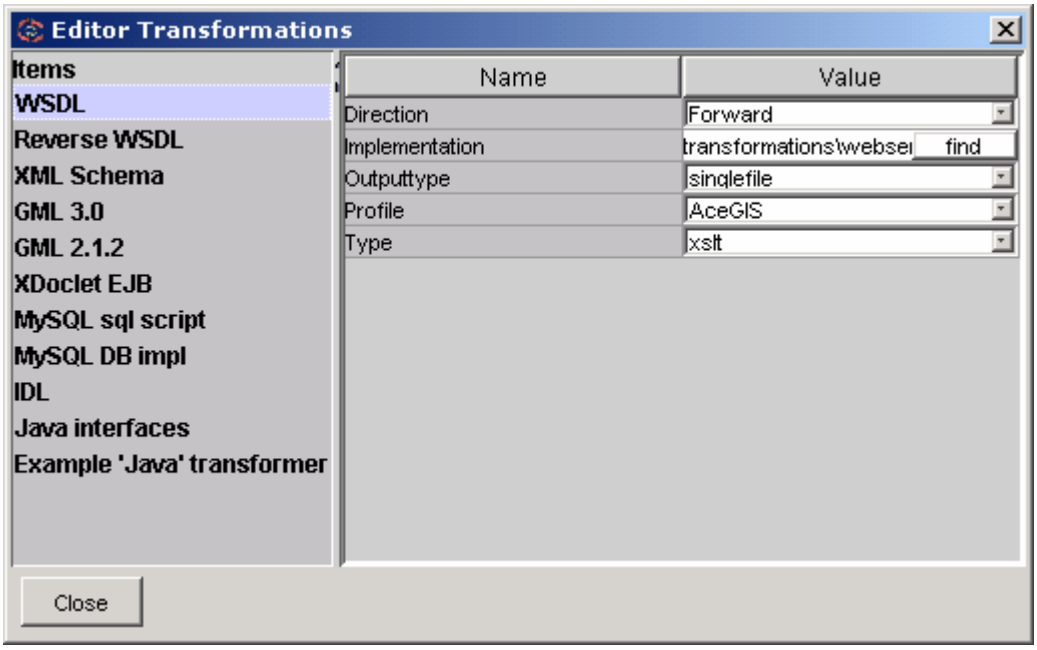


Figure 19 Transformation editor

The transformation editor displays the available transformers and allow to view/modify their properties.

The following table explains the properties of a transformation:

Property	**Values**	**Description**
Direction	Forward | Reverse	The direction of the transformation
Implementation	xslt or java file name	A pointer to the implementation of the emitter.
Outputtype **(used for type xslt)**	singlefile multifile	the output result will be a single file | the output will be several files
Profile	*<name of the associated profile or 'none'>*	Profile associated with the transformation *(not operational)*
Type	xslt java velocity (to be implemented)	xslt: The emitter is implemented by an XSLT file java: The emitter is implemented by a java class. The emitter is a (set of) velocity template(s).

To add a new transformation, launch the create dialog by right-clicking in the list view and choose 'add' (Figure 20).

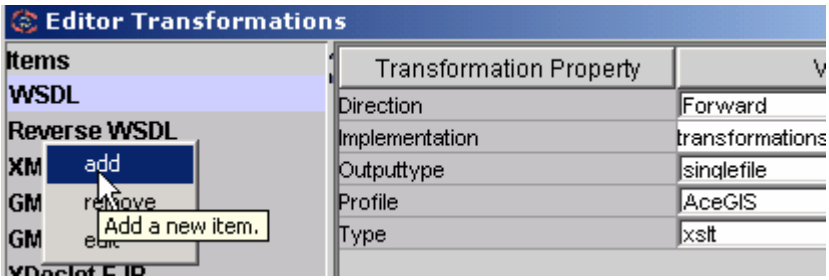


Figure 20 Adding a new transformation

Fill out the information in the dialog (Figure 21) and press 'Ok'. A new transformation is created and available for execution from the transformation menus. If needed, the transformation can be edited from the transformation menu after it is created (e.g. to change the transformer script/class).

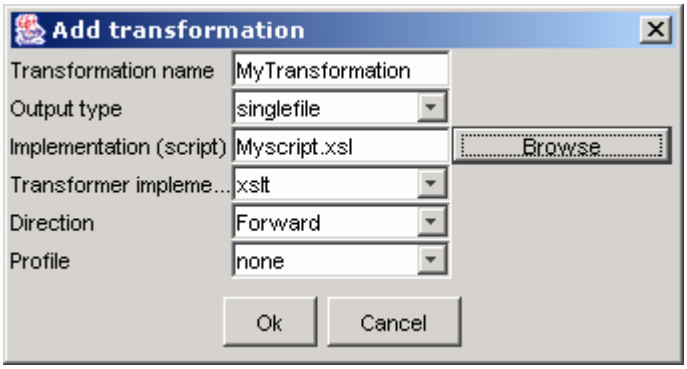


Figure 21 Add transformation dialog

To remove a transformation, use the 'remove' action from the right-click menu. The 'edit' option will allow you to change its name.

1.5.7.2. Profiles

Profiles define model concepts and is comparable to a UML profile. A profile in UMT define modelling concepts that is legal within a specific context/domain. It can then be used to check a model, and also to associate with projects and transformations.

Profiles can be managed in the profile editor, availble via 'Settings|Profiles' menu. The editor is similar to the transformation editor (Figure 22).

New profiles can be added/removed/edited by executing and add/remove/edit action from the right click menu in the list view.

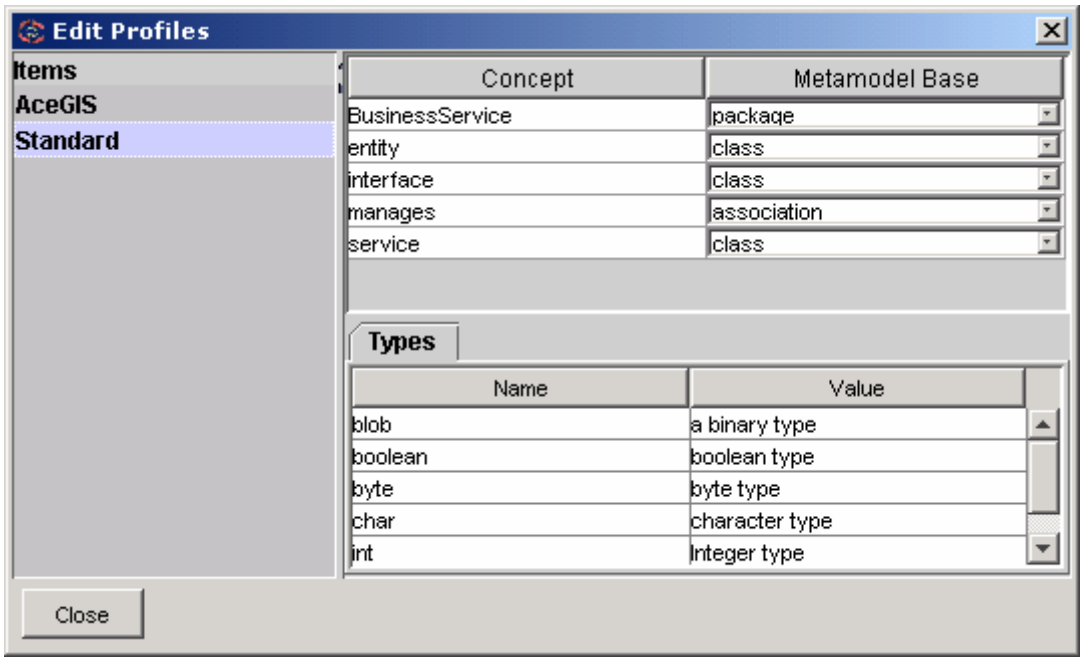


Figure 22 Profile editor

Concepts and types can be added or removed by right-clicking within the property area for the concepts/types, and click the add or remove action (Figure 23).

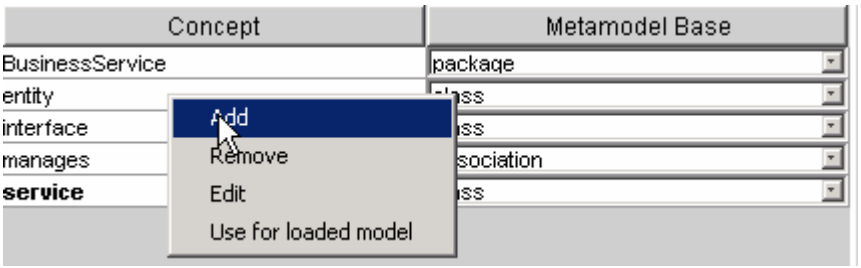


Figure 23 Adding a profile concept

A concept/type can be modified by editing the name directly or (for the concept), modify the meta type it applies to.

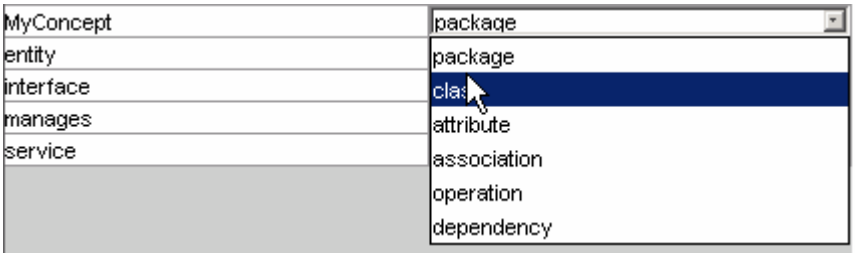


Figure 24 Changing a concept

1.5.8. Checking a model's consistency to a profile

A model can be checked with respect to its consistency with a defined profile. In the example above, a profile can be selected and "used" for the current project/model. It can later be checked by invoking the "Profiles/Check adherence to profile" menu action.

1.5.8.1. Preferences

UMT preferences can be edited through the preference editor, availble thorugh the 'Settings|Preference' menu.

There are currently two preferences that can be modified here:

Preference	Values	Description
umt.defaultview	infoview (defult)|modelview	The view visible in the right-most pane in UMT. This is either an information view or an XMI Light model view.
umt.external.editor	Default value is 'notepad'. Can be changed to anything	Editor launched in the Result tab for viewing results externally.

1.6. Command-line transformations

Command-line transformations is supported through the class umtmain.UMTCmdMain.

It can also be run through via the (windows) batch-file umtcmd (or provide your own batch-file).

The following command-line parameters are allowed/expected to the command-line interface:

umtcmd -xmi|-xmil -i<input_file> -t<transformation> [-o<output_file>][-d<output_dir>] | -l|list

-s = source type xmi or xmi light.
-t = transformation name.
-i = input file.
-o = output directory (optional) (Default is ./gen)
-l = list available transformations.

1.7. Activity graphs in UMT

UML Activity graphs can be read by UMT in the form of XMI for Activity Graphs. XMI 1.1 for UML1.3/UMT1.4 is currently supported.

To open an activity graph, choose on the File-menu, open and then XMI Activity.

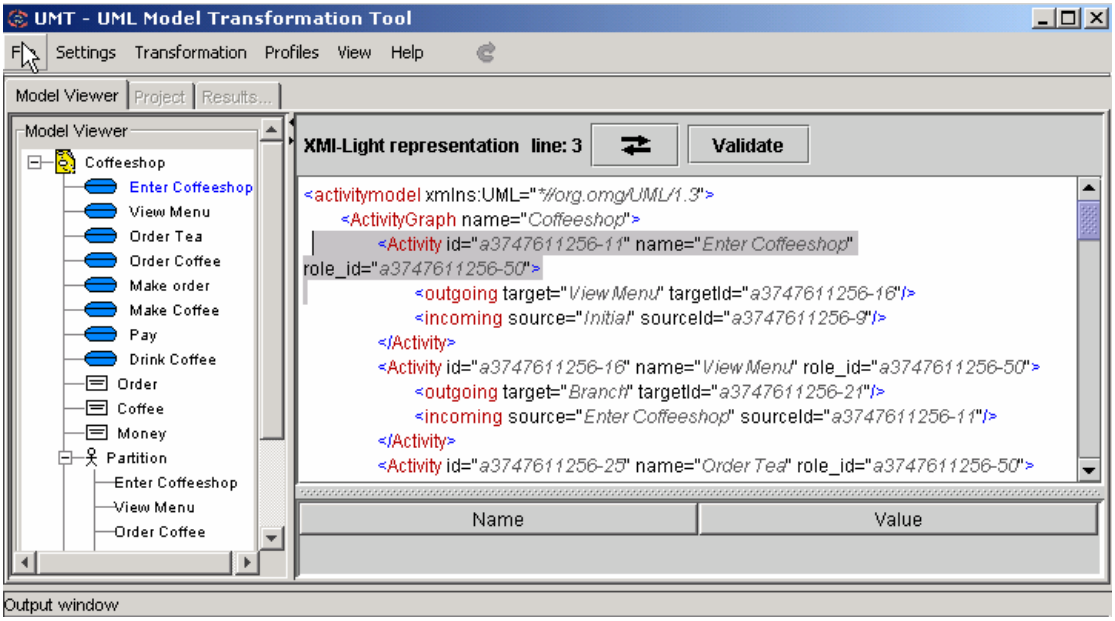


Figure 25 UMT activity view

1.8. UMT summary

Further information regarding UMT and the latest available downloads can be found at http://www.modelbased.net/umt.

2. Appendix (XMI Light schema)

```
- <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="unqualified" attributeFormDefault="unqualified">
  - <xsd:annotation>
      <xsd:documentation>XMI Light/HUTN. This format is inspired by
         Component Modelling Language (CML). It is a Human Understandable
         Textual Notation that describes the aspects of a UML model that
         is essential for code generation. As with XMI for UML models, XMI
         Light describes the UML model info, but in a much simpler manner,
         although not all aspects of UML info is described. XMI Light
         version: 1.1</xsd:documentation>
    </xsd:annotation>
  - <!--
    Main Element
    -->
  - <xsd:element name="model">
    - <xsd:complexType>
      - <xsd:sequence>
          <xsd:element name="package" type="package"
             maxOccurs="unbounded" />
          <xsd:element name="datatype" type="datatype" minOccurs="0"
             maxOccurs="unbounded" />
        </xsd:sequence>
        <xsd:attribute name="name" type="xsd:Name" />
        <xsd:attribute name="id" type="xsd:ID" use="optional" />
        <xsd:attribute name="stereoType" type="enumModelStereoTypeType"
           />
      </xsd:complexType>
    - <!--
      Enforcing constraints
      -->
    - <xsd:key name="classIds">
      - <xsd:annotation>
          <xsd:documentation>List of class ids. To be used by
             xsd:keyref definitions.</xsd:documentation>
        </xsd:annotation>
        <xsd:selector xpath=".//class" />
        <xsd:field xpath="@id" />
      </xsd:key>
    - <xsd:keyref name="superClassExists" refer="classIds">
      - <xsd:annotation>
          <xsd:documentation>Shall ensure that the superClass
             attribute refers to an existing
             class.</xsd:documentation>
        </xsd:annotation>
        <xsd:selector xpath=".//class" />
        <xsd:field xpath="@superClass" />
      </xsd:keyref>
    - <xsd:keyref name="targetClassExists" refer="classIds">
      - <xsd:annotation>
```

```
            <xsd:documentation>Shall ensure that the targetClass
               attribute refers to an existing
               class.</xsd:documentation>
         </xsd:annotation>
         <xsd:selector xpath=".//association" />
         <xsd:field xpath="@targetClass" />
      </xsd:keyref>
   </xsd:element>
- <!--
 Complex types
   -->
- <!--
 package is equivalent to a UML package
   -->
- <xsd:complexType name="package">
   - <xsd:sequence>
         <xsd:element name="uses" type="uses" minOccurs="0"
            maxOccurs="unbounded" />
         <xsd:element name="class" type="class" minOccurs="0"
            maxOccurs="unbounded" />
         <xsd:element name="datatype" type="datatype" minOccurs="0"
            maxOccurs="unbounded" />
         <xsd:element name="package" type="package" minOccurs="0"
            maxOccurs="unbounded" />
         <xsd:element name="taggedValue" type="taggedValue"
            minOccurs="0" maxOccurs="unbounded" />
      </xsd:sequence>
      <xsd:attribute name="name" type="xsd:Name" />
      <xsd:attribute name="id" type="xsd:ID" use="required" />
      <xsd:attribute name="stereoType" type="enumPackageStereoTypeType" />
      <xsd:attribute name="imports" type="xsd:Name" />
   </xsd:complexType>
- <!--
 uses indicates a UML dependency association
   -->
- <xsd:complexType name="uses">
      <xsd:attribute name="target" type="xsd:IDREF" />
   </xsd:complexType>
- <!--
 implements indicates a UML implements association
   -->
- <!--
 @id is a required reference
   -->
      - <!--
       @name is an optional reference provided for
             usability if the ids are not human understandable.
   -->
- <xsd:complexType name="implements">
      <xsd:attribute name="id" type="xsd:IDREF" use="required" />
      <xsd:attribute name="name" type="xsd:string" use="optional" />
   </xsd:complexType>
```

```
- <!--
  class is equivalent to a UML class
        In XMIL this is also used for
              Classes stereotyped <<Interface>> and
              Classes stereotyped <<DataType>> which has
              SUM(attributes, associations,superClass,operations) > 0
-->
- <xsd:complexType name="class">
  - <xsd:sequence>
      <xsd:element name="uses" type="uses" minOccurs="0"
        maxOccurs="unbounded" />
      <xsd:element name="implements" type="implements" minOccurs="0"
        maxOccurs="unbounded" />
      <xsd:element name="attribute" type="attribute" minOccurs="0"
        maxOccurs="unbounded" />
      <xsd:element name="association" type="association"
        minOccurs="0" maxOccurs="unbounded" />
      <xsd:element name="operation" type="operation" minOccurs="0"
        maxOccurs="unbounded" />
      <xsd:element name="taggedValue" type="taggedValue"
        minOccurs="0" maxOccurs="unbounded" />
    </xsd:sequence>
    <xsd:attribute name="name" type="xsd:Name" />
    <xsd:attribute name="id" type="xsd:ID" use="required" />
    <xsd:attribute name="superClass" type="xsd:IDREF" />
    <xsd:attribute name="abstract" type="xsd:boolean" />
    <xsd:attribute name="stereoType" type="enumClassStereoTypeType" />
  </xsd:complexType>
- <xsd:complexType name="datatype">
    <xsd:attribute name="name" />
    <xsd:attribute name="id" type="xsd:ID" use="required" />
  </xsd:complexType>
- <!--
 attribute is equivalent to a UML attribute
-->
- <xsd:complexType name="attribute">
  - <xsd:sequence>
      <xsd:element maxOccurs="unbounded" minOccurs="0"
        name="taggedValue" type="taggedValue" />
    </xsd:sequence>
    <xsd:attribute name="name" type="xsd:Name" use="required" />
    <xsd:attribute name="id" type="xsd:ID" use="optional" />
    <xsd:attribute name="type" type="xsd:IDREF" use="required" />
  - <!--
    Used to be hutnType
    -->
    <xsd:attribute name="cardinality" type="xsd:string" />
    <xsd:attribute name="stereoType" type="enumAttributeStereoTypeType"
      use="optional" />
  </xsd:complexType>
- <!--
 association is equivalent to a UML association
-->
```

```
- <xsd:complexType name="association">
    <xsd:attribute name="name" type="xsd:Name" />
    <xsd:attribute name="id" type="xsd:ID" use="optional" />
    <xsd:attribute name="targetClass" type="xsd:IDREF" />
    <xsd:attribute name="cardinality" type="xsd:string" />
    <xsd:attribute name="collectionType" type="enumCollectionType" />
    <xsd:attribute name="aggregationType" type="enumAggregationType" />
    <xsd:attribute name="stereoType"
       type="enumAssociationStereoTypeType" />
    <xsd:attribute name="reverse" type="xsd:string" />
  - <!--
   Ordering
    -->
  </xsd:complexType>
- <!--
 operation is equivalent to a UML operation
  -->
- <xsd:complexType name="operation">
  - <xsd:sequence>
    - <xsd:element name="parameter" minOccurs="0"
        maxOccurs="unbounded">
      - <xsd:complexType>
          <xsd:attribute name="direction"
             type="enumDirectionType" />
          <xsd:attribute name="name" type="xsd:Name" />
          <xsd:attribute name="type" type="xsd:IDREF" />
        </xsd:complexType>
      </xsd:element>
      <xsd:element maxOccurs="unbounded" minOccurs="0"
         name="taggedValue" type="taggedValue" />
    </xsd:sequence>
    <xsd:attribute name="name" type="xsd:string" />
    <xsd:attribute name="id" type="xsd:ID" use="optional" />
    <xsd:attribute name="returnType" type="xsd:IDREF" />
    <xsd:attribute name="stereoType" type="enumOperationStereoTypeType"
       />
    <xsd:attribute name="visibility" type="xsd:string" />
  </xsd:complexType>
- <!--
 Tagged values for the UML extension mechanism name=value
  -->
- <xsd:complexType name="taggedValue">
    <xsd:attribute name="tag" type="xsd:string" />
    <xsd:attribute name="value" type="xsd:string" />
  </xsd:complexType>
- <!--
 Simple types
  -->
- <xsd:simpleType name="enumAggregationType">
  - <xsd:restriction base="xsd:string">
```

```
      - <!--
       UML composition
        -->
        <xsd:enumeration value="composite" />
      - <!--
       UML aggregation
        -->
        <xsd:enumeration value="aggregate" />
      - <!--
       UML association
        -->
        <xsd:enumeration value="none" />
      </xsd:restriction>
  </xsd:simpleType>
- <xsd:simpleType name="enumCollectionType">
   - <xsd:restriction base="xsd:string">
        <xsd:enumeration value="set" />
        <xsd:enumeration value="list" />
        <xsd:enumeration value="bag" />
        <xsd:enumeration value="dictionary" />
      </xsd:restriction>
  </xsd:simpleType>
- <xsd:simpleType name="enumDirectionType">
   - <xsd:restriction base="xsd:string">
        <xsd:enumeration value="in" />
        <xsd:enumeration value="out" />
        <xsd:enumeration value="inout" />
      </xsd:restriction>
  </xsd:simpleType>
- <xsd:simpleType name="enumModelStereoTypeType">
     <xsd:restriction base="xsd:string" />
  </xsd:simpleType>
- <xsd:simpleType name="enumPackageStereoTypeType">
     <xsd:restriction base="xsd:string" />
  </xsd:simpleType>
- <xsd:simpleType name="enumClassStereoTypeType">
     <xsd:restriction base="xsd:string" />
  </xsd:simpleType>
- <xsd:simpleType name="enumAssociationStereoTypeType">
     <xsd:restriction base="xsd:string" />
  </xsd:simpleType>
- <xsd:simpleType name="enumOperationStereoTypeType">
     <xsd:restriction base="xsd:string" />
  </xsd:simpleType>
- <xsd:simpleType name="enumAttributeStereoTypeType">
     <xsd:restriction base="xsd:string" />
  </xsd:simpleType>
```

```
- <xsd:simpleType name="hutnType">
    <xsd:restriction base="xsd:string" />
  </xsd:simpleType>
</xsd:schema>
```

3. Appendix (UMT Java Emitter example)

The example given below shows an UMT Java Emitter class that parses the source (XMI Light) model using an XML DOM parser and then outputs some dummy Java classes to standard IO.

```
package transformer;

/**
 * @version          1.0
 *
 * @description
 *
 * @author     Jon Oldevik, (jon.oldevik@sintef.no)
 *
 * @copyright (c) SINTEF 2002 (www.sintef.no)
 *
 */

import javax.swing.*;
import org.w3c.dom.*;

import umtmain.*;
import java.io.*;


public class DefaultJavaTransformer extends AbstractTransformer
   implements TransformerEngine, OutputListener
{
   private String _transformationimpl;
   private static InputHandlerFactory _handlerfactory;

   public DefaultJavaTransformer () {
      _handlerfactory = new InputHandlerFactory ();
   }

   public void setTransformationImpl (Object impl) {
      try {
           _transformationimpl = (String)impl;
      } catch (Exception ex) {
      }
   }

   /**
    *
    * doTransformation does all the transformation work.
    * Assumes Hutn XML input
    *
    */


   public void doTransformation () {
      try {
           System.out.println("DefaultJavaTransformer::doTransformation
   ()");
           XMLUtility xmlutil = new XMLUtility ();
           java.util.Collection c = xmlutil.readXML(_inputsource);

           java.util.Iterator it = c.iterator();
           if (it.hasNext()) {
```

```
                Element root = (Element)c.iterator().next();
                transformDocument (root);
            } else {
                System.out.println ("No model to transform.");
            }

      } catch (Exception ex) {
            addLine ("Failed to perform transformation" +
   ex.getMessage());
      }
   }

   protected void transformDocument (Element element)
   {
      try {
//          System.out.println ("transformDocument " +
   element.getNodeName());
            _handlerfactory.createInputHandler(element).processContent();
      } catch (Exception ex)
      {
            ex.printStackTrace();
      }
   }


   protected interface InputHandler
   {
      public void processContent ();
   }

   protected class InputHandlerFactory
   {
      public InputHandlerFactory () {
      }

      public InputHandler createInputHandler (Element elem) {
//          System.out.println("createInputHandler : " +
   elem.getNodeName());
            String name = elem.getTagName();
            InputHandler handler = null;
            if (name.equalsIgnoreCase("model")) {
                return new ModelInputHandler (elem);
            } else if (name.equalsIgnoreCase("package")) {
                return new PackageInputHandler (elem);
            }else if (name.equalsIgnoreCase ("class")) {
                return new ClassInputHandler (elem);
            }else if (name.equalsIgnoreCase ("attribute")) {
            }else if (name.equalsIgnoreCase ("association")) {
            }else if (name.equalsIgnoreCase ("operation")) {
            }else
                return new AbstractInputHandler(elem);
            return handler;
      }
   }


   protected class AbstractInputHandler implements InputHandler
   {
      protected Element _element;
      public AbstractInputHandler (Element elem) {
            _element = elem;
      }

      public void processContent () {
            // NOOP
      }

   }
```

```
protected class ModelInputHandler extends AbstractInputHandler
{
   public ModelInputHandler (Element element) {
        super (element);
   }

   public void processContent () {
        try {
            NodeList list = _element.getChildNodes();
            for (int i = 0; i < list.getLength(); i++) {
                  if (list.item(i).getNodeType() ==
Element.ELEMENT_NODE)

   _handlerfactory.createInputHandler((Element)list.item(i)).processC
ontent();
            }
        } catch (Exception ex) {
            ex.printStackTrace ();
        }
   }
}

protected class PackageInputHandler extends AbstractInputHandler
{
   public PackageInputHandler (Element element) {
        super (element);
   }

   public void processContent () {
        try {

            /*
             * Create directory
             */

            String stereotype = _element.getAttribute("stereoType");

            /*
             * test stereotype
             */
            addLine("package " + _element.getAttribute("name") + "
{");

            NodeList list = _element.getChildNodes();
            for (int i = 0; i < list.getLength(); i++) {
                  if (list.item(i).getNodeType() ==
Element.ELEMENT_NODE)

   _handlerfactory.createInputHandler((Element)list.item(i)).processC
ontent();
            }

            addLine("} // end of package " +
_element.getAttribute("name"));
        } catch (Exception ex) {
            ex.printStackTrace ();
        }
   }
}

protected class ClassInputHandler extends AbstractInputHandler
{
   public ClassInputHandler (Element element) {
        super (element);
   }
   public void processContent () {
        try {
                  /*
```

```
                        * Create file(s)
                        */

                       String stereotype = _element.getAttribute
   ("stereotype");

                       addLine("class " + "/* " + stereotype + " */ " +
   _element.getAttribute("name"));
                       addLine("{");
                       NodeList list = _element.getElementsByTagName("");
                       for (int i = 0; i < list.getLength(); i++) {

                              NodeList attributes =
   _element.getElementsByTagName("attribute");
                              NodeList associations =
   _element.getElementsByTagName("association");
                              NodeList operations =
   _element.getElementsByTagName("association");
                       }
                       addLine("}");
               } catch (Exception ex) {
                   ex.printStackTrace ();
               }
      }
   }



   public void addLine (String line) {
      System.out.println(line);
   }

}
```