

# MOMOCS

## Model driven Modernisation of Complex Systems

### DELIVERABLE # D53 PROCESS MODERNIZATION TOOL (XSM TRANSFORMATION TOOL)

---

Dissemination Level:	Public
Work package:	WP5
Lead Participant:	TXT e-solutions
Contractual Delivery Date:	M17
Document status:	Final
Preparation Date:	30 November, 2007
Document Version:	1.0

## Revisions

Document Version	Date	Author (Partner)	Description of Changes
0.1	7 December 2007	Alessandra Bagnato (TXT) and Marco Serina (TXT)	TOC
0.2	15 January 2007	Marco Serina (TXT)	Use case description and minor changes
0.3	16 January 2008	Marco Serina (TXT)	Usage manual and minor changes
0.4	17 January 2008	Marco Serina (TXT)	Cheat sheet and tutorial section, open issues and minor changes
0.5	18 January 2008	Marco Serina (TXT)	Annexes and minor changes
0.6	21 January 2008	Marco Serina (TXT)	Minor changes
0.7	14 February 2008	Alessandra Bagnato(TXT)	Introduction to new software deliverable structure, title update to harmonize D5* as agreed in the Milan Meeting, Minor changes
0.8	19 February 2008	Marco Serina (TXT)	Restructuring sections and contents
1.0	20 February 2008	Marco Serina (TXT), Alessandra Bagnato (TXT)	Final version

<b>1</b>	<b>SCOPE .....</b>	<b>7</b>
1.1	PURPOSE OF THE DOCUMENT .....	7
1.2	DOCUMENT STRUCTURE.....	8
<b>2</b>	<b>XSM TRANSFORMATION TOOL INTRODUCTION .....</b>	<b>9</b>
<b>3</b>	<b>TRANSFORMATION TOOL GETTING STARTED .....</b>	<b>10</b>
3.1	REQUIREMENTS.....	10
3.2	INSTALLATION GUIDE .....	11
3.2.1	<i>Eclipse and EMF installation.....</i>	12
3.2.2	<i>Installing ATL.....</i>	19
3.2.3	<i>Installing XSM plug-ins.....</i>	23
3.2.4	<i>Installing XSM Transformation Tool plug-ins .....</i>	24
3.3	GETTING STARTED .....	24
<b>4</b>	<b>TRANSFORMATION TOOL USER'S GUIDE.....</b>	<b>31</b>
4.1	SETTING UP A NEW PROJECT .....	33
4.2	INITIALIZING A TRANSFORMATION .....	35
4.3	EDITING.....	38
4.3.1	<i>Adding ATL expression .....</i>	39
4.3.2	<i>Tree structured view for ATL operations .....</i>	44
4.4	SAVING.....	50
4.5	COMMENTING A TRANSFORMATION .....	51
4.6	RUNNING.....	52
4.7	ATLFORXIRUP LIBRARY .....	55
4.7.1	<i>Xirup2Xirup .....</i>	56
4.7.2	<i>UML2Xirup and Xirup2UML.....</i>	63
4.8	HELP FACILITIES GUIDELINES .....	67
<b>5</b>	<b>TROUBLESHOOTING.....</b>	<b>69</b>
5.1	KNOWN ISSUES .....	69
5.1.1	<i>EJavaClass type error.....</i>	69
5.2	BUG REPORTING.....	73
5.3	FAQ.....	73
<b>6</b>	<b>APPENDIXES .....</b>	<b>75</b>
6.1	ONGOING FEATURES.....	75
6.2	ATL EXECUTION MODES.....	75
6.3	XIRUPCOPY TRANSFORMATIONS .....	77
6.3.1	<i>XirupCopy2004.....</i>	78
6.3.2	<i>XirupCopy2006.....</i>	78
6.3.3	<i>Work case example.....</i>	78
<b>7</b>	<b>ANNEXES.....</b>	<b>78</b>
7.1	ACRONYMS AND GLOSSARY.....	78
7.2	REFERENCE DOCUMENTS .....	78

## INDEX OF TABLES

Table 1: Resources created after using XirupCopy2004 wizard .....	61
Table 2: Resources created after using UML2Xirup wizard .....	66

## INDEX OF FIGURES

Figure 1: XSM Transformation Tool stack .....	11
Figure 2: Update Manager location .....	13
Figure 3: New features to install .....	14
Figure 4: Adding EMF remote site.....	15
Figure 5: Select EMF SDK feature .....	16
Figure 6: User's acceptance .....	17
Figure 7: Final sum-up before installation .....	18
Figure 8: Restart Eclipse .....	18
Figure 9: ATL installation.....	19
Figure 10: Select ATL features .....	21
Figure 11: Select ATL required plug-ins.....	22
Figure 12: Adding UML2 tools.....	23
Figure 13: Select proper perspective .....	25
Figure 14: TT workbench.....	26
Figure 15: Selecting the cheat-sheet (1) .....	28
Figure 16: Selecting the cheat-sheet(2) .....	29
Figure 17: TT cheat-sheet.....	30
Figure 18: Selecting wizard to create a new ATL project.....	33
Figure 19: Setting up the name.....	34
Figure 20: Project in the Navigator view .....	34
Figure 21: New ATL File wizard .....	35
Figure 22: ATL File creation.....	36
Figure 23: Transformation header .....	37
Figure 24: Common editing activity.....	39
Figure 25: ATLEditing menu .....	41
Figure 26 - Insert Called rule pattern .....	42
Figure 27: Insert matched rule pattern.....	43
Figure 28: Customizing pattern .....	44
Figure 29: ATL operations Tree viewer (1) .....	46
Figure 30: ATL operations Tree viewer (2) .....	47
Figure 31: Browse operations and select.....	48
Figure 32: Double-click on operation.....	49
Figure 33: Operation customized by the user.....	49
Figure 34: Dumping ASM.....	50
Figure 35: Errors occurred.....	50

Figure 36: Accessing to file properties.....	51
Figure 37: Commenting transformation file.....	52
Figure 38: Run dialog.....	53
Figure 39: Loading parameters.....	54
Figure 40: load UML2Xirup transformation.....	55
Figure 41: Get access to XirupCopy wizards (1) .....	58
Figure 42: Get access to XirupCopy wizards (2) .....	58
Figure 43: XirupCopy 2004 (for ATL compiler 2004) wizard .....	59
Figure 44: Navigator view.....	60
Figure 45: Launch configuration stub .....	62
Figure 46: UML2Xirup wizard .....	64
Figure 47: Navigator view.....	65
Figure 48: From UML to Xirup and back again.....	66
Figure 49: Xirup transformation help contents .....	67
Figure 50: TagType element properties .....	70
Figure 51: Tag element properties.....	71
Figure 52: XirupCopy2004 (1).....	78
Figure 53: XirupCopy2004 (2).....	78
Figure 54: XirupCopy2006 (1).....	78
Figure 55: XirupCopy2006 (2).....	78
Figure 56: Old XirupCopy2004 .....	78
Figure 57: New XirupCopy2004.....	78
Figure 58: Another example of porting XirupCopy transformation.....	78

# 1 Scope

## 1.1 Purpose of the Document

This deliverable, but also D5.1 and D5.2, does not fully comply with the organization described in the Description of work, where we were thinking of slicing our tools according to the distinction among data, \*ware, and process. While designing our tools, we understood that that division was unnatural, and would have led to conceivable overlapping among the different tools. Our tools are functionality-specific, but they can easily work on the artefacts at the different levels of the hierarchy identified by the MOMOCS Description of Work.

This is why we opted for a different partitioning based on supplied features, instead of the abstraction level. In other words, we prefer to get rid of the horizontal layers originally proposed, and identify consistent and self-contained vertical segments that easily spread across data, \*ware, and process.

This document, in particular, describes how to install and use the XSM Transformation Tool whose aim is to help the modernization process proposed by [2007e] by means of specific transformation rules.

## 1.2 Document Structure

The document is structured as follows:

- Section 2, “Xsm Transformation Tool introduction” briefly explains the aim of XSM Transformation Tool
- Section 3, “Transformation Tool getting started” is for installing instructions and a first use of the tool
- Section 4, “Transformation Tool User’s Guide” depicts how to use the tool and which the core functionalities are
- Section 5, “Troubleshooting”, mainly presents known issues and a FAQ section
- Section 6, “Appendixes”, regards ongoing features and useful information that better detail some concepts described in the deliverable
- Section 7, “Annexes” contains glossary and references



## 2 XSM Transformation Tool Introduction

The **XSM Transformation tool** is in charge of aiding the evolution of a TBMS model into a MS model performed by following the Xirup methodology.

It generally takes as an input:

- A XIRUP System Model (XSM) describing the To Be Modernised System (TBMS) or a partial modernized version of it
- the XIRUP Metamodel, namely the reference metamodel [2007f]

The output of the Transformation Tool will be:

- a set of transformations applicable to the input XSM
- An XSM which can be a fully or partially “Modernized System” (MS) model, according to the complexity of the modernization process and the way it is faced.

The XSM Transformation tool (from here on called simply “TT”) is based upon the language and functionalities provided by the Eclipse M2M ATL project [ATL].

Its main goal is encapsulating and improving those functionalities in order to transform TBMS models into suitable MS models which are compliant to user’s modernization requirements.

For more references and documentation, please also refer to [ATL\_DOC].

## 3 Transformation Tool getting started

This section aims at providing a brief introduction of TT and a step-by-step installation guide for all tools.

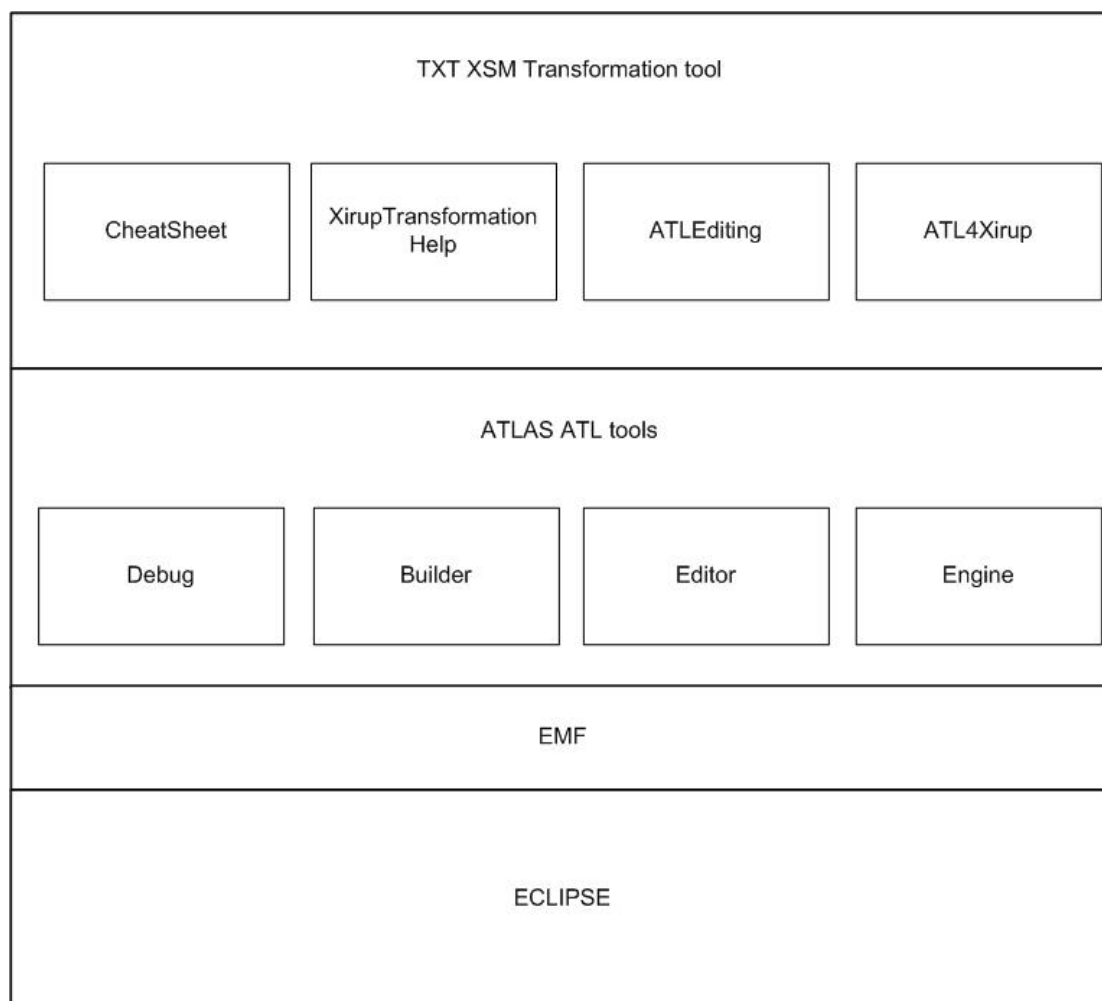
### 3.1 Requirements

The XSM Transformation Tool runs within Eclipse [Eclipse] and it requires the following software components:

- Eclipse Modeling Framework plug-ins [EMF]
- ATL plug-ins [ATL]

At present, Windows 2000/XP is the reference OS and JRE 1.6 is suggested even though JRE 1.5 is supported as well.

Figure 1 shows the TT architectural stack.



**Figure 1: XSM Transformation Tool stack**

## 3.2 Installation Guide

This section will guide the user throughout the overall installation process, starting from retrieving the Eclipse platform to the execution of TT environment.

### 3.2.1 Eclipse and EMF installation

The first step is installing both the Eclipse environment and the EMF framework.

Eclipse is available at <http://www.eclipse.org/downloads>, release 3.3.1, which results from the Europa Project [Europa].

EMF can be downloaded at:

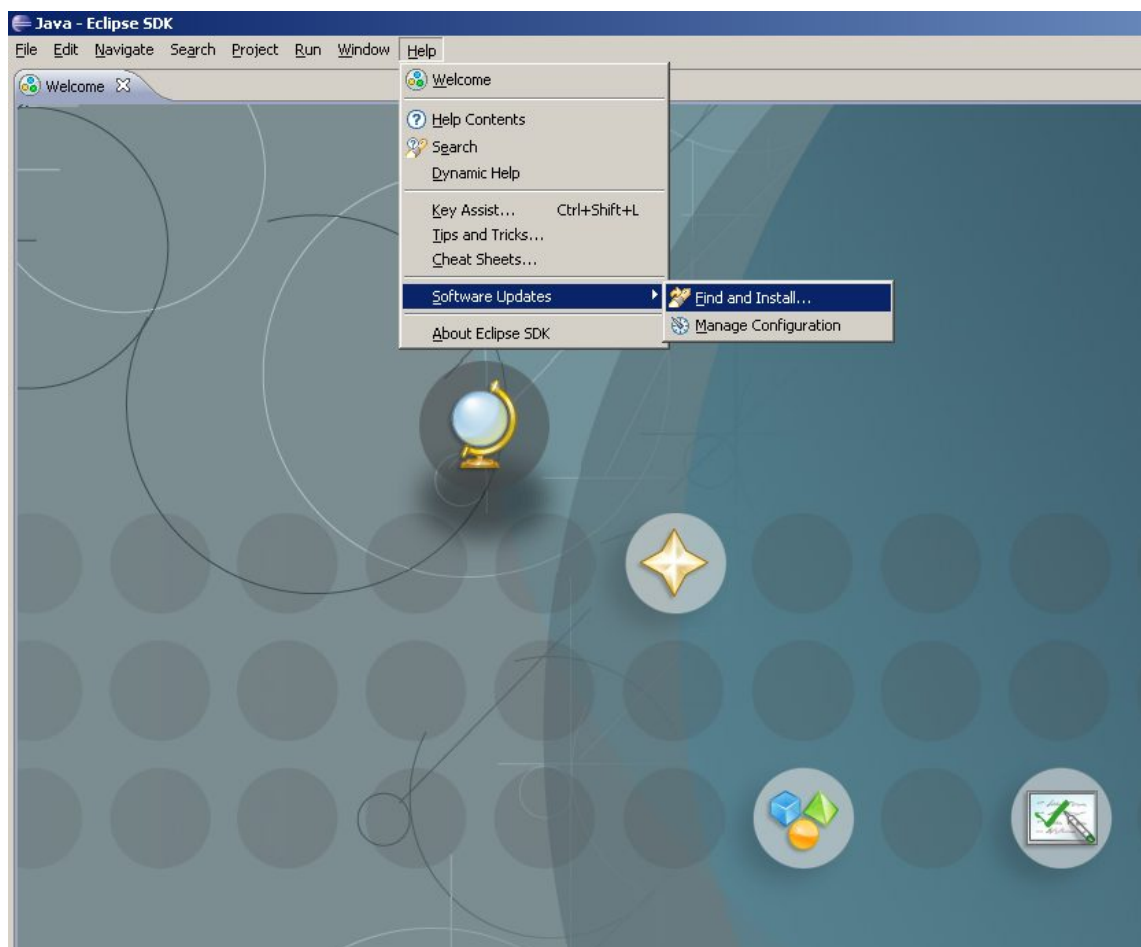
<http://www.eclipse.org/modeling/emf/downloads/?project=emf> (version 2.3)

which contains significant changes from previous releases.

Installing the Eclipse environment is simply achieved by unzipping the corresponding archive; this will create a “\eclipse” directory containing all the Eclipse files required. At this stage, the EMF installation consists in unzipping the EMF archive into the same target directory. This last operation writes the set of EMF plug-ins in the “\eclipse\plugins” directory. Another way to get EMF plug-ins is using the Eclipse Update Manager which prevents the user from manually downloading and unzipping the required archives and, instead, allows installing all plug-ins only by just giving the right http address.

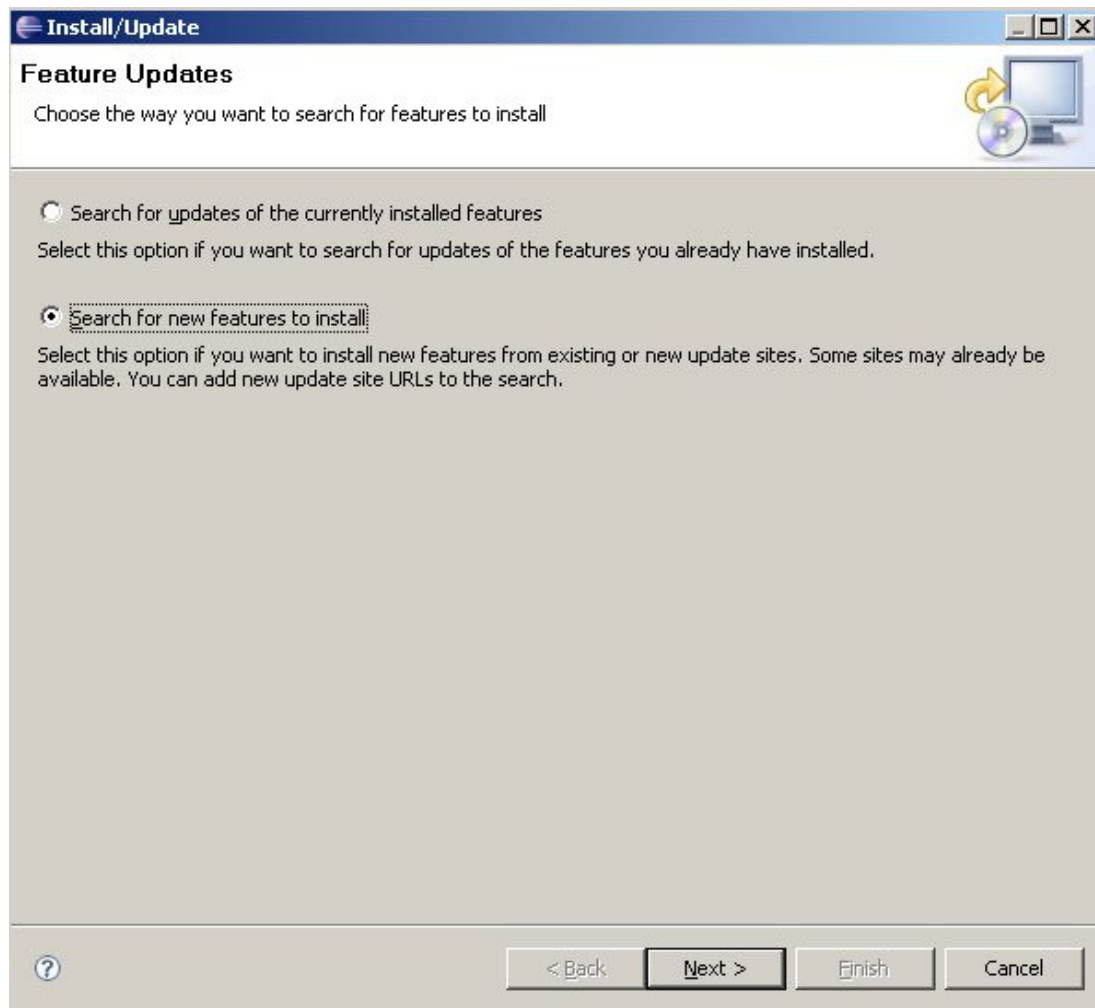
Once the basic Eclipse bundle has been installed, the user can go to (Figure 2):

Help -> Software Updates -> Find and Install



**Figure 2: Update Manager location**

and select the “Search for new features to install” radio button as displayed in Figure 3

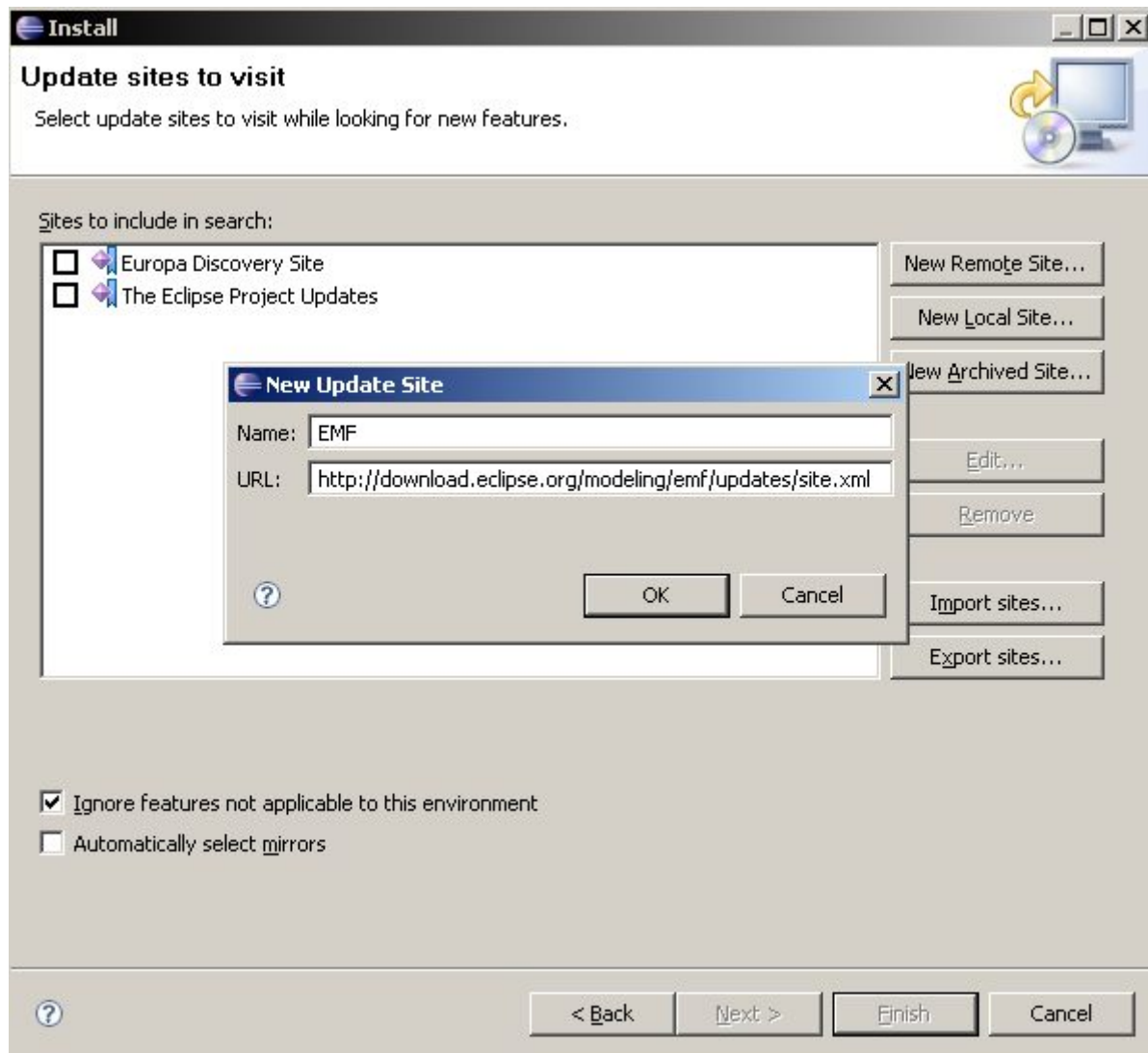


**Figure 3: New features to install**

By clicking on the “Next” button the user can now add a new remote site and get all the EMF required plug-ins. The right address is retrievable from EMF project web site at:

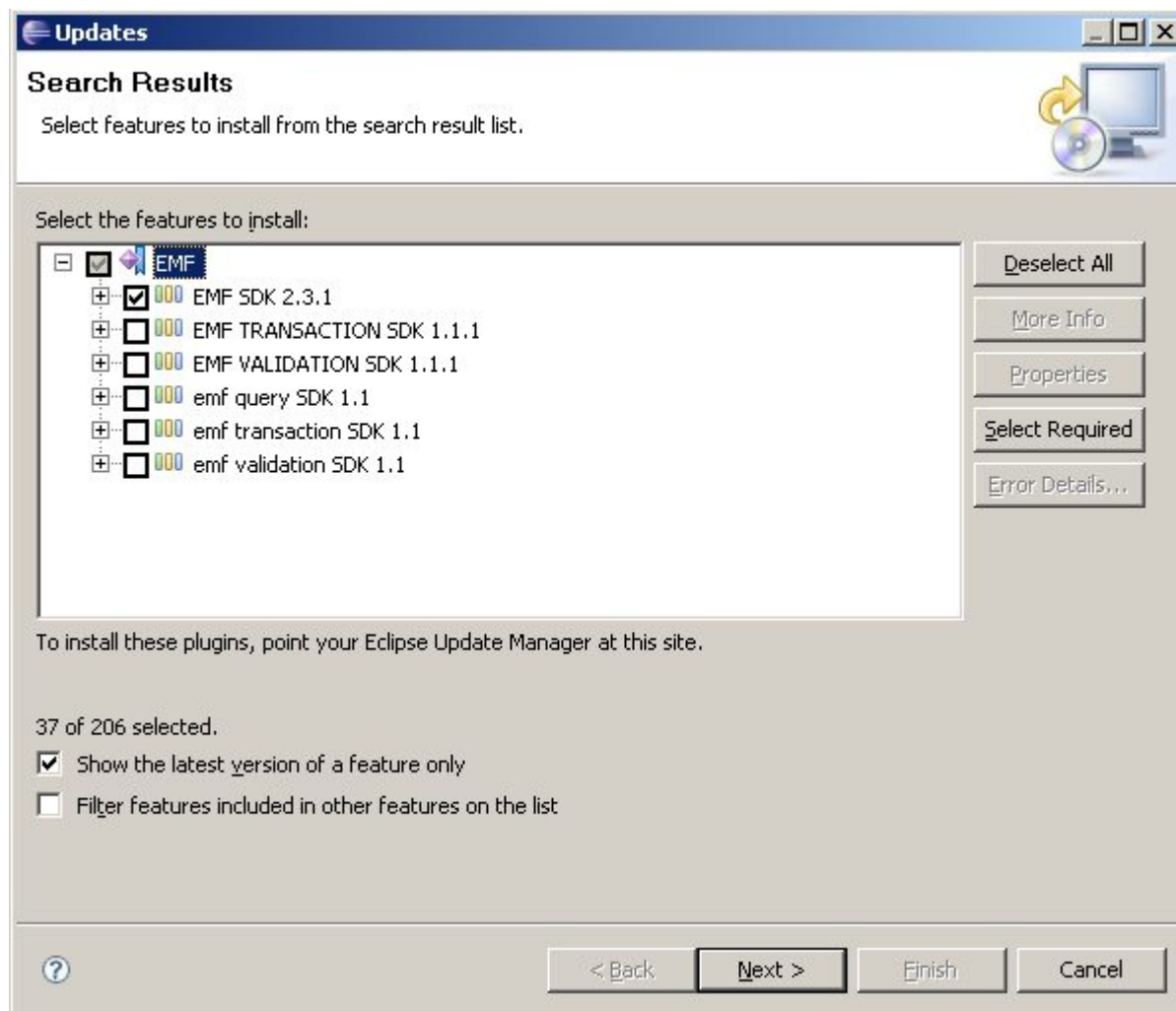
<http://www.eclipse.org/modeling/emf/updates>

Figure 4 shows this process.



**Figure 4: Adding EMF remote site**

Subsequently, the user must select which kind of plug-ins he/she requires. Please be sure that the “Show the latest version of a feature” check box is selected. After that, navigate through the EMF feature tree and tick “EMF SDK 2.3.1” only (Figure 5)



**Figure 5: Select EMF SDK feature**

By clicking the “Next” button, a licence agreement box is displayed and, after user’s acceptance, it leads to plug-in downloading and installation (Figure 6 and Figure 7).



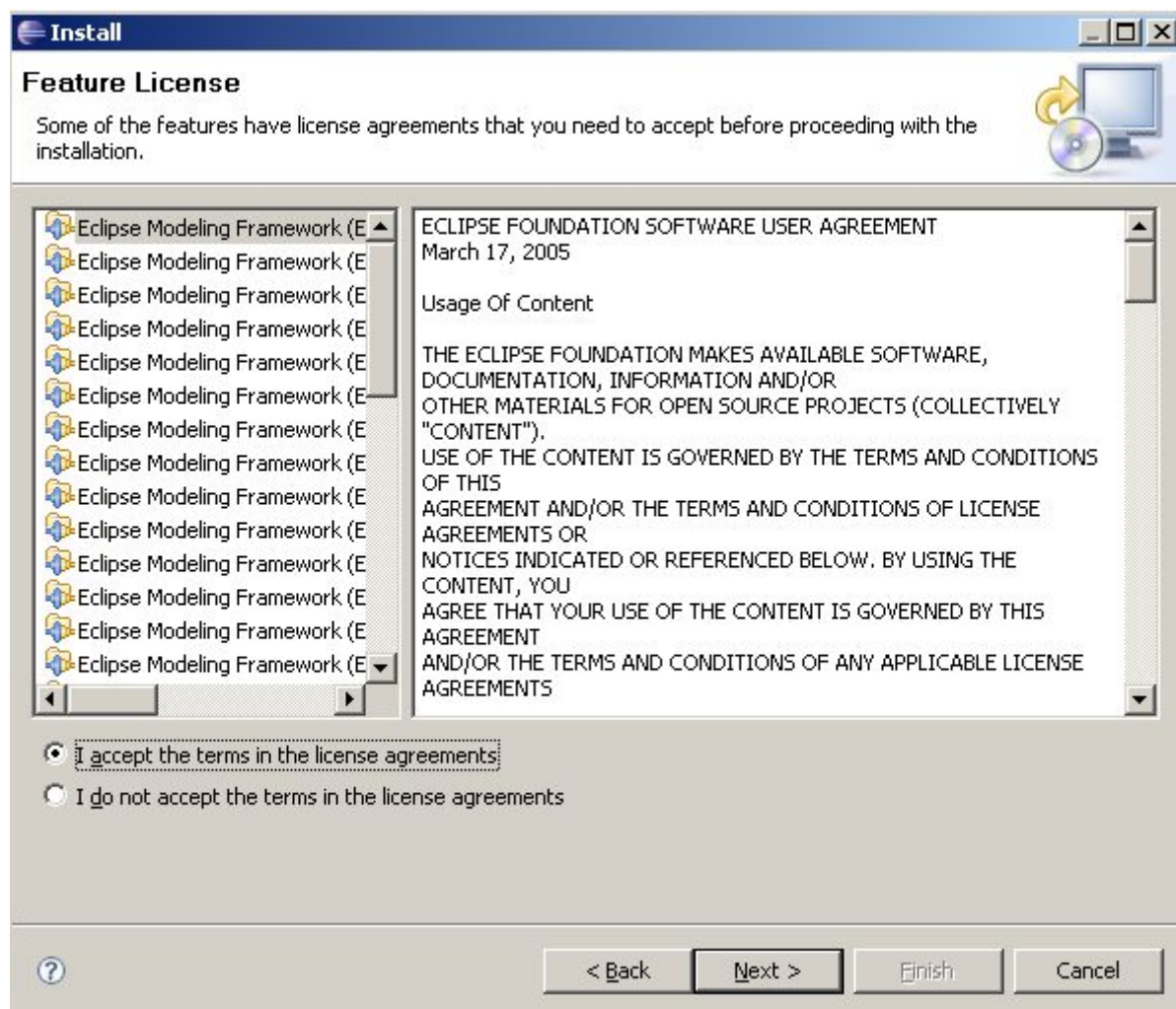
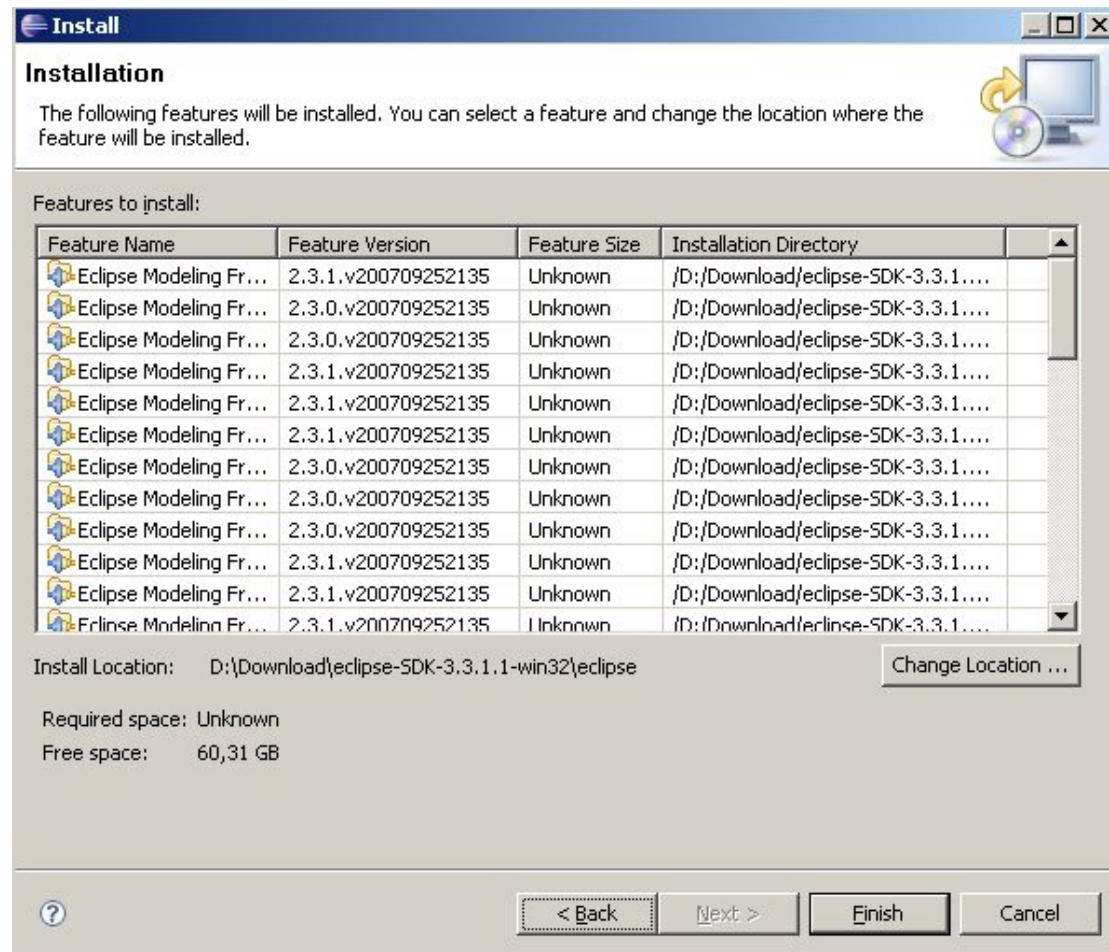
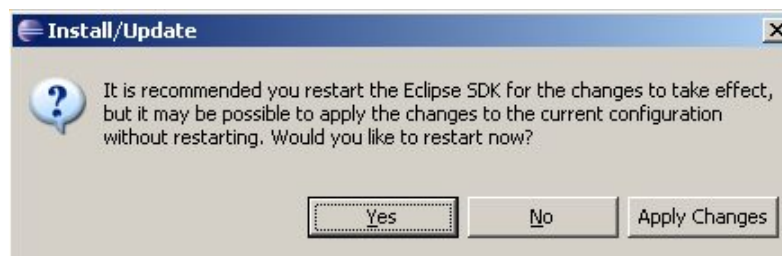


Figure 6: User's acceptance



**Figure 7: Final sum-up before installation**

Finally, the user is prompted to restart the whole application in order to let the changes take effect and have EMF correctly installed

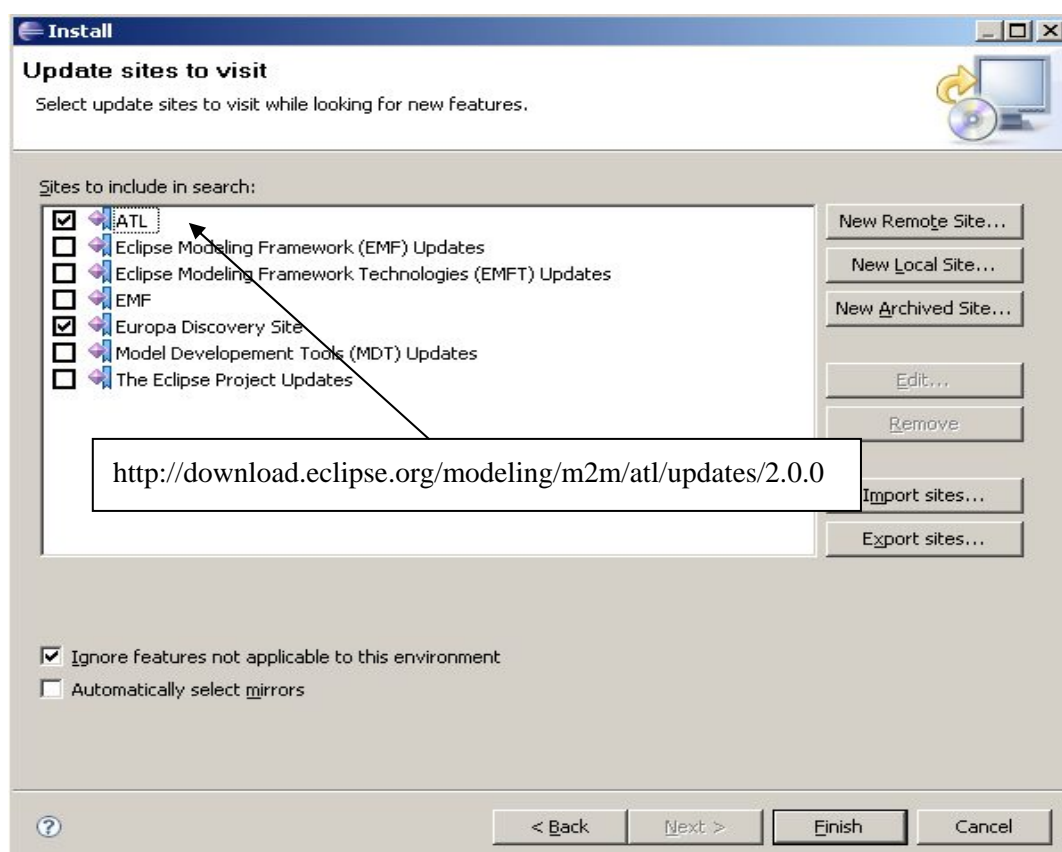


**Figure 8: Restart Eclipse**

### 3.2.2 Installing ATL

By following a similar process to the one depicted in the previous section, it is possible to install all the required plug-ins to get ATL correctly works. The simplest and most effective way to do that is once again by using the Update Manager.

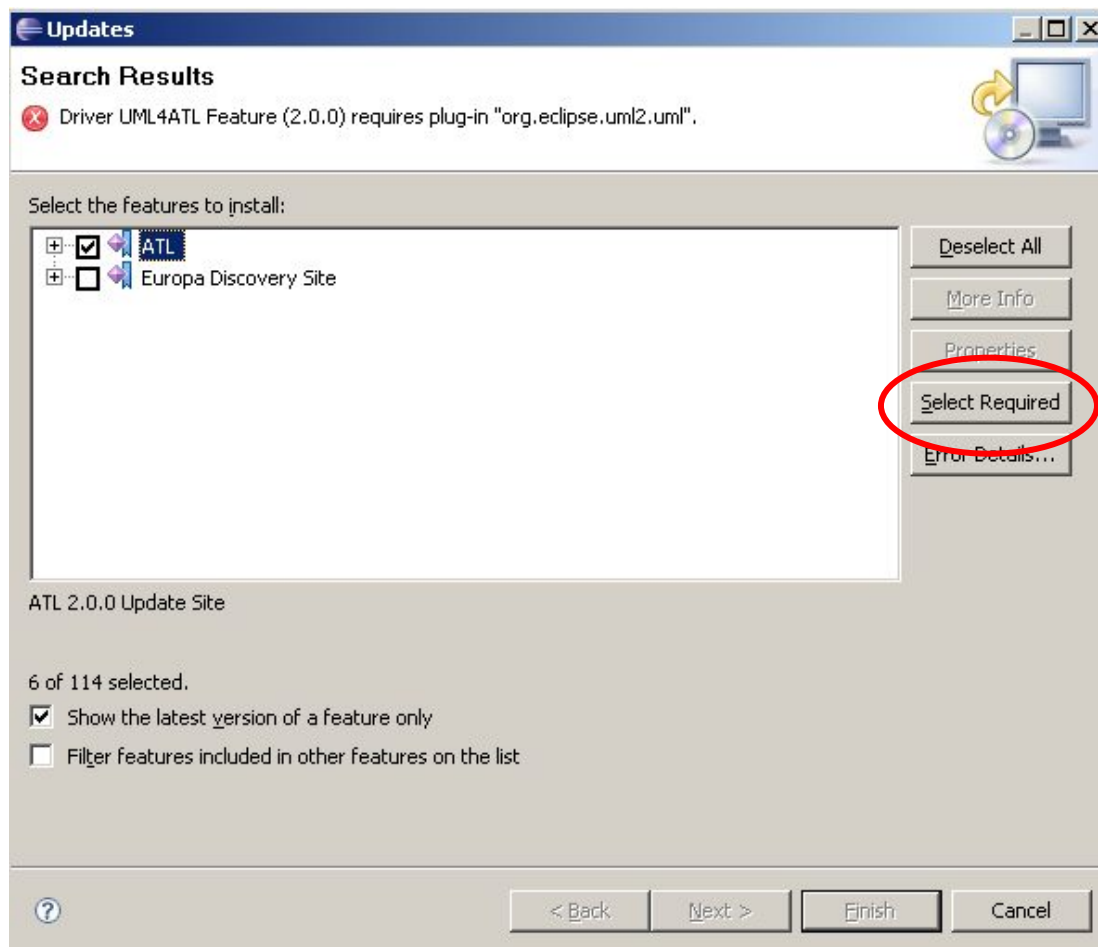
First of all, the right feature address must be retrieved from the ATL project web site (currently located at <http://www.eclipse.org/m2m/atl/download>) and a new remote site reference ought to be created in order to get all features (just call it “ATL”). Additionally, the already existing “Europa” remote site has to be selected to get and install some required plug-ins by ATL UML4ATL drivers (Figure 9).



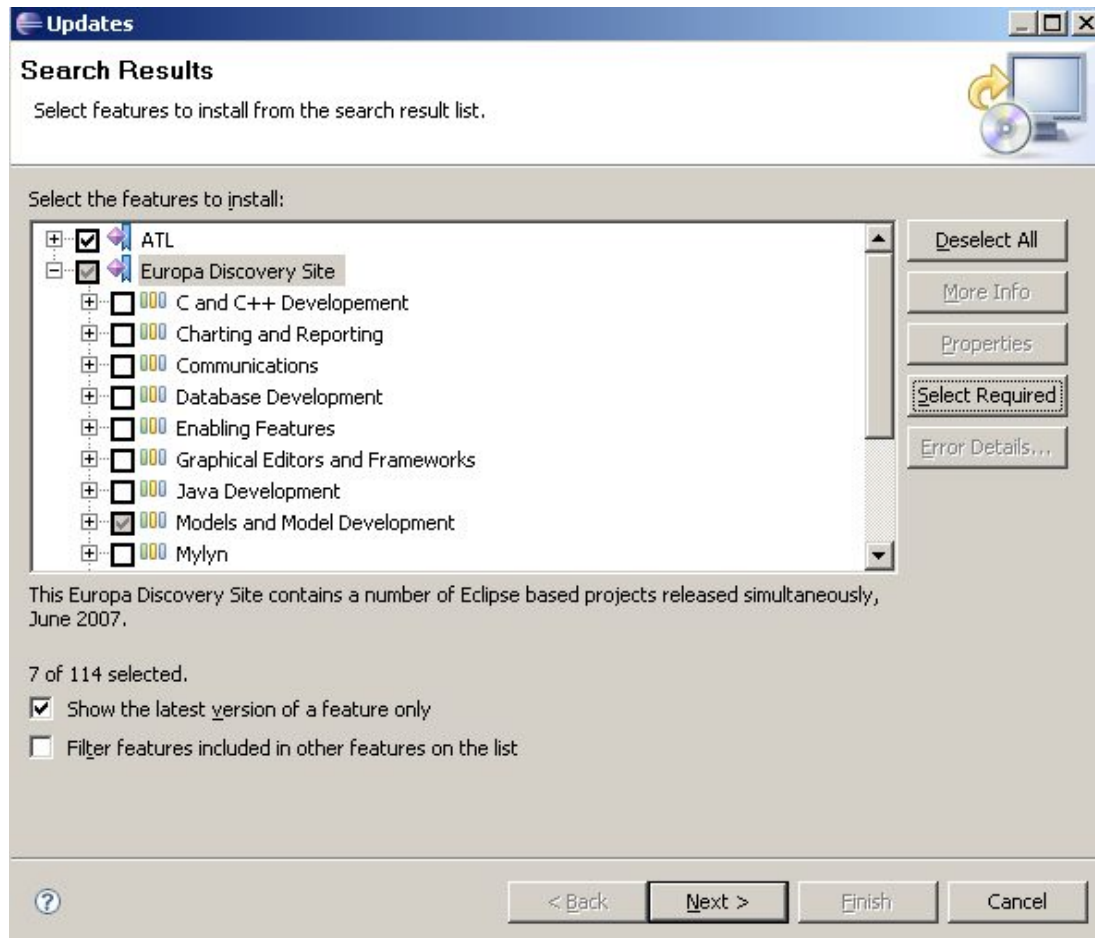
**Figure 9: ATL installation**

After ticking the ATL item and then clicking on the “Select required plug-ins” button, the Update Manager automatically selects all the bundles needed by ATL features.

In addition, and following the same procedure, UML2 Tools, UML2 SDK and UML2 Extender SDK [UML2Tools] features should be selected as well, since they provide functionalities to work on UML models: if missing plug-in errors appear, the user has just to load required bundles by pressing the related button. Figure 10, Figure 11 and Figure 12 show this process.

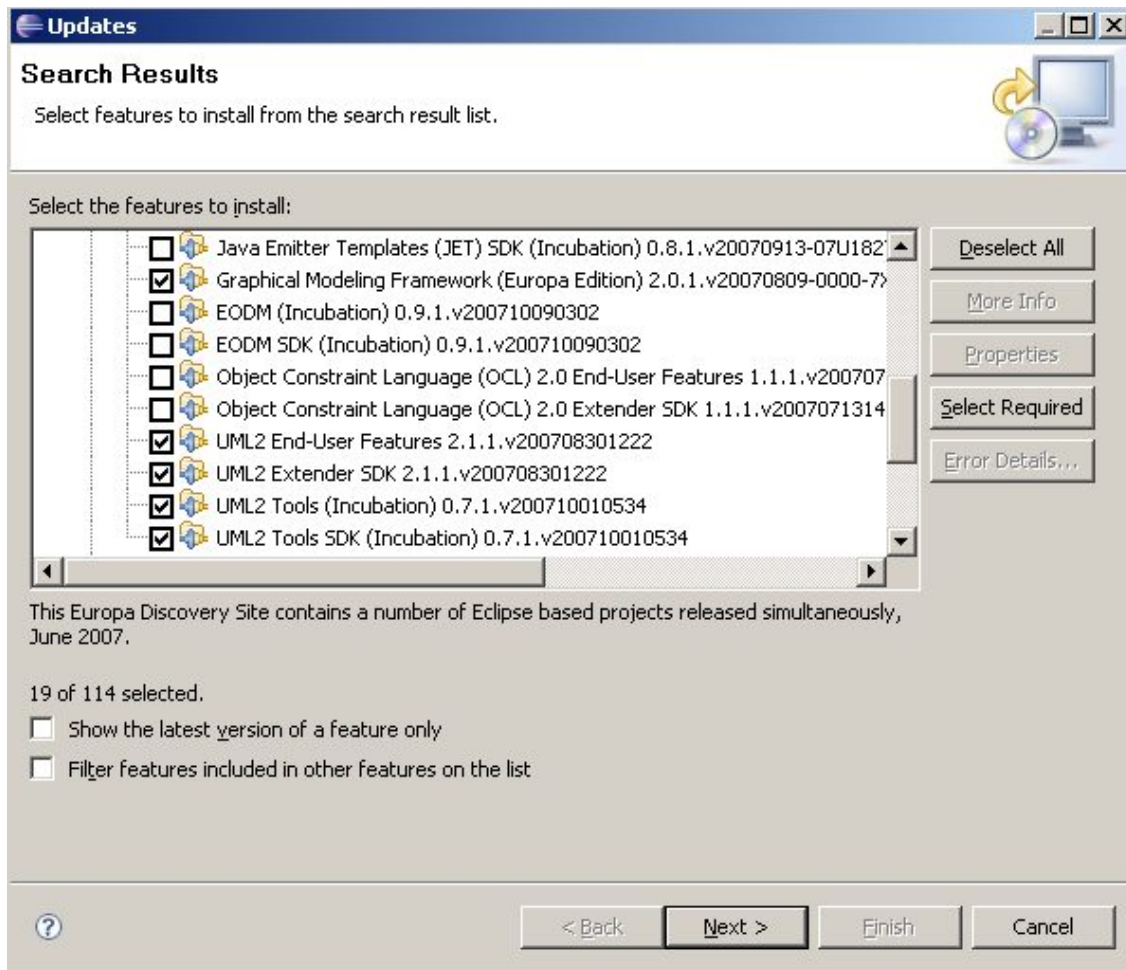


**Figure 10: Select ATL features**



**Figure 11: Select ATL required plug-ins**





**Figure 12: Adding UML2 tools**

After installation, Eclipse should be restarted as usual.

### 3.2.3 Installing XSM plug-ins

The XSM feature contains primarily an Ecore [ECORE] version of the reference Xirup metamodel and provides a simple tree viewer to edit instances of that metamodel.

Through the Update Manager it is possible to install the related plug-ins at [https://services.txt.it/crs\\_subversioning/momocs/WP5/dev/polimi/momocs/](https://services.txt.it/crs_subversioning/momocs/WP5/dev/polimi/momocs/).

The current XSM version is 1.0.1

### 3.2.4 Installing XSM Transformation Tool plug-ins

The TT plug-ins has been released as a zip file and they are downloadable from SVN at:

[https://services.txt.it/crs\\_subversioning/momocs/WP5/dev/txt/](https://services.txt.it/crs_subversioning/momocs/WP5/dev/txt/)


Plug-ins can be put into the Eclipse “\plugins” directory after unpacking the archive (version 1.0.1); they are situated in the “\bin” folder. There some additional folders containing source code, some resources (ATL example files), installation notes and a JavaDoc. An updateSite has also been created and it is situated under SVN, at: [https://services.txt.it/crs\\_subversioning/momocs/WP5/dev/txt/update/xsmtransformationtool/](https://services.txt.it/crs_subversioning/momocs/WP5/dev/txt/update/xsmtransformationtool/)

These are the names of each plug-in:

- org.momocs.txt.transformation.cheatsheet
- org.momocs.txt.transformation.atlending
- org.momocs.txt.transformation.atl4xirup
- org.momocs.txt.transformation.xiruptransformationphelp

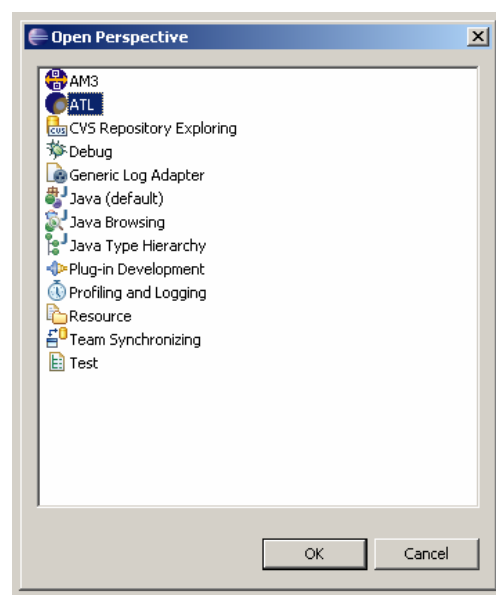
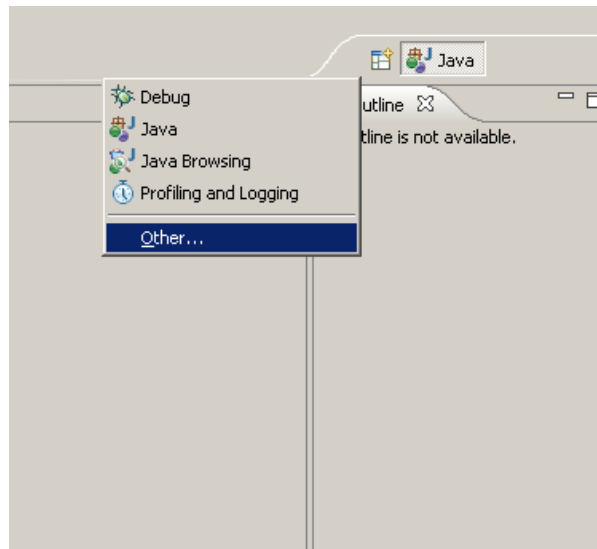
At this point, Eclipse is ready to be re-started in order to let all changes take effect.

## 3.3 Getting Started

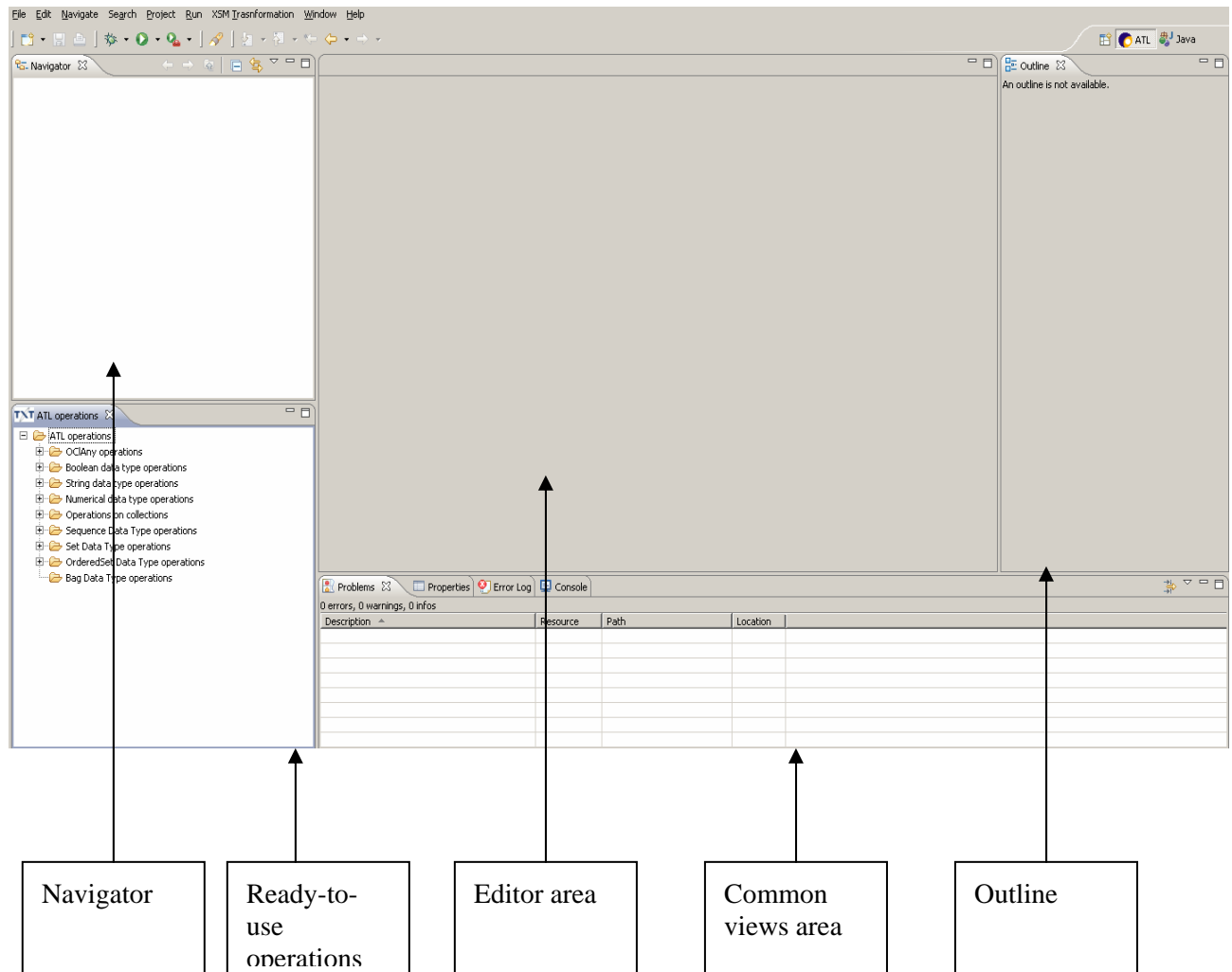
Launch Eclipse by double-clicking on the proper icon  and select the right perspective: TT extends the existing ATL perspective by means of specific views, menu and wizards. In the Eclipse Platform a perspective determines the visible actions and views within a window. Perspectives also go well beyond this by providing mechanisms for task oriented interaction with resources in the Eclipse



Platform, multi-tasking and information filtering [Eclipse\_ref]



**Figure 13: Select proper perspective**



**Figure 14: TT workbench**

Five major areas are displayed:

- Navigator – for workspace browsing
- ATL operations view – for easy use and browsing of ATL functions while editing transformations
- Editor area – for writing, editing and creating transformation rules

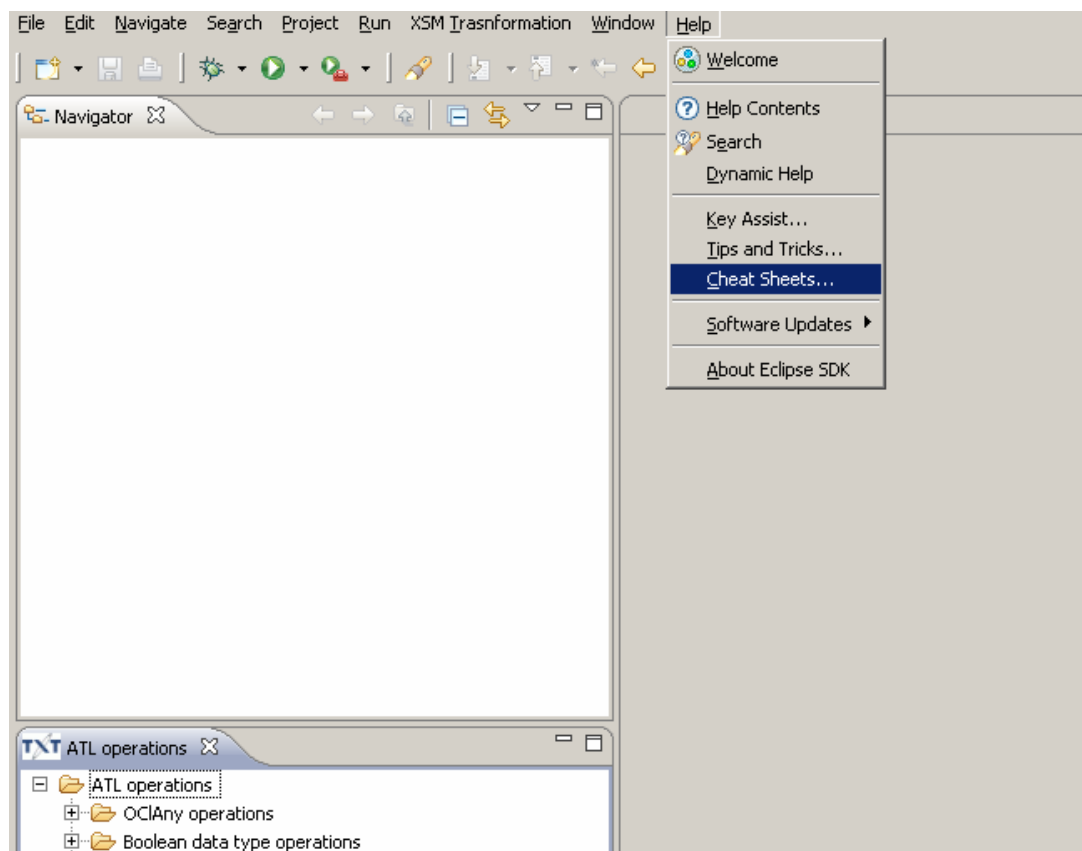
- Common views area – see resource properties, problems occurred, error logs and stack traces
- Outline – shows an outline of the structural elements of the editor

To get an idea of how TT works, which kind of resources it handles and the main common steps a user performs while running a transformation, a first interactive tutorial has been developed.

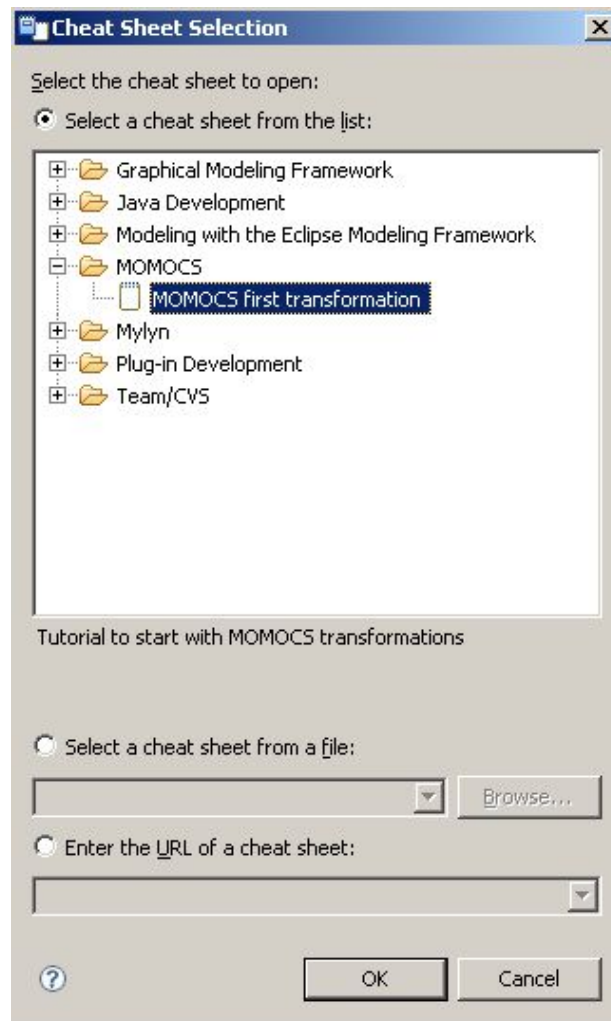
Eclipse provides a built-in mechanism for displaying mini-tutorials called *cheat sheets*. Cheat sheets are quick-and-dirty instructions showing how to perform multi-step processes and they are displayed on the side of the workbench where users can quickly and easily step through them. Available cheat sheets are displayed by clicking on:

- *Help > Cheat Sheets* or
- *Window > Show View > Other > Cheat Sheets > Cheat Sheets > OK.*

Cheat sheets appear as a view on the right side of the workbench, making it easy to simultaneously read and execute the cheat sheet instructions.



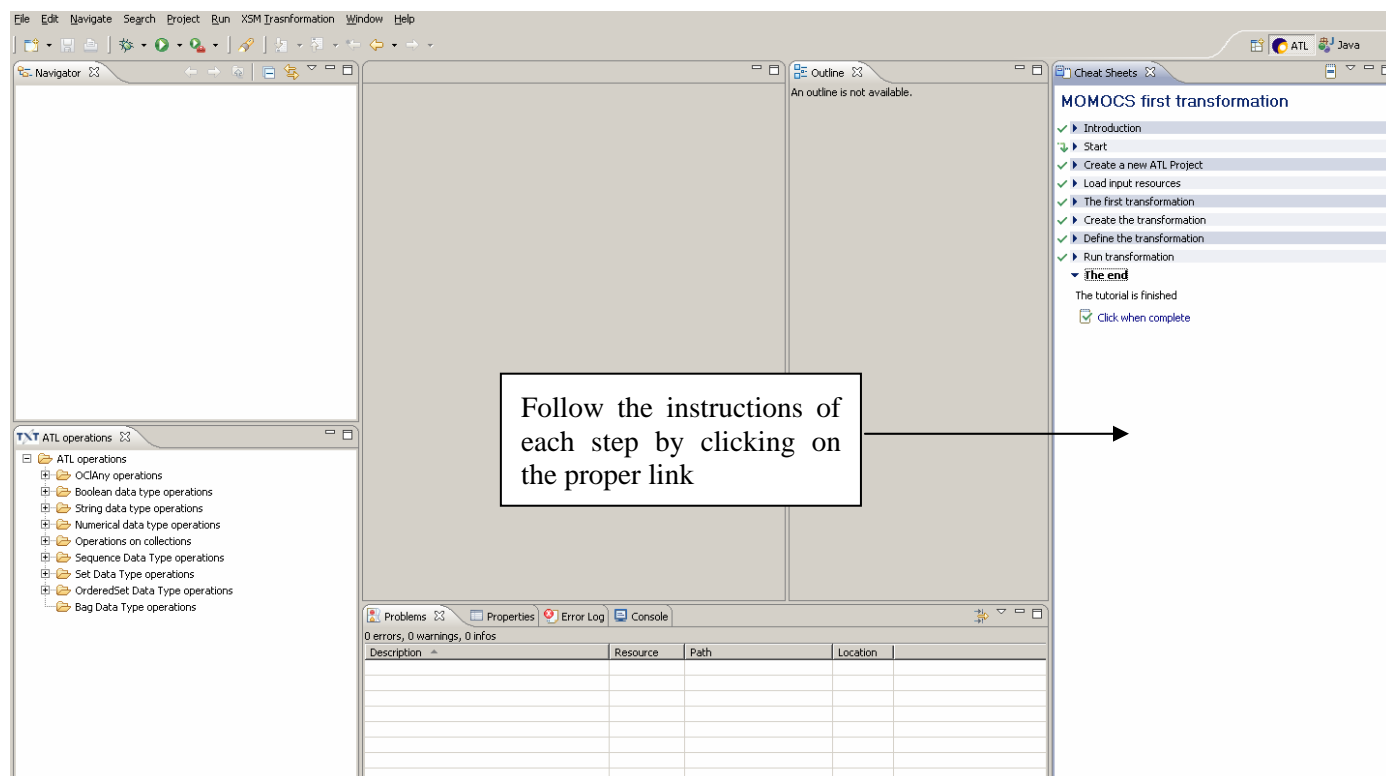
**Figure 15: Selecting the cheat-sheet (1)**



**Figure 16: Selecting the cheat-sheet(2)**

A first MOMOCS cheat sheet has been created to help the user start with transformations.

In the case of “MOMOCS first transformation”, the user is guided through the process of creating and importing a set of resources (a project, a transformation, input/output models and the Xirup metamodel) by means of simple instructions combined with automatic actions.



**Figure 17: TT cheat-sheet**

Due to the high number of interactions and resources involved while using transformations, we believe that this kind of approach, based on heavy user-interactions, can be useful to take a first look of TT.

## 4 Transformation Tool User's Guide

As depicted in section 2, TT is based upon the set of plug-ins provided by [ATL].

Thus, compiling and executing functionalities are managed by ATL tools while TT is mainly intended to help the user use them and to encapsulate them within the concepts that Xirup promotes.

As a consequence, the purpose of this deliverable is not describing the ATL language: the related documentation can be found at [ATL\_DOC].

For instance, TT extends ATL editing features and provides the user with facilities to get in touch with transformations.

Most importantly, TT has another primary goal to cope with: it actually strives to “bridge” the Xirup methodology and its related metamodel to the ATL concept of model transformation.

In few words, **TT is the tool that allows end-users to use ATL for Xirup, placing itself exactly “inside” the methodology** (see “ATL4XIRUP Library” section for details).

For this reason, one of the most important results is the generation of core TT-embedded ATL transformations for Xirup aiming at giving to every MOMOCS user is going to face model transformations, a set of solid basis to start with. This idea will be better explained in section 4.7

The TT is mainly composed of four components:

- CheatSheet: it provides quick tutorials for highly interacting get-started facilities
- ATLEditing: editing functionalities extending ATL

- ATL4Xirup library: provides ready-to-use base Xirup ATL transformations, related resources and user-interfaces to really speed up the use of transformations within the Xirup methodology
- XirupTransformationHelp: help contents

Each bullet above will be described in more detail throughout the current deliverable.

Next sections describe common actions and TT enhancing features.

In addition to this User's Manual, **screencasts** are available at:

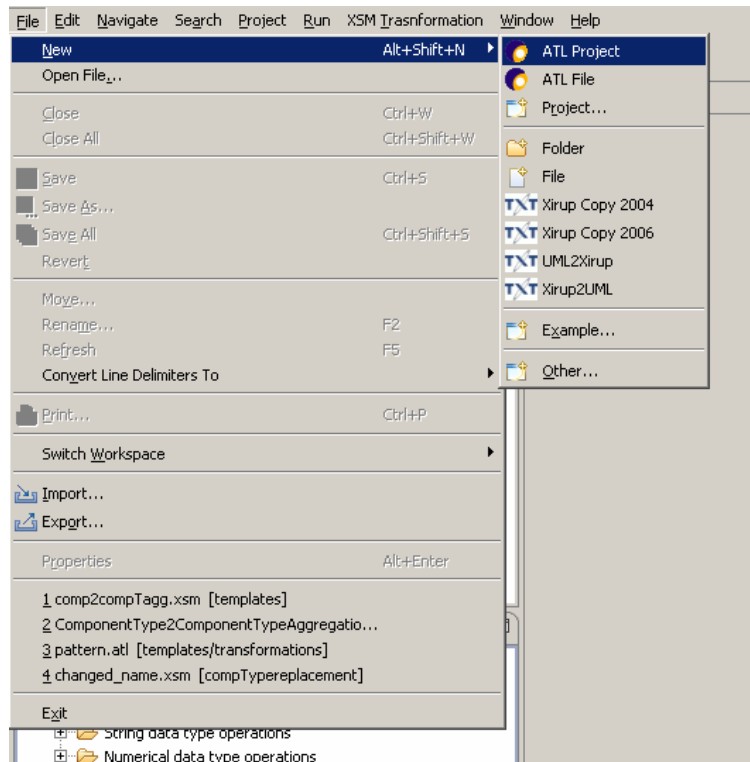
[https://services.txt.it/crs\\_subversioning/momocs/WP5/dev/txt/screencast/](https://services.txt.it/crs_subversioning/momocs/WP5/dev/txt/screencast/)

They describe TT functionalities by means of videos focusing on common activities.

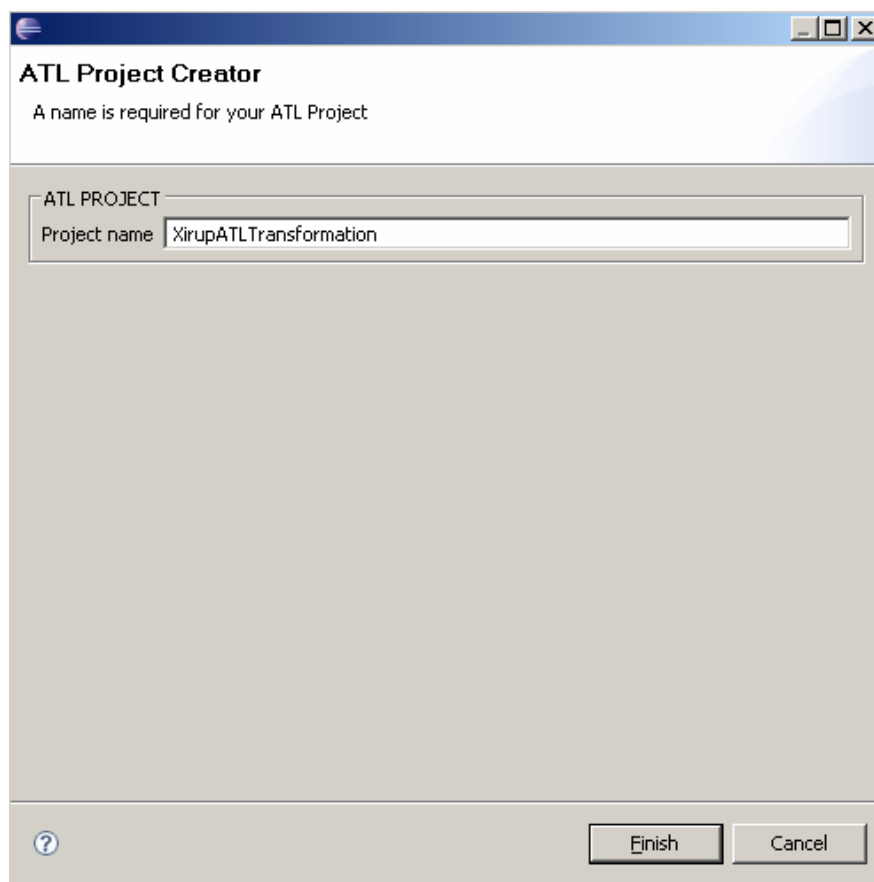


## 4.1 Setting up a new project

The user can create a new ATL project by selecting File->New->ATL Project and using the related wizard.



**Figure 18: Selecting wizard to create a new ATL project**



**Figure 19: Setting up the name**



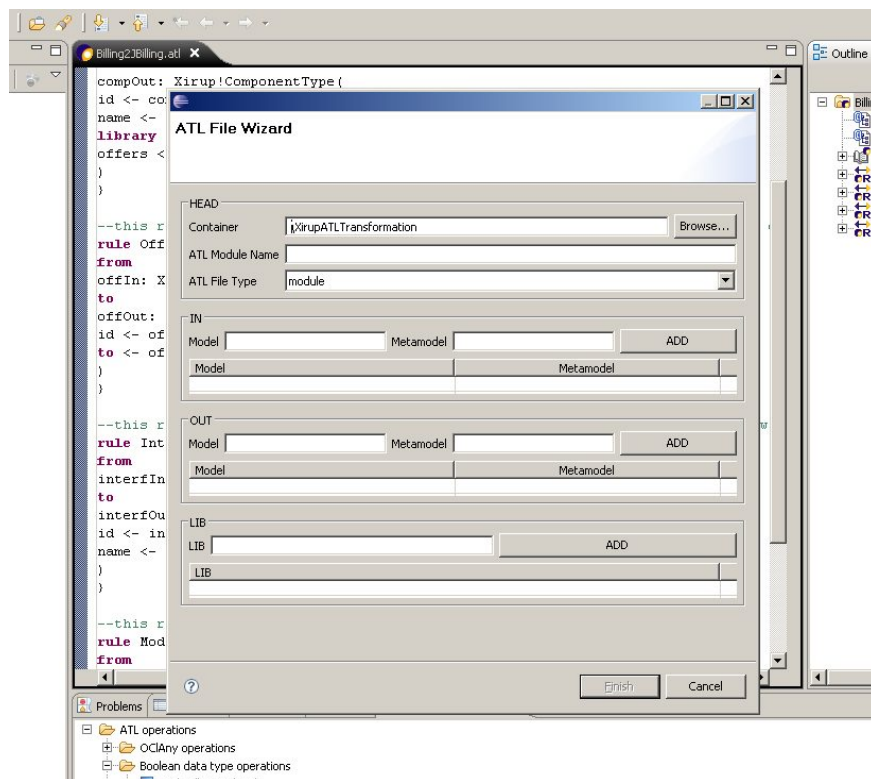
**Figure 20: Project in the Navigator view**

After that, the Navigator view will display the project just created which will act as a transformation container during the modernization process.

## 4.2 Initializing a transformation

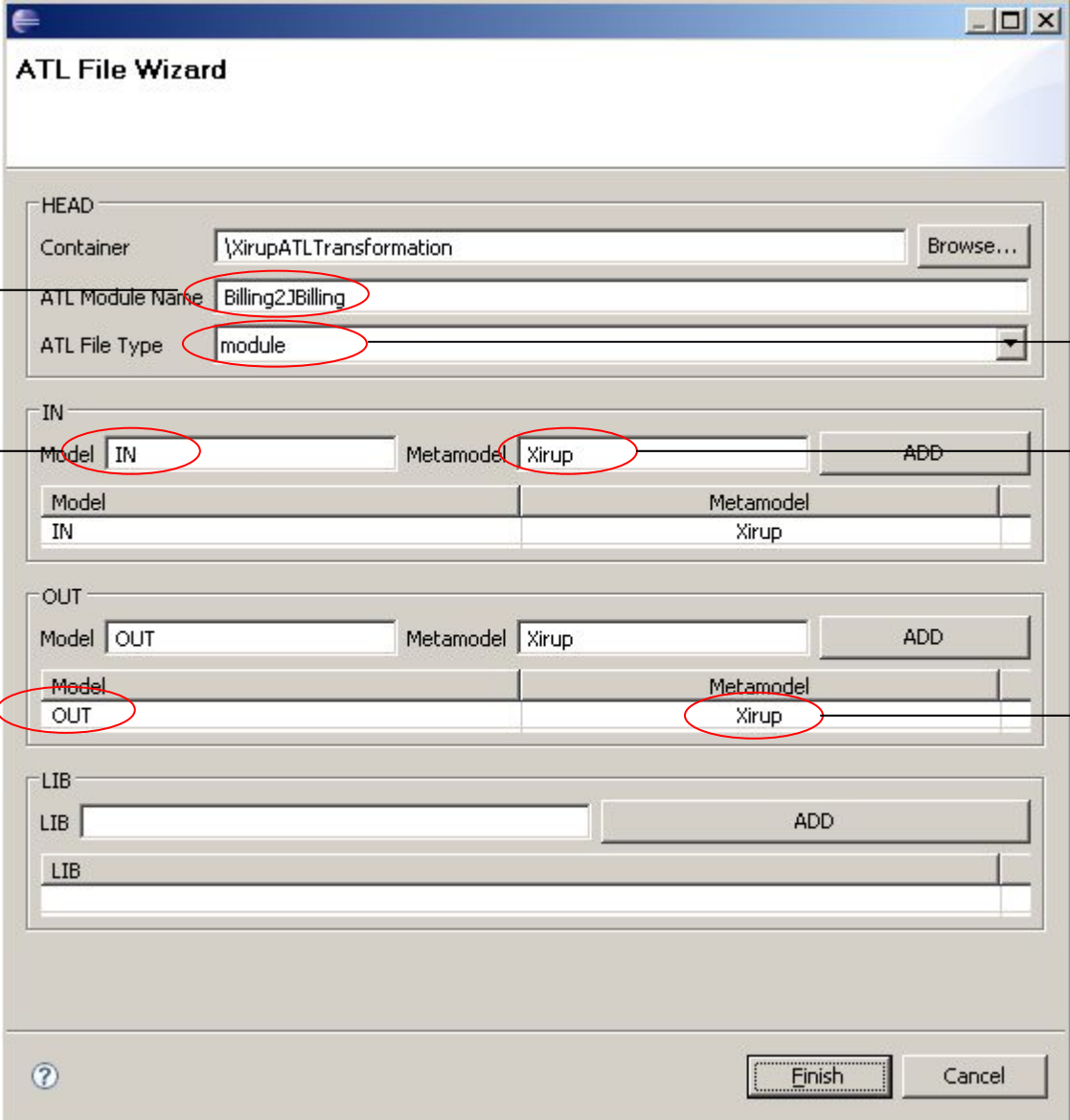
A new transformation is created and initialized by using the ATL file wizard accessible through File->New->ATL File.

Figure 21 shows the wizard to create a new ATL transformation: most important fields to be filled in are the name of the transformation itself (here called “Module”) and input/output metamodel/model labels which are used to access source and target model elements during the editing process.



**Figure 21: New ATL File wizard**

Figure 22 shows instead a module creation example.



The screenshot shows the 'ATL File Wizard' dialog box. It has several sections: HEAD, IN, OUT, and LIB. Annotations 1 through 6 point to specific fields:

- 1 points to the 'ATL Module Name' field, which contains 'Billing2JBilling'.
- 2 points to the 'ATL File Type' dropdown menu, which is set to 'module'.
- 3 points to the 'Model' field in the 'IN' section, which contains 'IN'.
- 4 points to the 'Metamodel' field in the 'IN' section, which contains 'Xirup'.
- 5 points to the 'Model' field in the 'OUT' section, which contains 'OUT'.
- 6 points to the 'Metamodel' field in the 'OUT' section, which contains 'Xirup'.

Other visible fields include 'Container' (set to '\\XirupATLTransformation'), 'LIB' (empty), and buttons for 'Browse...', 'ADD', 'Finish', and 'Cancel'.

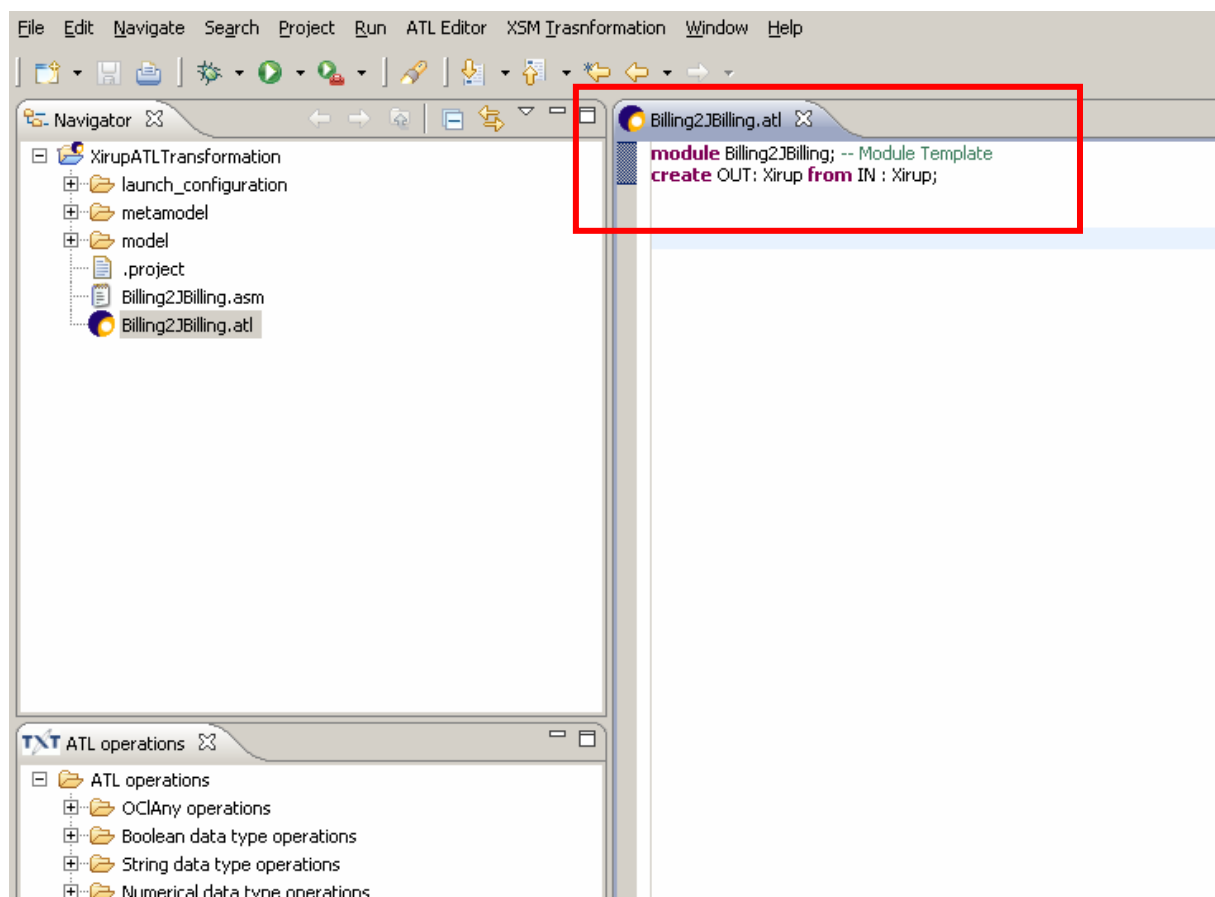
**Figure 22: ATL File creation**

Basic fields to be filled in are:

1. name of the transformation
2. type of transformation

3. label to identify input model within the transformation
4. label to identify input metamodel within the transformation
5. label to identify output model within the transformation
6. label to identify output metamodel within the transformation

After inserting the desired labels to identify model elements within the ATL module, the user can press the “Finish” button to trigger the creation of the related ATL file (Figure 23 below).



**Figure 23: Transformation header**

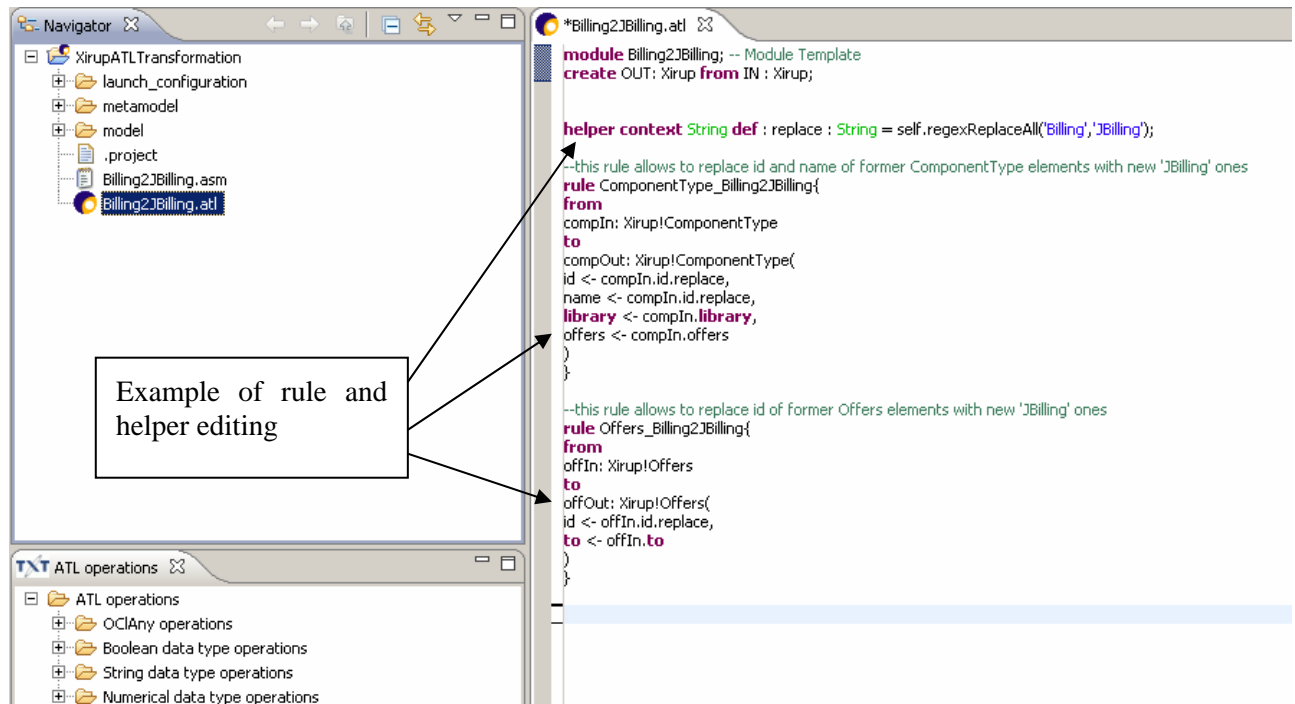
The header shown in the picture above states that: “this module ‘Billing2JBilling’ creates a target model labelled ‘OUT’ from a source model labelled IN”. Under this header, declarative and imperative ATL instructions (see [ATL\_DOC] for references and documentation) must be inserted in order to explicitly declare all rules performing the mappings and allowing to get the desired output model from an input model.

### 4.3 Editing

Despite ATL is a powerful tool and, in addition, it’s published together with a good documentation [ATL\_DOC], it is not as user-friendly as a new MOMOCS user could desire.

Textual editing for writing proper rules can be not much intuitive and continuous references to documentation and manuals for using the right syntax could slow the modernization process.

Figure 24 shows an example.



**Figure 24: Common editing activity**

Therefore, as long as we have been coping with ATL transformations and editing activities, we took notes about possible editing improvements for helping end-users master and manage transformations.

Next subsections will describe TT editing features extending and enhancing current editing functionalities.

These features represent a first contribution towards a “friendlier” editing support.

### 4.3.1 Adding ATL expression

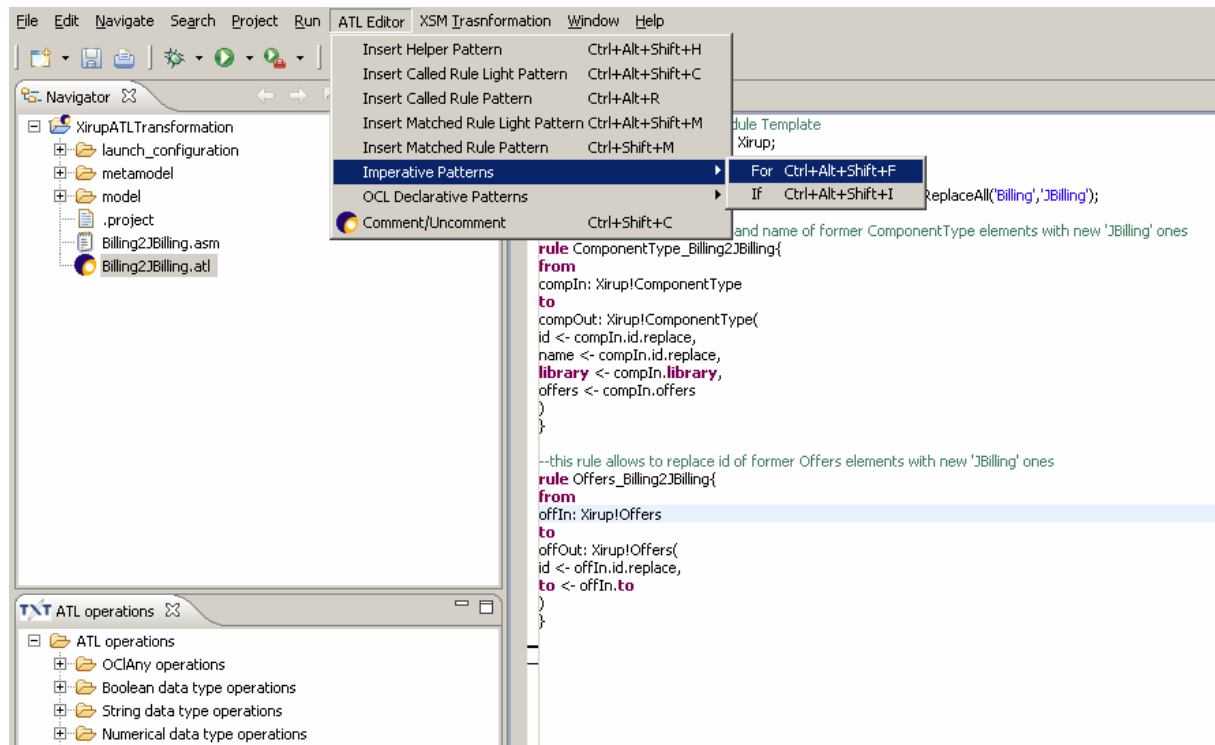
TT improves the “ATL Editor” with ready-to-use templates in order to use the main language constructs.

These constructs are the following (please see [ATL\_DOC] for references):

- Helper
- Matched rule
- Light Matched rule – reduced pattern with the most used keywords only (chosen according to our experience)
- Called rule
- Light Called rule – see “Light Matched rule”
- Imperative expressions:
  - If
  - For
- OCL [OCL] declarative expressions:
  - If
  - Let

Figure 25 shows a screenshot of this extended menu.





**Figure 25: ATLEditing menu**

They can be easily inserted thanks to menu actions to which also key-binding features are associated. This way, users (especially the first times they're using ATL transformation) don't have to keep on checking manuals and documentation to get references for right syntax.

Figure 26 in the next page shows an example of inserting a "Called Rule Pattern" block by clicking on the proper menu item, while Figure 27 displays a "Matched Rule Pattern" insertion obtained, instead, by means of key-binding.

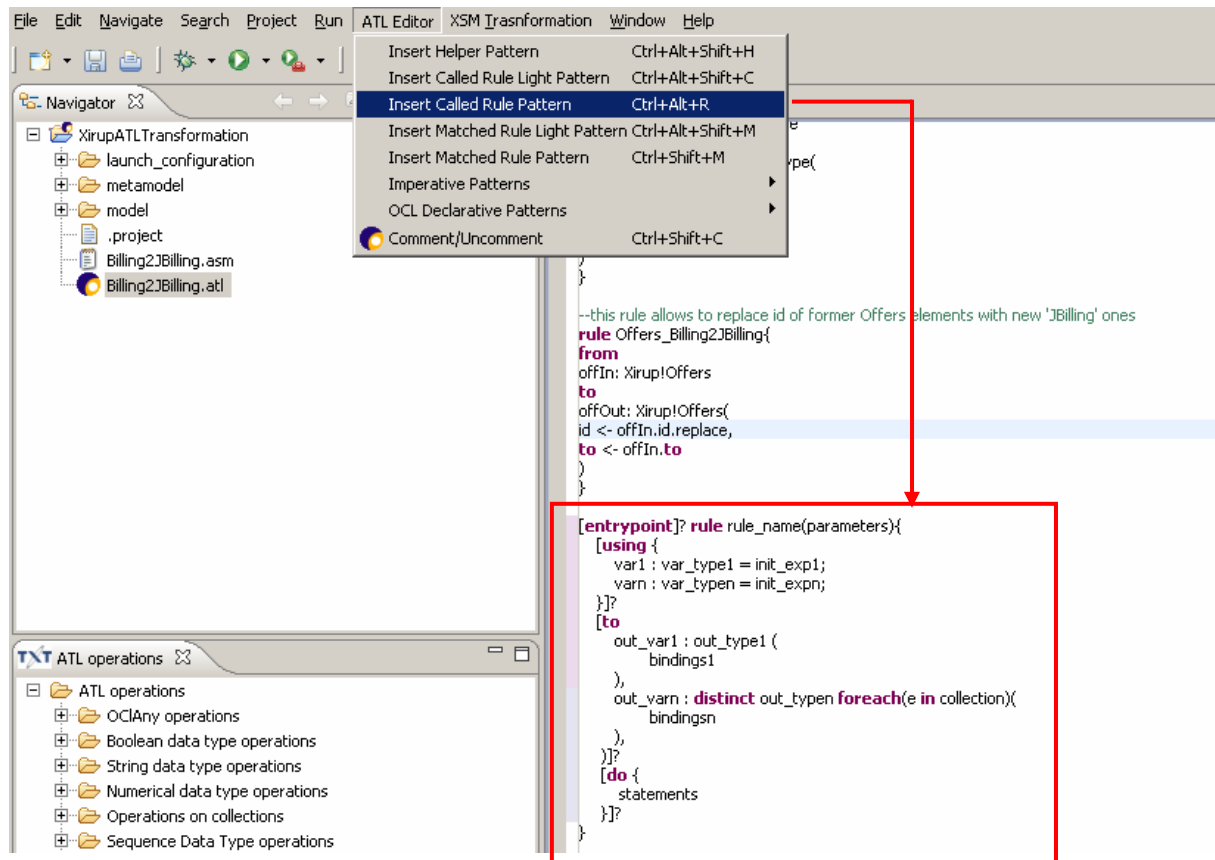
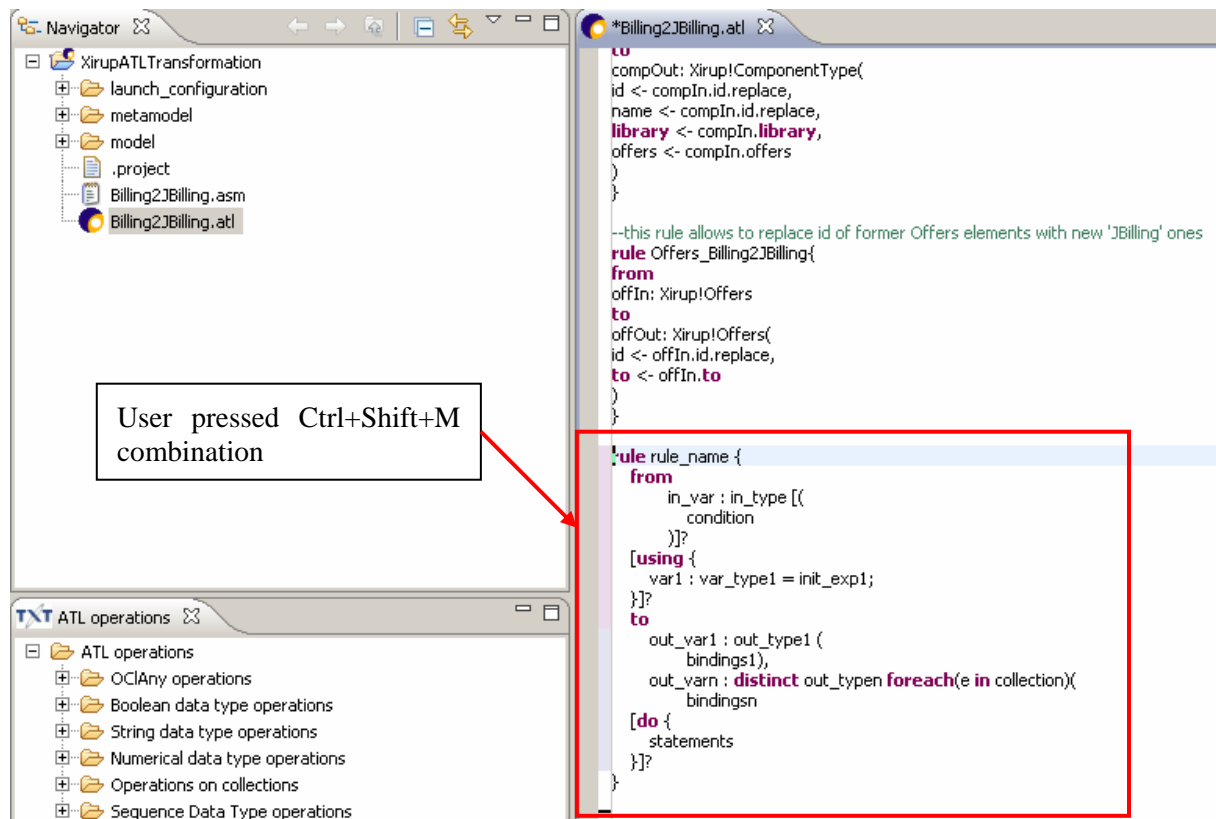


Figure 26 - Insert Called rule pattern



**Figure 27: Insert matched rule pattern**

After inserting the desired pattern, the user can then modify it and customize it (Figure 28).

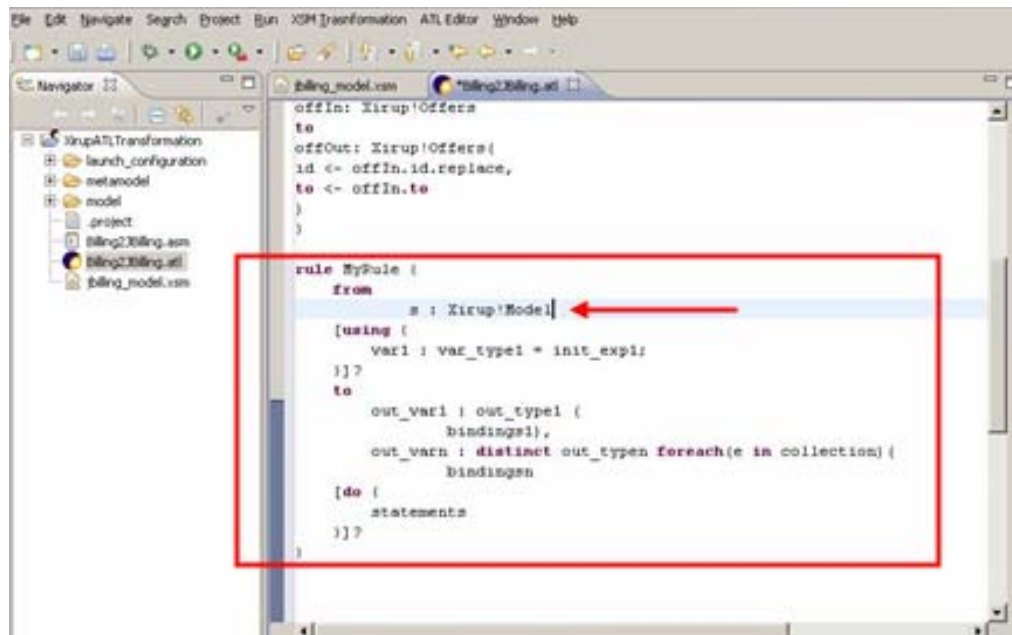


Figure 28: Customizing pattern

#### 4.3.2 Tree structured view for ATL operations

Apart from general constructs used to express both declarative and imperative instructions (as discussed in the previous section), the ATL language also provides a set of both OCL-derived and characteristic operations.

They are organized as follows:

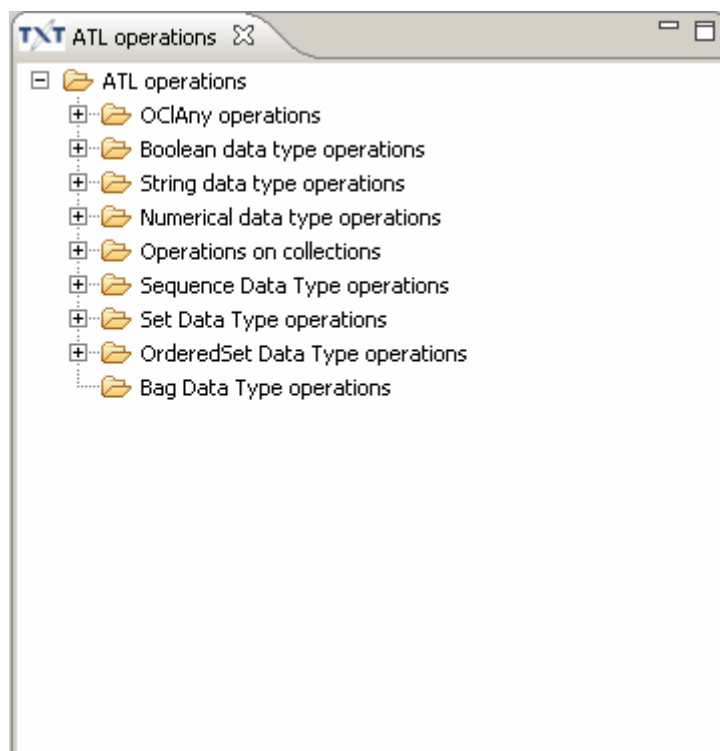
- OclAny operations
- Boolean Data Type operations
- String Data Type operations
- Numerical Data Type operations
- Operations on collections

- Sequence Data Type operations
- Set Data Type operations
- OrderedSet Data Type operations
- Bag Data Type operations (defined by OCL specification but not still available with current ATL implementation)

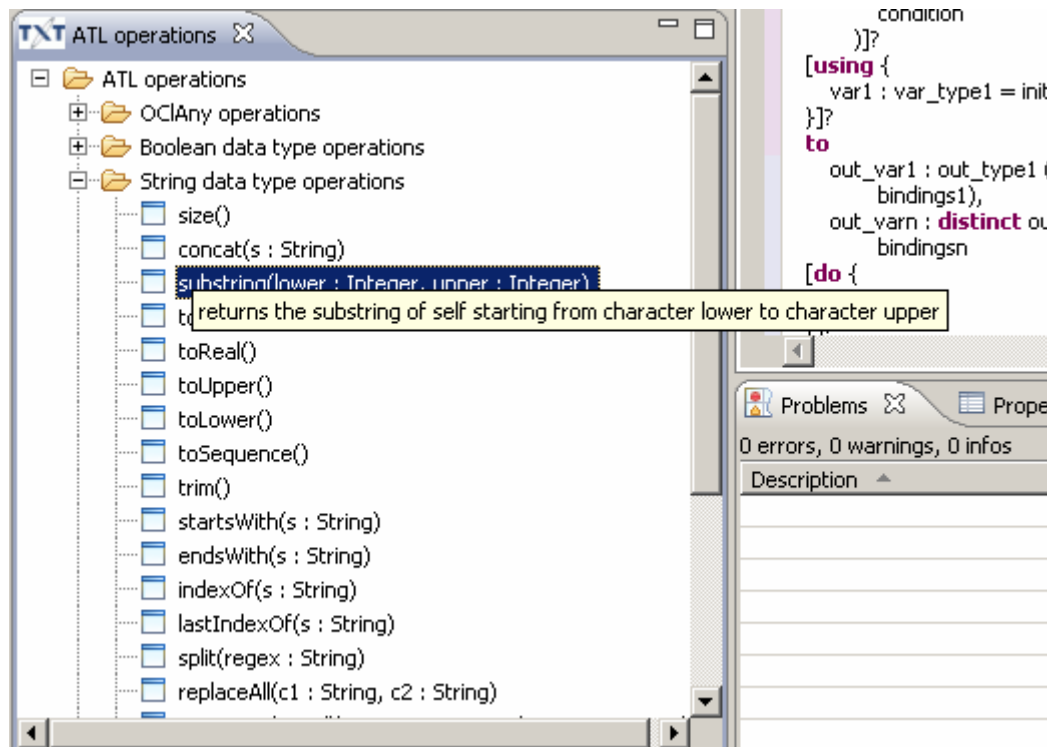
It's not however in the scope of this document explaining the meaning and the use of these operations, but instead showing how TT aids the user to manage them.

TT provides a tree-structured Eclipse view showing all main ATL operations organized according to the separation presented in the bullets above.

Each item is associated to a tip describing the item itself and thus giving information about the operation it represents: user's double-click will trigger the insertion of the correct operation syntax into the ATL file which is actually in editing mode. This view can be thus considered an "active" manual reference provided by TT to make the process of creating and editing transformations much faster.

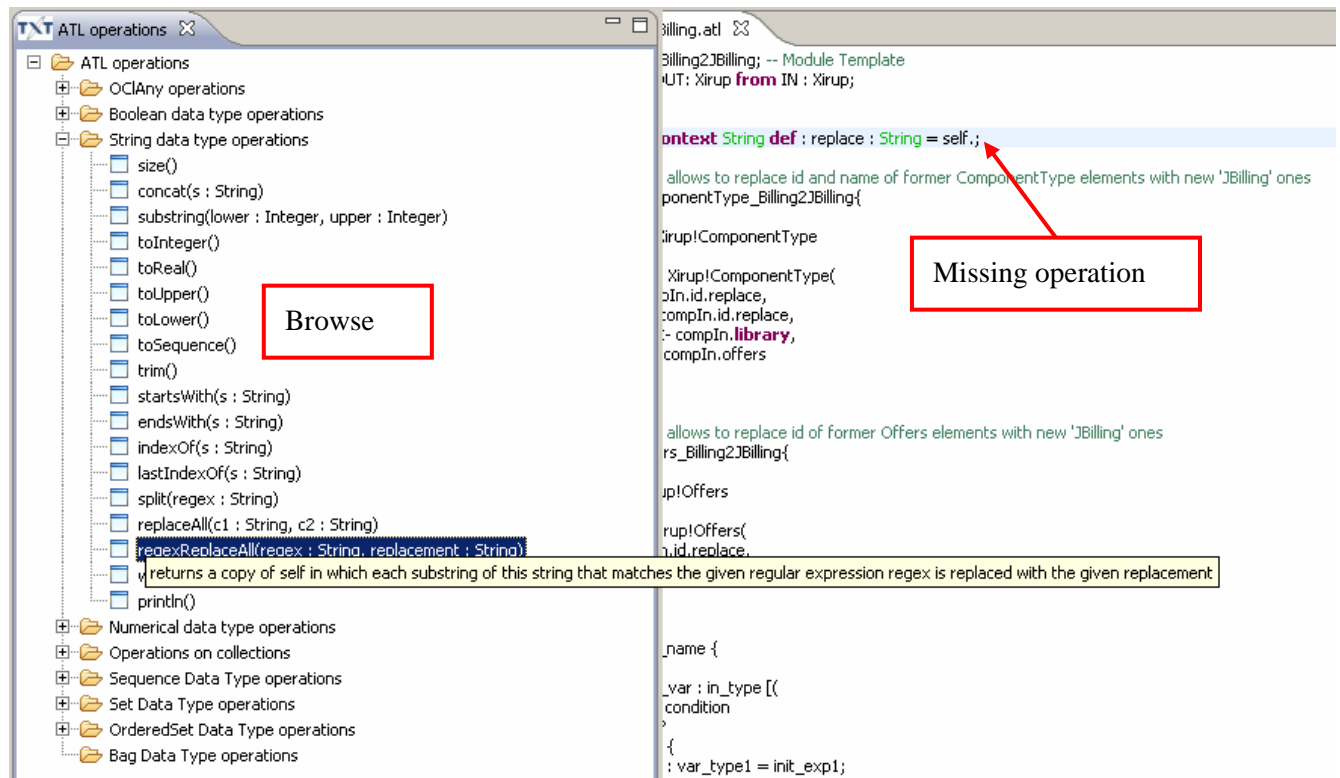


**Figure 29: ATL operations Tree viewer (1)**



**Figure 30: ATL operations Tree viewer (2)**

Thus, the user can browse the operations organized in categories and select the one it's most suitable for the current task (Figure 31).



**Figure 31: Browse operations and select**

By double-clicking on the selected item, the related operation pattern is inserted into the transformation currently in editing mode (Figure 32)



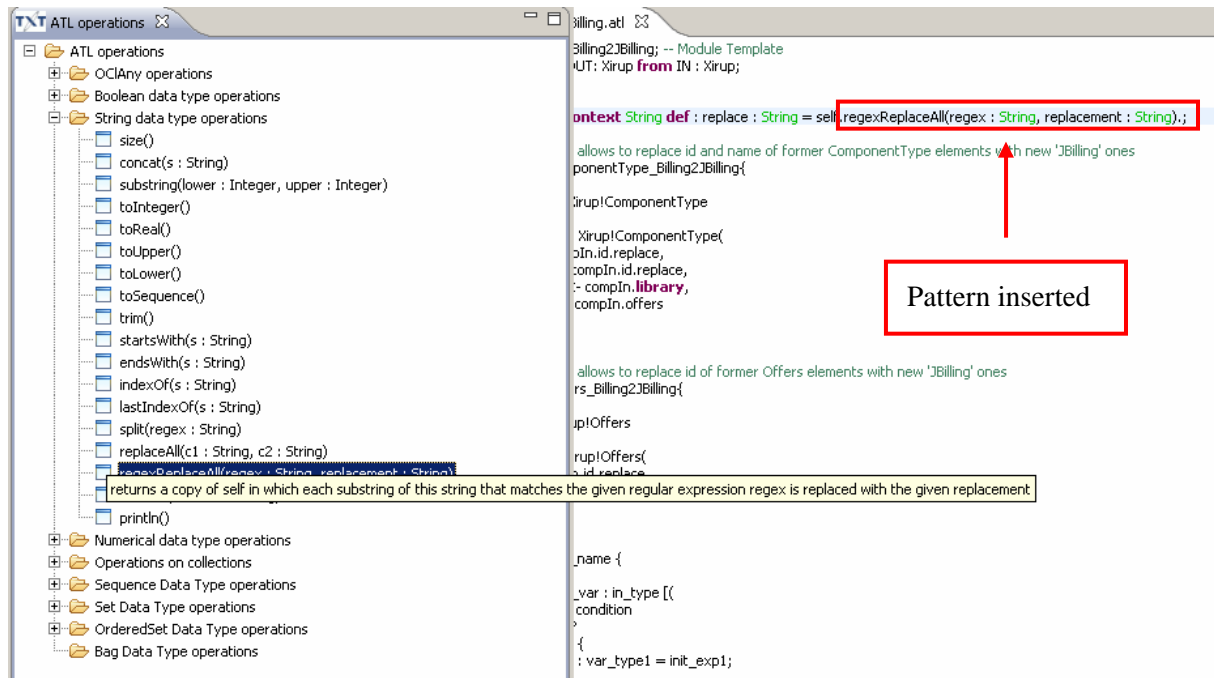


Figure 32: Double-click on operation

And finally the user can modify it according to actual needs (Figure 33)

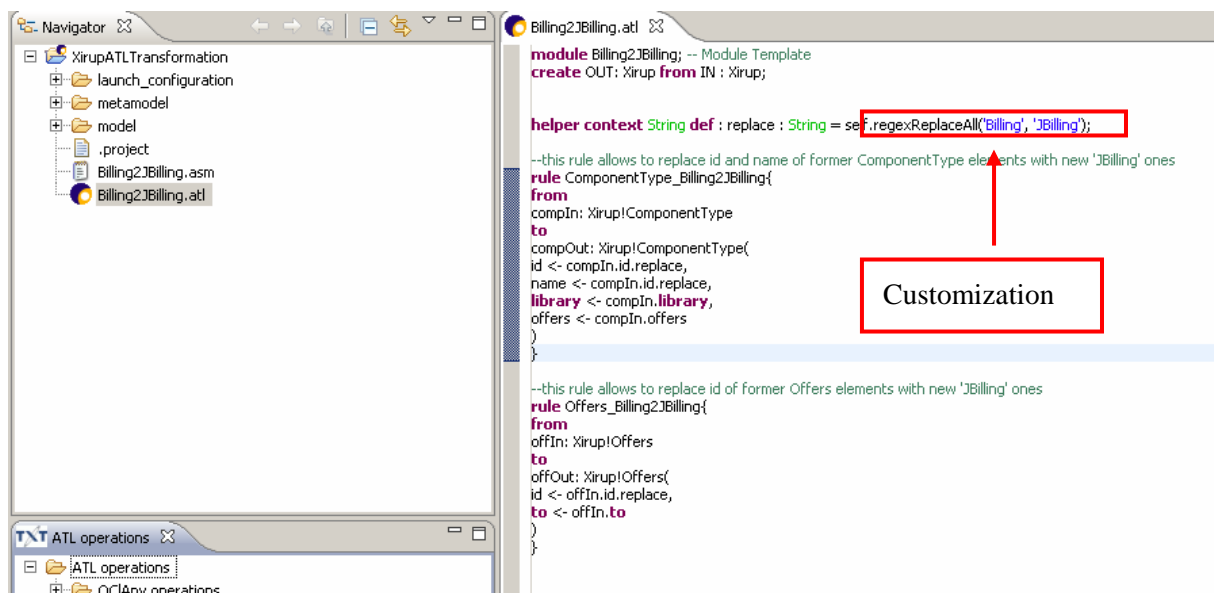


Figure 33: Operation customized by the user

## 4.4 Saving

Every time a transformation is saved, the ATL Builder dumps ATL Virtual Machine instructions to a \*.asm file which is then executed when running the transformation itself.

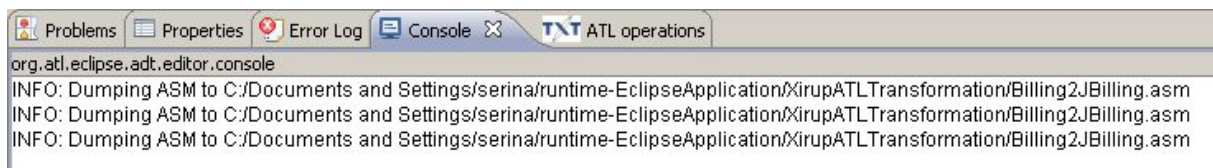


Figure 34: Dumping ASM

If dumping has no success, the Console shows all errors occurred and why new ATL Virtual Machine instructions have not been generated (Figure 35).

It must be highlighted that at present, ASM files can be automatically created only if corresponding ATL files are stored inside ATL projects.

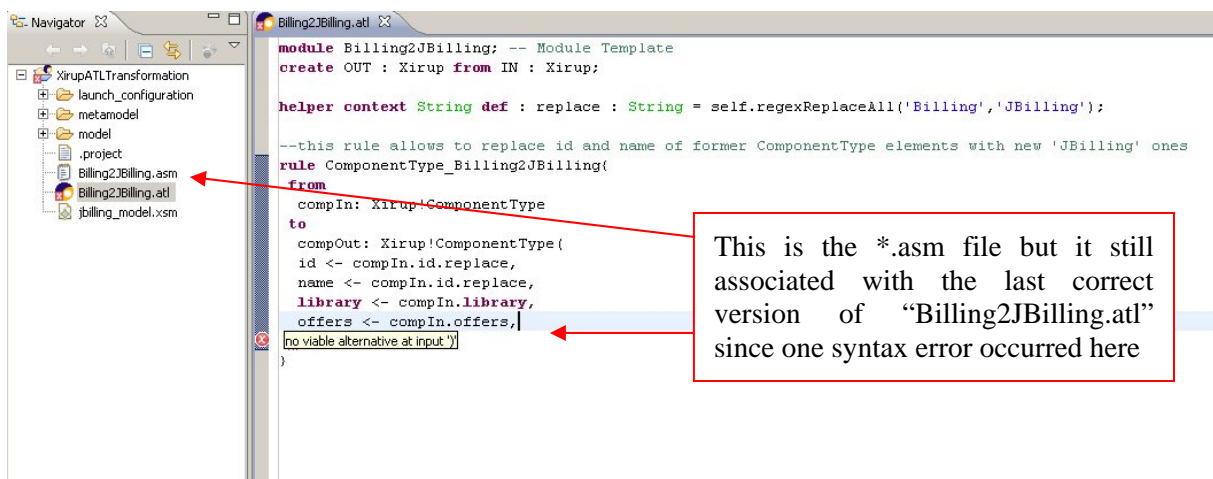
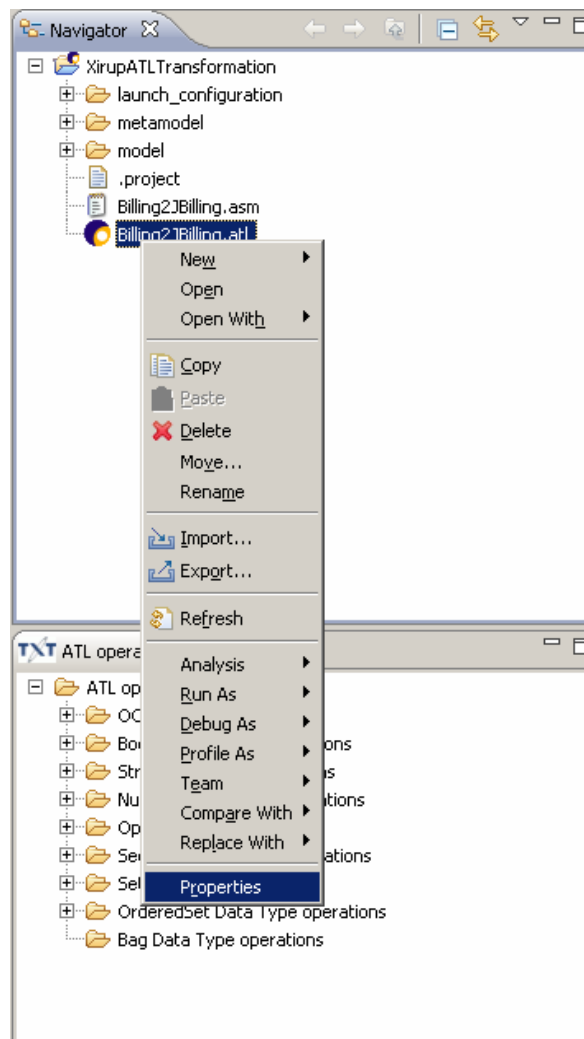


Figure 35: Errors occurred

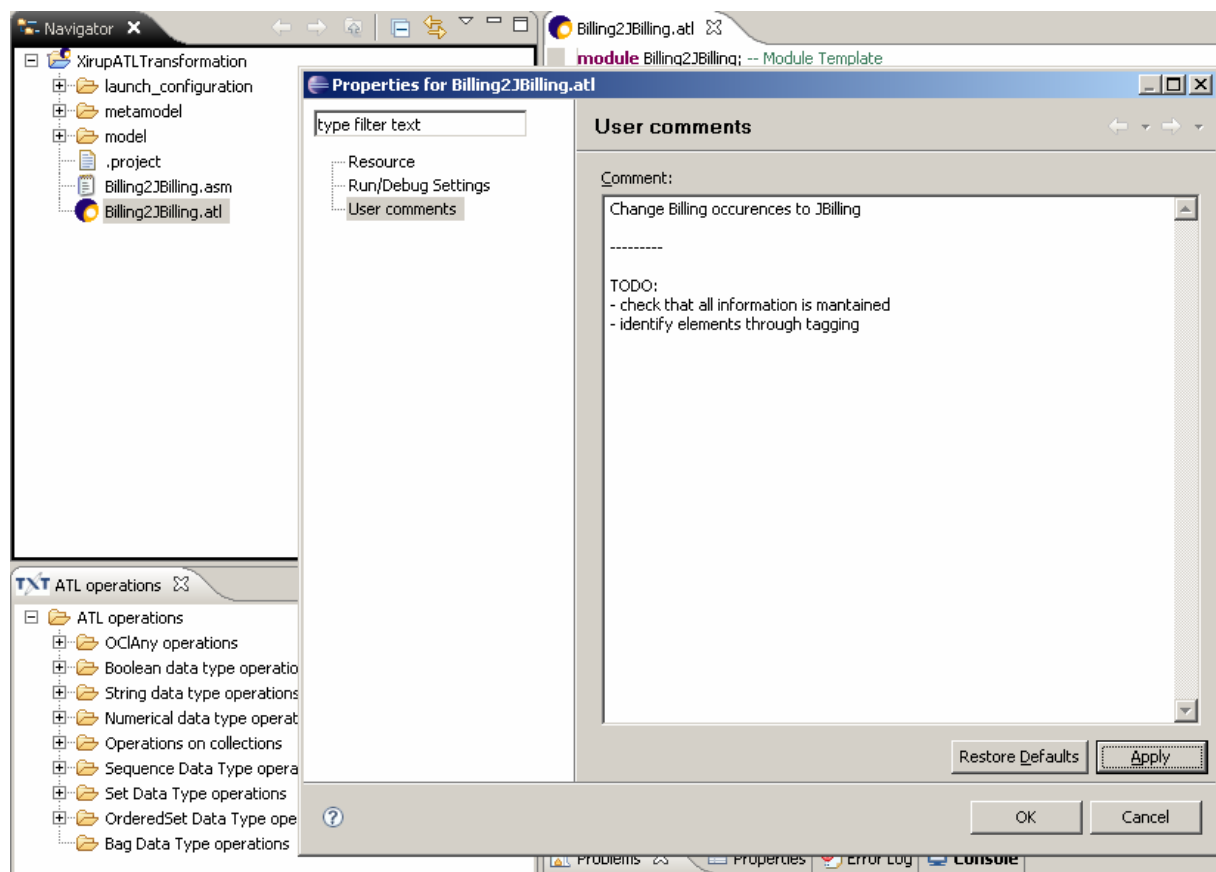
## 4.5 Commenting a transformation

Since it could be useful adding some extra information to each ATL file that has been created and edited, a user can add personal comments by means of a special property page. The content he/she writes is then attached in a persistent way to the file itself.



**Figure 36: Accessing to file properties**

A property page is thus displayed and the user can comment the transformation to better identify its contents.

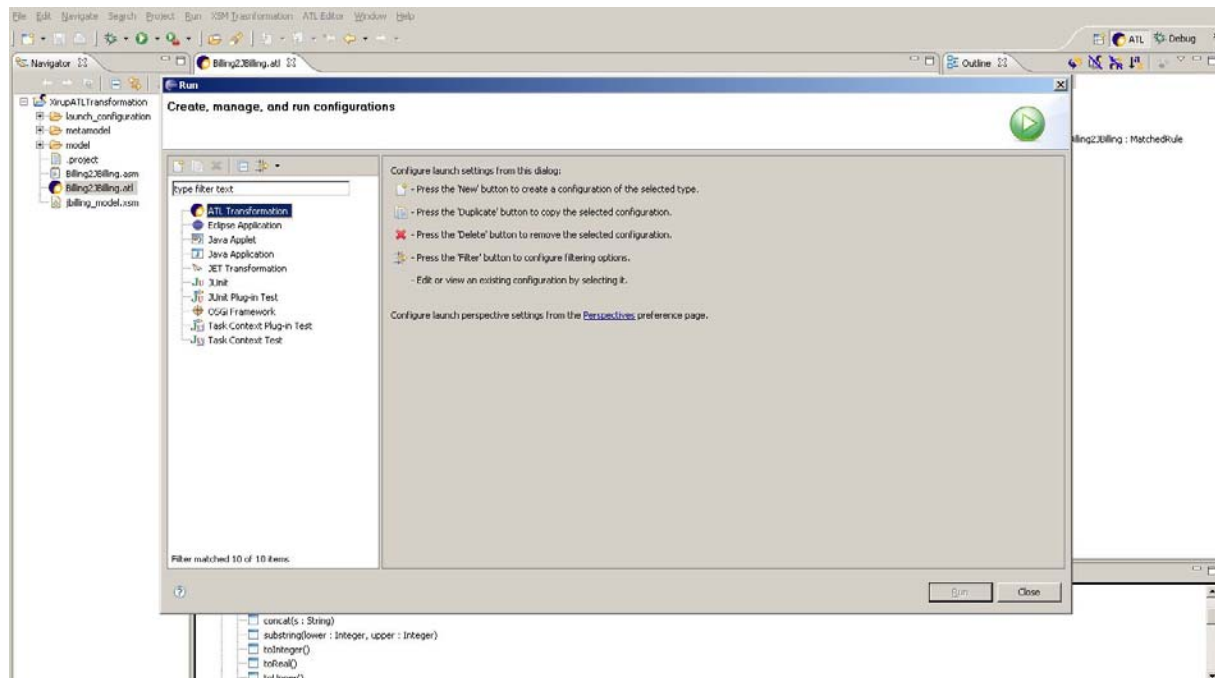


**Figure 37: Commenting transformation file**

## 4.6 Running

In order to start the execution of a transformation the user has to set and launch an ATL launch configuration which represents the actual way to prepare and glue together all the resources involved in the execution of the transformation itself.

By clicking on Run->Open Run Dialog a Run box appears showing different types of editable launch configurations (Figure 38).

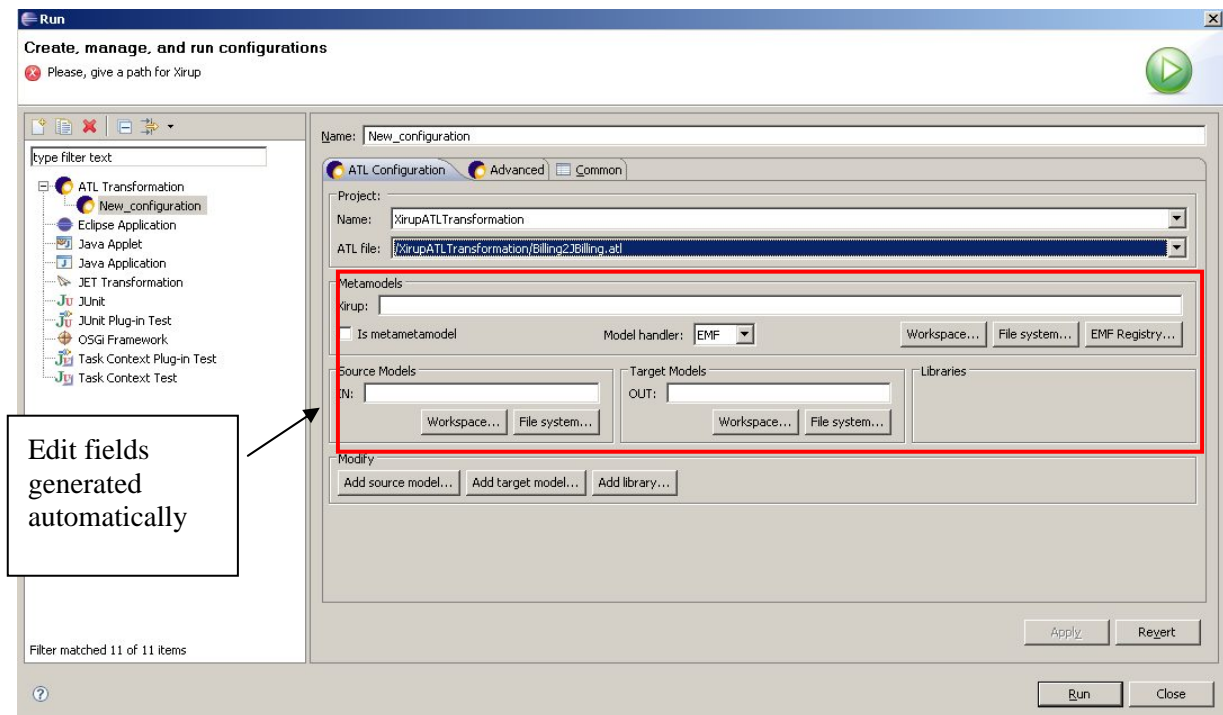


**Figure 38: Run dialog**

By double-clicking on “ATL Transformation”, a launch configuration edit panel shows: here all resources can be loaded and running parameters can be properly configured.

First of all, a transformation container (ATL project) and a transformation file have to be chosen.

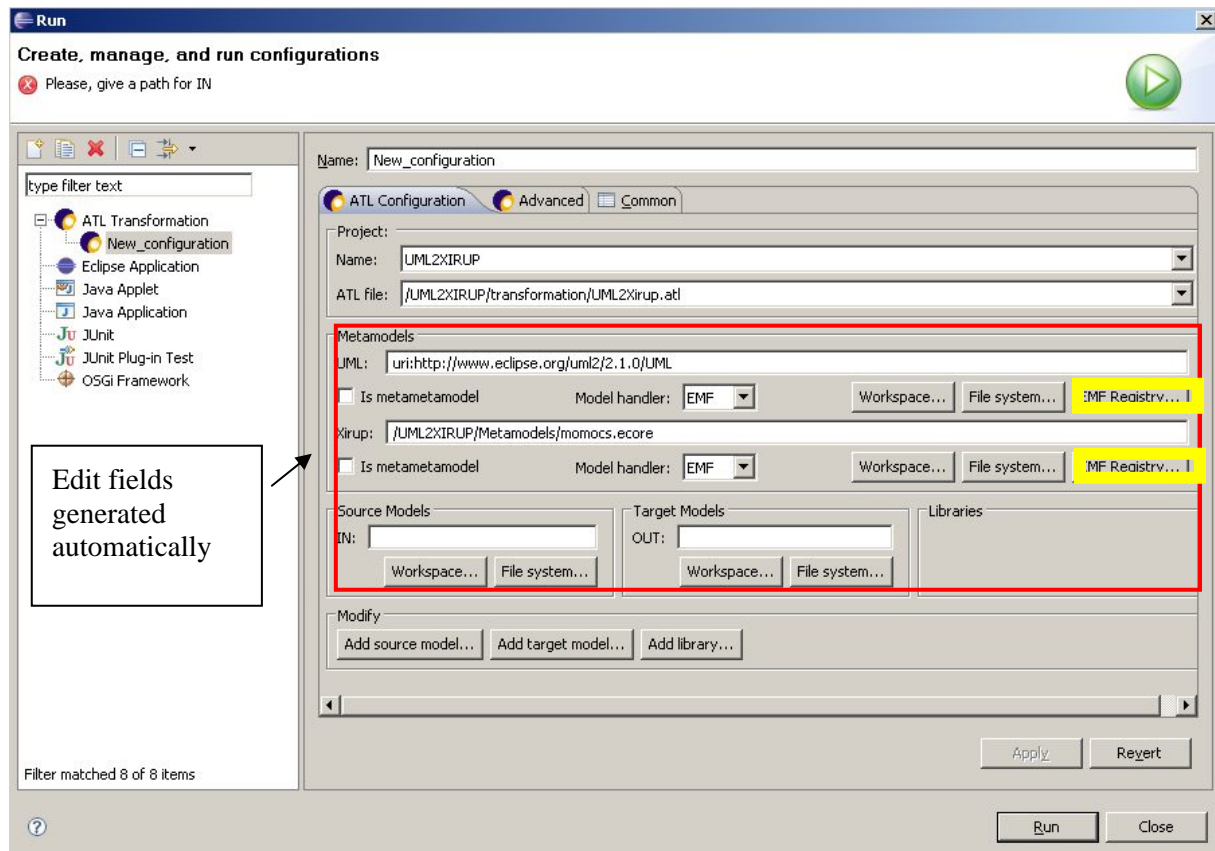
This way, by reading metamodel and model labels contained into the transformation file, ATL knows how many metamodels and models have to be loaded. Figure 39: Loading shows an example of this procedure.



**Figure 39: Loading parameters**

The “Billing2JBilling” transformation is a Xirup2Xirup transformation, thus it requires only one metamodel to be loaded, and it takes one model as input in order to generate one output model.

In case, for example, of an UML2Xirup transformation (thus working with two metamodels, namely UML and Xirup), the launch configuration dialog box will prompt two metamodel input edit boxes as displayed in Figure 40: load UML2Xirup transformation.



**Figure 40: load UML2Xirup transformation**

Metamodels can be loaded from workspace, file system or from EMF Registry (see yellow highlighted buttons) provided that they have been previously registered in the EMF registry itself. Usually, metamodels are serialized and stored as *Ecore* files [ECORE].

## 4.7 ATLforXirup Library

This sections depicts the work done to bridge Xirup concepts (intended as methodology and metamodel) and ATL.

#### 4.7.1 Xirup2Xirup

At present, more than one ATL technique exists in order to transform a source model into a target model and, additionally, two ATL compilers can be used to perform a transformation (see [ATL\_DOC] for references): this means different possibilities with different pros and cons to pursue the same task.

That said, TT must give to users the chance to choose among these several possibilities and select the best solution that suit to a particular scenario.

Basically, when creating and defining a transformation, a user can proceed in two ways:

1. declare each rule so that the ATL engine allocates only those model elements that are explicitly mapped by a rule (default mode)
2. refine a model so that the ATL lets the developer map only model elements he/she is interested in and automatically allocates all other elements that have not been explicitly managed by user rules (refining mode)

According to scenarios, (1) or (2) can result more suitable each other.

It should be clear that with (1), the developer has a full control over the whole transformation process, thus he/she must explicitly manage every model element in order to keep the source model consistent with the target model: not doing that would mean generating only those elements that have been specified in user rules, with an evident loss of information.

This leads to the concept of “**Copy transformation**”: a copy transformation explicitly manages, thus generates, all model elements and permits the user to focus only on those that have to be changed for modernization purposes (this is strictly correlated to the so called technique “superimposition” [ATL\_DOC]).



On the other side, (2) implements a similar approach as (1) but in an automatic way since the ATL engine does the “copy” in place of the user. Note that, due to current execution semantics of the refining mode, some specific precautions still have to be taken by developers. It may be required, for example, when designing an ATL module in refining mode, to specify additional explicit rules in order to make sure that all source model elements are transformed into their corresponding target model elements.

However, it's not the purpose of this section describing the different ATL execution modes, for further information please refer to [ATL\_DOC] or “Annexes: ATL Execution modes”

The brief description above, aims instead at letting the reader understand that, according to the scenario, type of elements to be managed and user preferences, (1) and (2) can be preferred to each other. If (2) could be easily managed by developers but it's not flexible, (1) should require a Xirup copy transformation to be defined, but it warrants a complete control over the transformation process.

Giving the amount of Xirup metamodel elements, the work done for TT also includes that copy transformation, preventing end-users from defining it by themselves.

Additionally, the ATL2006 compiler introduces new functionalities (such as rule extension mechanism) but it still doesn't support some features, for example the refining mode itself.

As a consequence, two Xirup copy transformations have been defined: one for ATL2004 compiler and one for ATL2006 compiler.

These two transformations are embedded in TT and can be always retrieved by means of proper wizards. They can be accessed by selecting File->New->XirupCopy 2004/2006 or File->New->Other->ATL4Xirup category.

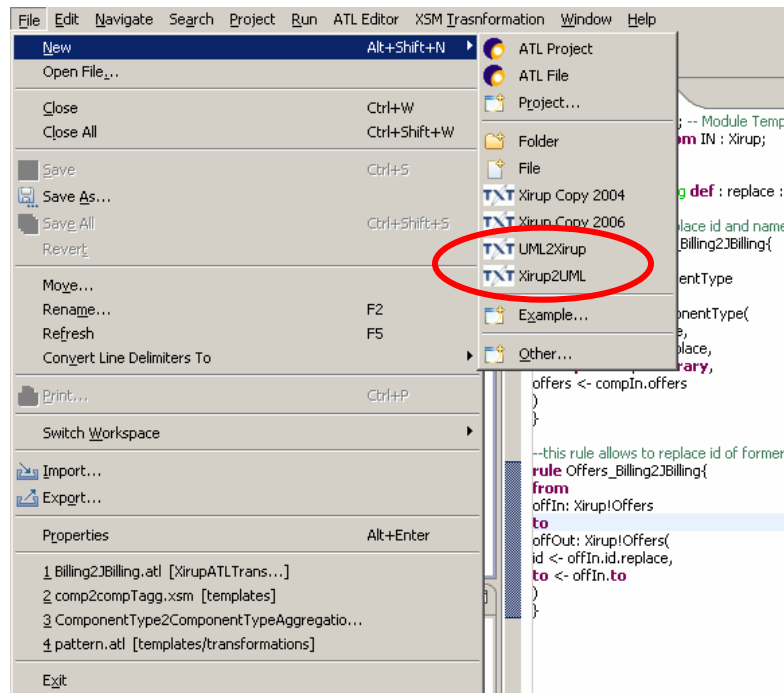


Figure 41: Get access to XirupCopy wizards (1)

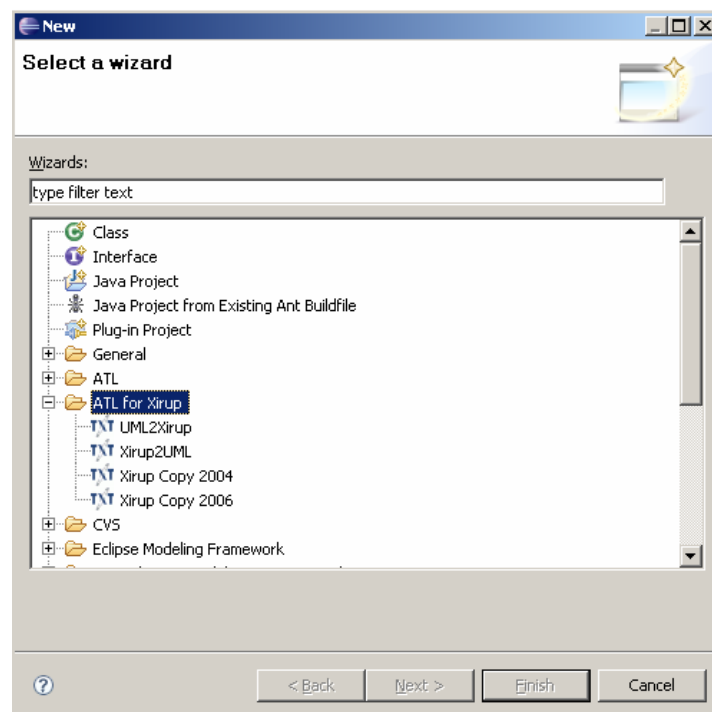
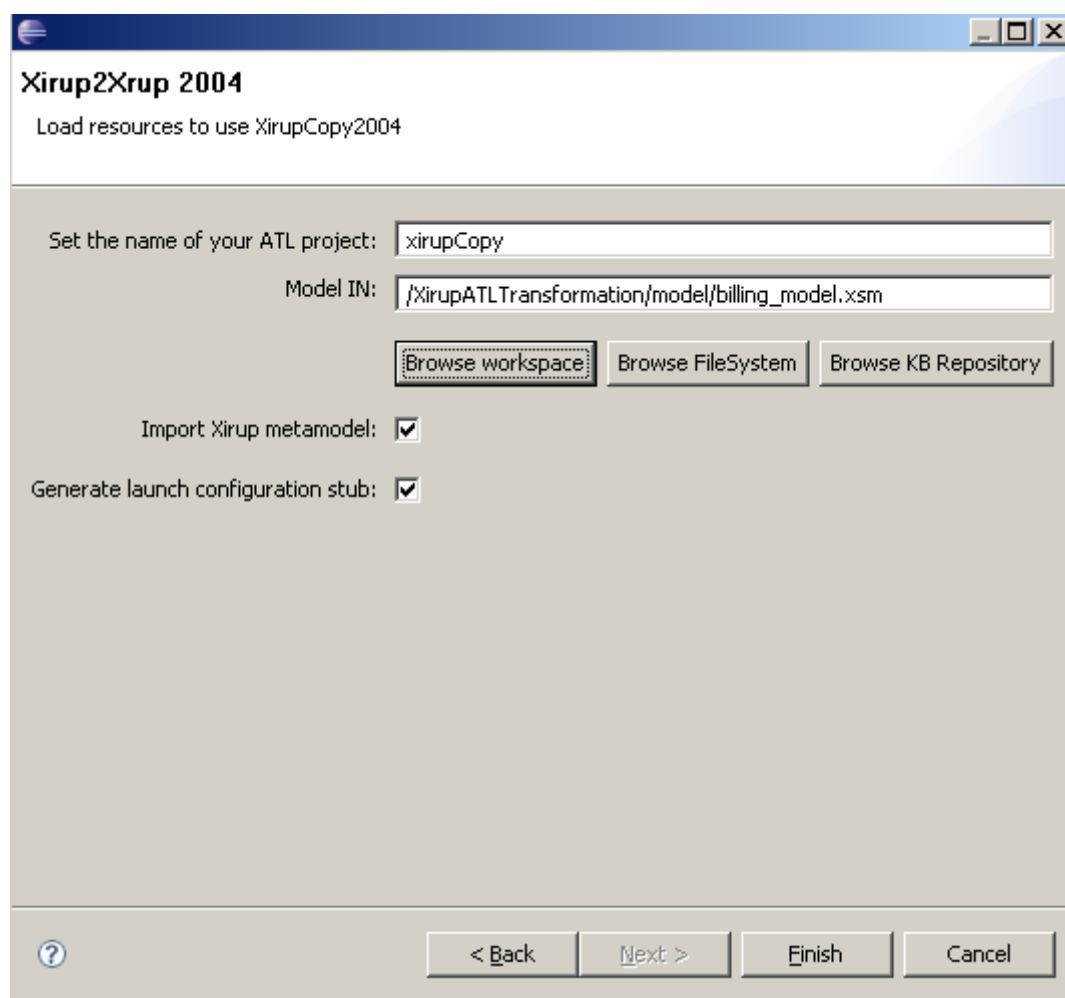


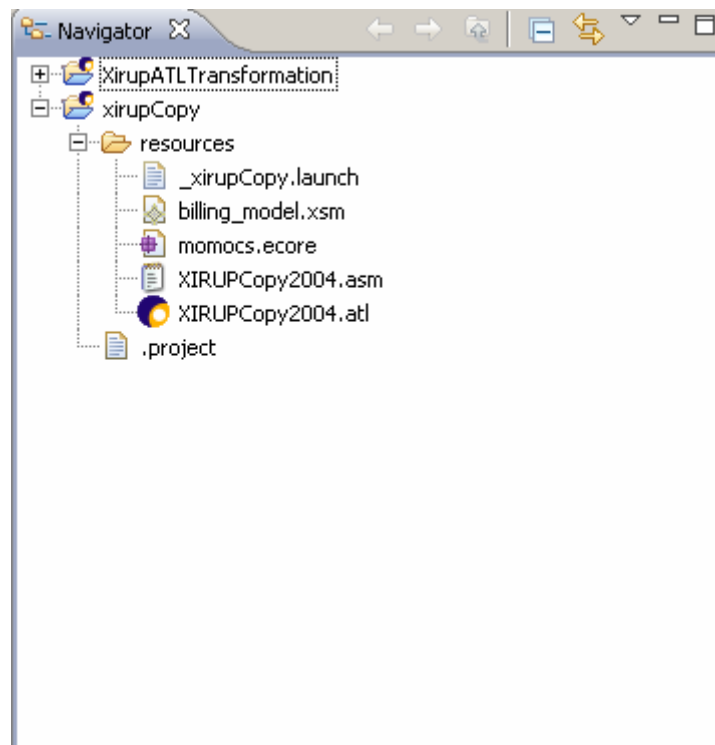
Figure 42: Get access to XirupCopy wizards (2)

- importing an existing Xirup model
- importing the Xirup metamodel ecore file
- generating an ATL launch configuration stub





**Figure 43: XirupCopy 2004 (for ATL compiler 2004) wizard**




What Figure 44 displays is the Navigator View after all the resources have been created/imported into the proper ATL project.



**Figure 44: Navigator view**

The following table gives a brief explanation of these resources.

Icon	Name	Type
	<i>_xirupCopy.launch</i>	ATL launch configuration stub generated automatically by TT to run the transformation (the name of this file derives from the name of the containing project)
	billing_model.xsm	Input XSM to be modernized that has been

		imported
	momocs.ecore	Xirup metamodel file
	XirupCopy2004.asm	Execution file generated automatically. Users do not have to worry about this file
	XirupCopy2004.atl	ATL transformation file containing all rules to copy source elements to target elements

**Table 1: Resources created after using XirupCopy2004 wizard**

XirupCopy2004 transformation **consists of more than forty rules intended to copy all the information from source to target models**: at this point, the user can directly modify that transformation by creating, deleting and changing rules of his/her interest in order to generate a (partly or fully) modernized output model without worrying about the rest, namely model elements that must be kept unchanged.

If the result is satisfactory, the modified XirupCopy transformation (actually no more a “XirupCopy”) can be stored to the KB Repository: whenever the user will need a new XirupCopy transformation, a new wizard can be used.

In addition to a direct modification, a new transformation can be created (see section 4.2) and “superimposed” to the XirupCopy transformation.

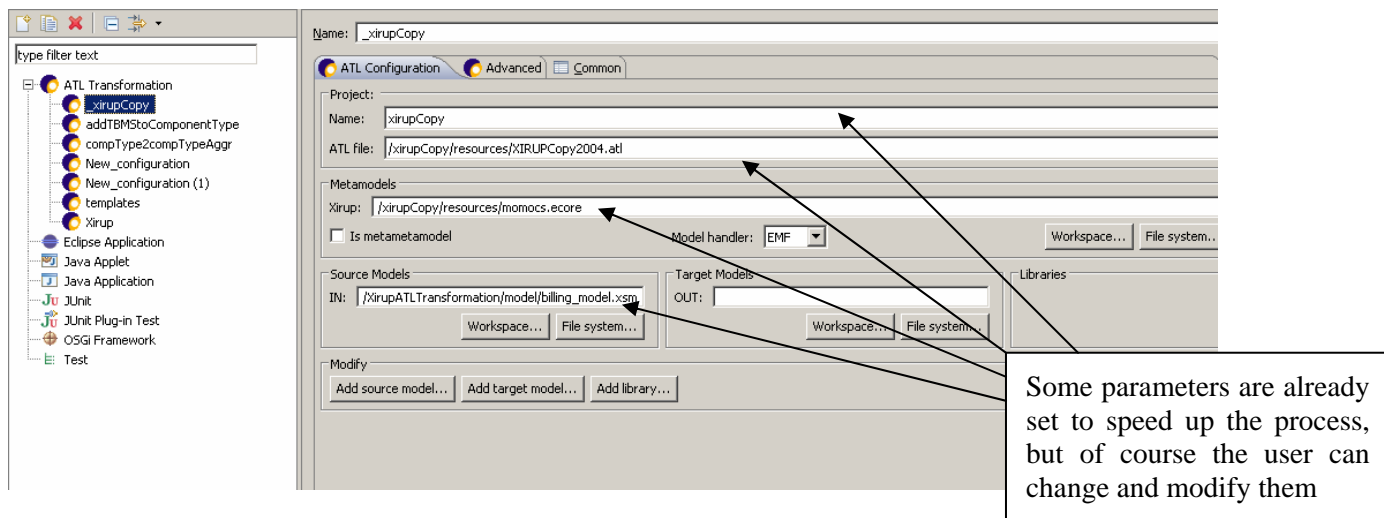
Superimposition means that “while ATL transformation modules and queries are normally run by themselves, that is one transformation module or query at a time, it is also possible to superimpose several transformation modules on top of each other. The end result is a transformation module that contains the union of all transformation rules and all helpers, where it is possible for a transformation module

to override rules and helpers from the transformation modules underneath” [ATL\_DOC].

Screencasts on these features have been provided and are available under SVN (see beginning of section 4 for references).

Resuming, the aim of XirupCopy and related wizards is having a ready-to-use tool for Xirup2Xirup modernization, allowing the user to focus only on the actual, real target of the modernization itself and not wasting time for out of scope activities.

After having imported the resources presented in Table 1, creating and modifying transformations according to his\her current needs, the user can run the transformation basing upon the launch configuration stub generated automatically by the wizard.



**Figure 45: Launch configuration stub**

XirupCopy 2006 wizard provides the same functionalities as XirupCopy 2004 one, but the embedded copy transformation is written to be executed with ATL compiler 2006.

For more information about core differences between those two approaches, please see section 6.3 or [ATL\_DOC].

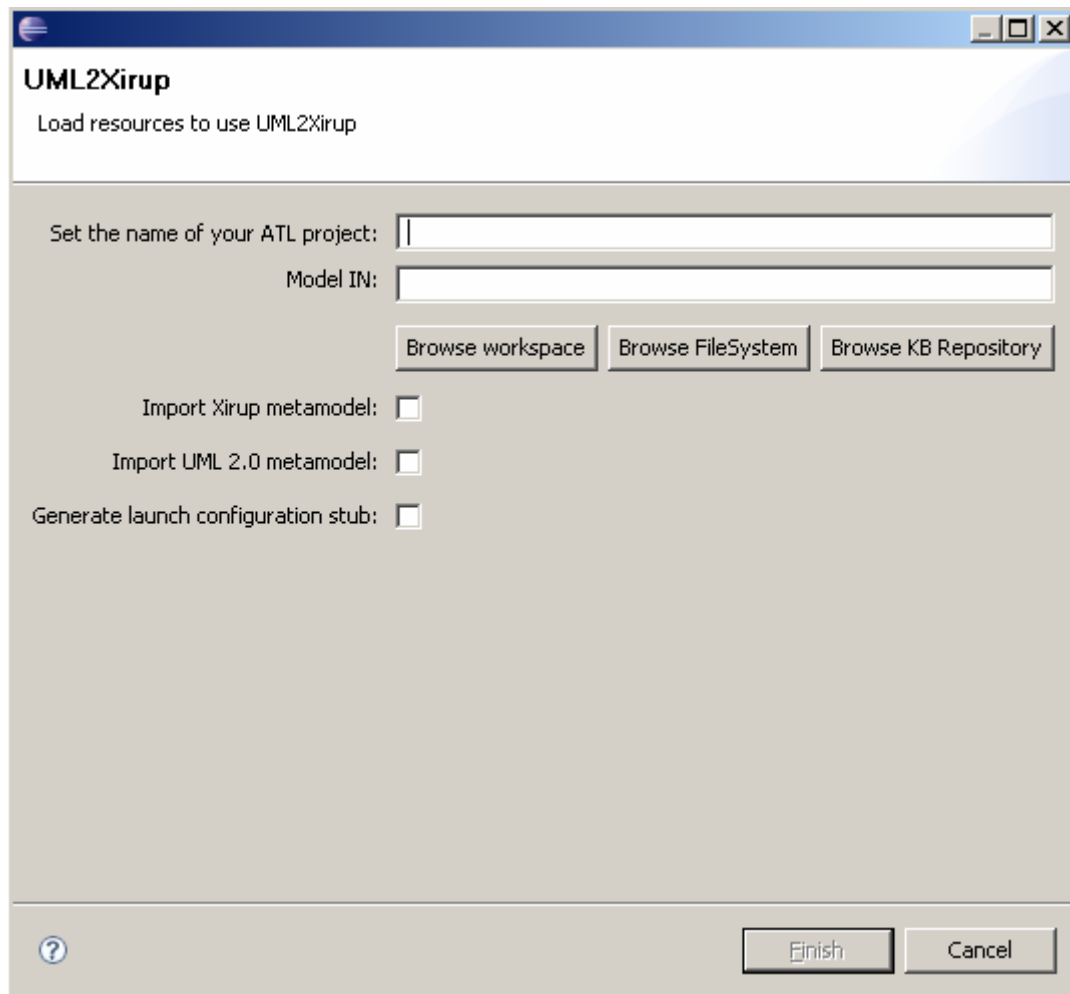
#### 4.7.2 UML2Xirup and Xirup2UML

Similarly to what presented in section 4.7.1, UML2Xirup and Xirup2UML wizards aim at giving help in getting Xirup models from UML models and vice versa.

The current supported UML format is EMF serializable, namely the same format which UML models created with Eclipse UML2Tools [UML2Tools] are based on.

The aim of these wizards is creating and importing a set of resources, first of all proper transformations, in order to let the user switch from UML models to Xirup models and vice versa.

Figure 46: shows the UML2Xirup wizard: the only difference from the one described in the previous section regards the possibility to import the UML metamodel file which is needed to run the converting transformation (thus UML2Xirup or Xirup2UML).

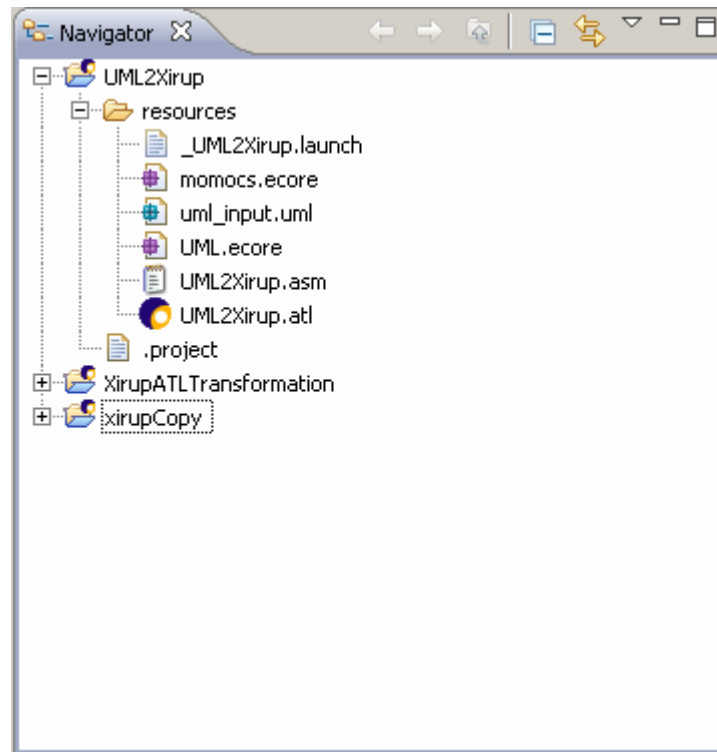


**Figure 46: UML2Xirup wizard**

After filling in all the fields of interest (the only mandatory one is project name), a set of resources are created, similarly to what happened for XirupCopy.





Figure 47: shows the Navigator view after using the wizard.







**Figure 47: Navigator view**

The following table gives a brief explanation of resources above.

Icon	Name	Type
	<i>_UML2Xirup.launch</i>	ATL launch configuration stub generated automatically by TT to run the transformation (the name of this file derives from the name of the containing project)
	uml_input.uml	Input UML model to be converted into a TBMS XSM
	momocs.ecore	Xirup metamodel file
	UML.ecore	UML metamodel file

	UML2Xirup.asm	Execution file generated automatically. Users do not have to worry about this file
	UML2Xirup.atl	ATL transformation file containing rules to get a XSM from a UML model

**Table 2: Resources created after using UML2Xirup wizard**

We do not present the Xirup2UML wizard since it's based on the same concepts that have already been depicted throughout this section.

It's mandatory mentioning that actual UML2Xirup and Xirup2UML transformations are not complete, but they are instead initial examples giving a first idea of what is going to be provided for sure in future: a set of tools to quickly switch from UML and Xirup.

Along with XirupCopy wizards, these functionalities are intended to cover the following process:



**Figure 48: From UML to Xirup and back again**

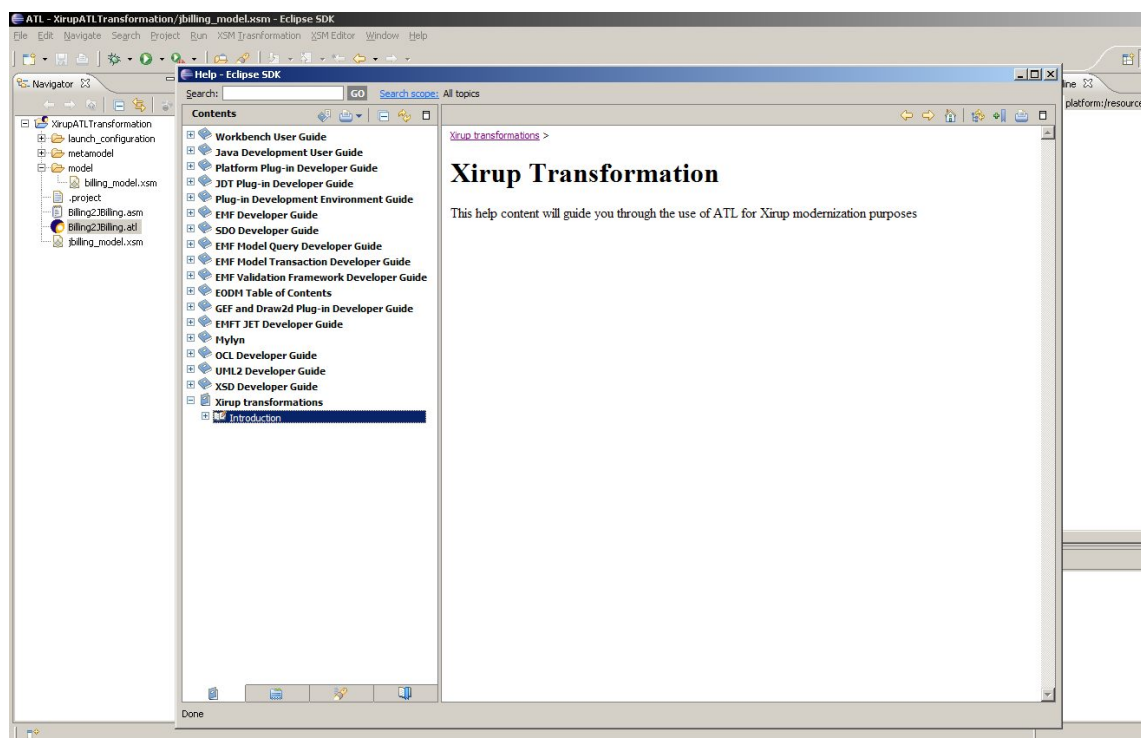
## 4.8 Help facilities guidelines

In addition to editing facilities and transformations from and to Xirup, TT intends to supply a general support to correctly use the tool.

Thus, we want to provide learning instruments such as:

- Help contents
- Cheat sheets

Help contents are going to be filled in while collecting end-users' feedback in order to build a meaningful documentation (screenshot available in Figure 49).



**Figure 49: Xirup transformation help contents**

A first cheat sheet, instead, has already been developed (see section 3.3).

Future cheat sheets can be developed according to actual user needs that come from testing and evaluation of TT.

Possible tutorial can concern, for instance, explanation of superimposition and refining techniques.

## 5 Troubleshooting

### 5.1 Known issues

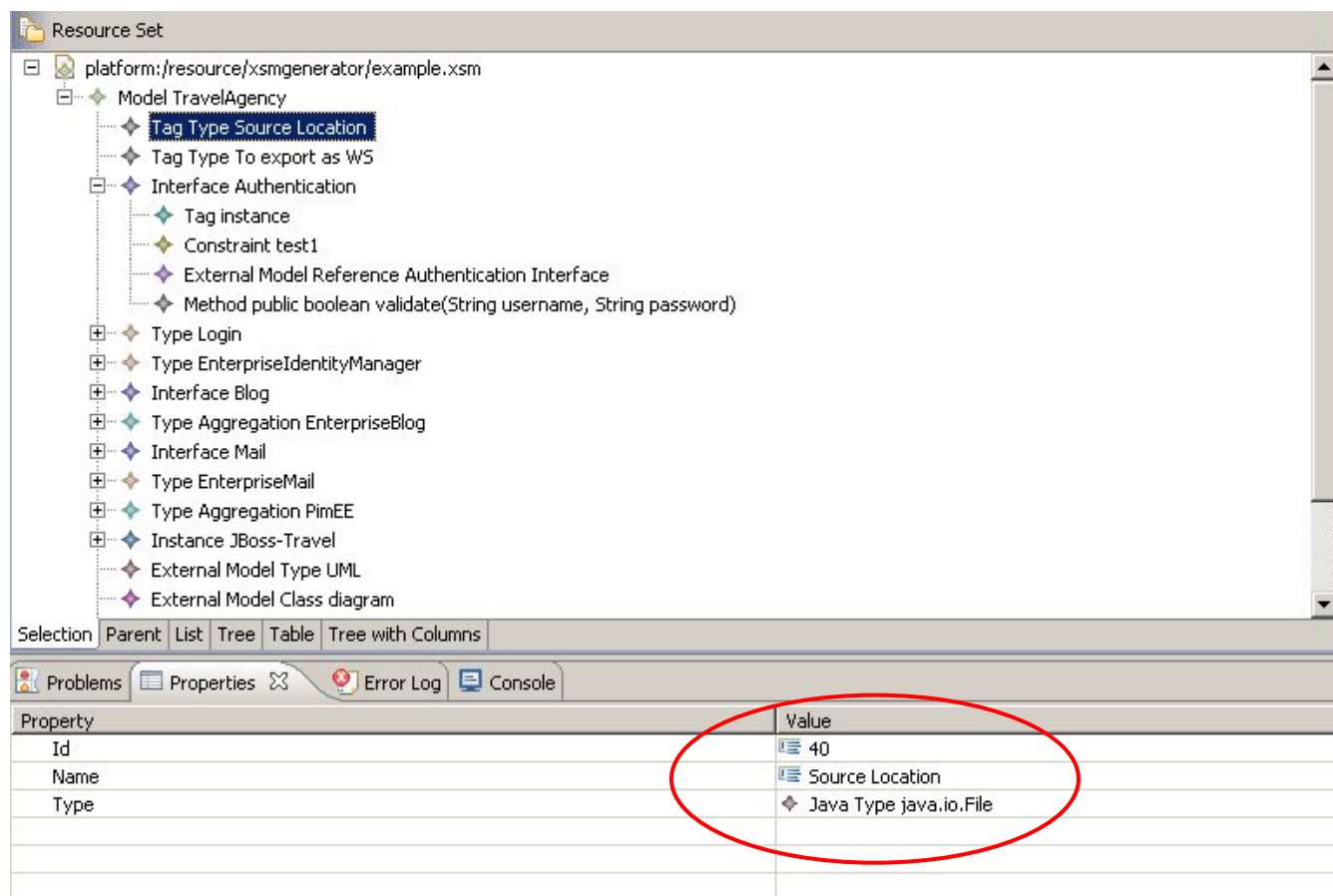
#### 5.1.1 EJavaClass type error

As far as we could see, ATL is not able to generate those target elements having a type which is related to non-primitive Java types such as Boolean, Float, etc. and `java.lang.String`. This issue may be better explained through an example.

Let's consider a *JavaType* element deriving from `java.io.File` and a *TagType* element with the following properties:

- Id = "40"
- Name = "Source Location"
- Type = *JavaType* `java.io.File`

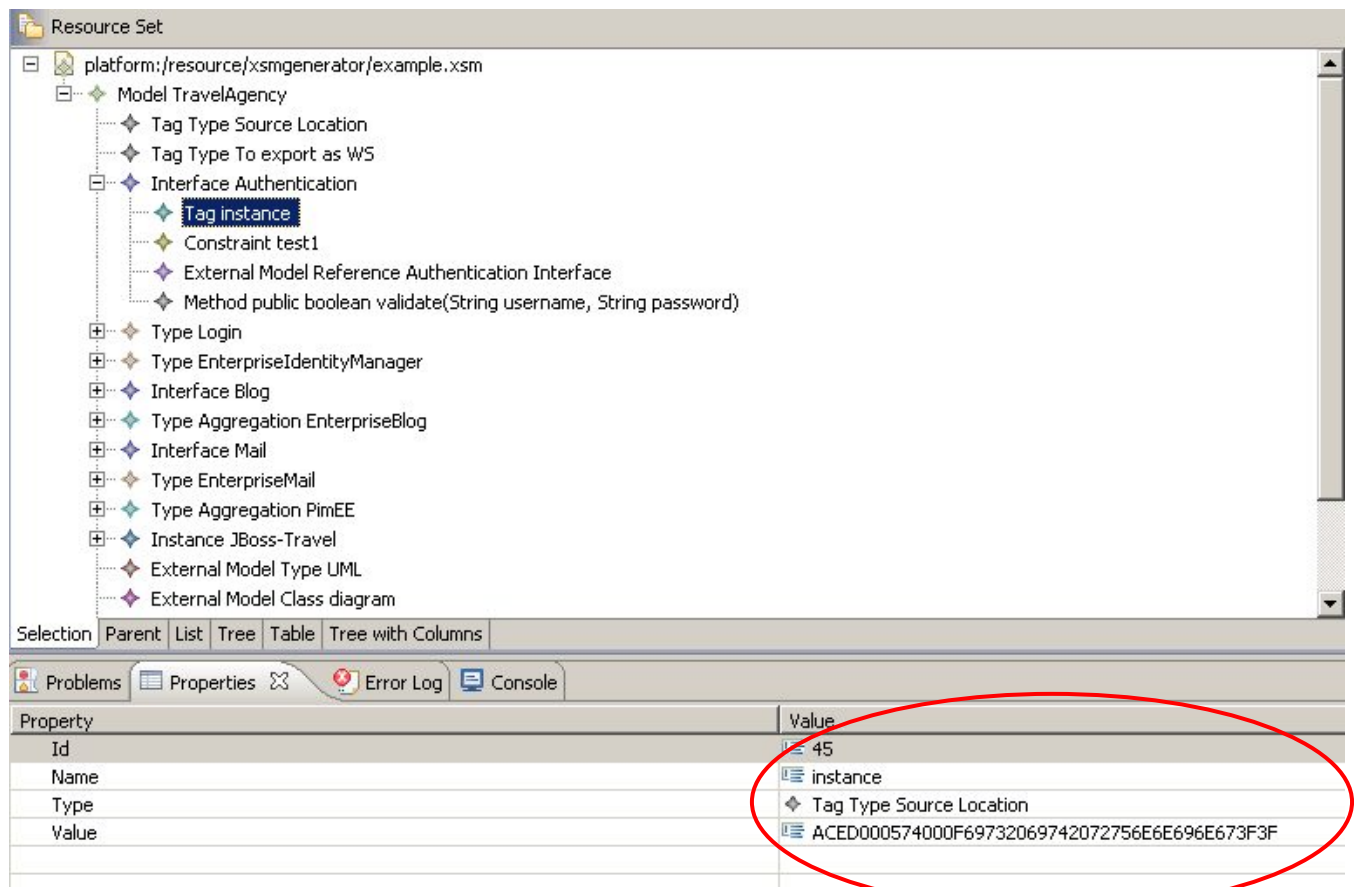
The screenshot in the next page shows this example:



**Figure 50: TagType element properties**

We have also a *Tag* element having these properties:

- Id = "45"
- Name = "Instance"
- Type = *TagType* Source Location
- Value = ACED000574000F69732069742072756E6E696E673F3F [This is due to EMF serialization, the model is actually generated by code]



**Figure 51: Tag element properties**

If we try to allocate the “Value” field of *Tag* (shown in the picture above) by means of any ATL transformation (tests have been done with XirupCopy transformations) the ATL Virtual machine return the following error:

```

GRAVE: ***** BEGIN Stack Trace
GRAVE:      message: ERROR: cannot convert d:\Login.java : class java.io.File from EMF.
GRAVE: A.main() : ??#24 null
GRAVE:      local variables = {self=XIRUPCopy2004 : ASModule}
GRAVE:      local stack = []
GRAVE: A.__exec__() : ??#78 null
GRAVE:      local variables = {e=TransientLink {rule = 'Tag', sourceElements = {s = IN!Login.java},
targetElements = {t = OUT!Login.java}, variables = {}}, self=XIRUPCopy2004 : ASModule}
GRAVE:      local stack = []
GRAVE: A.__applyTag(1 : NTransientLink;) : ??#64 129:15-129:22

```

```
GRAVE:      local variables = {t=OUT!Login.java, s=IN!Login.java, link=TransientLink {rule = 'Tag',
sourceElements = {s = IN!Login.java}, targetElements = {t = OUT!Login.java}, variables = {}},
self=XIRUPCopy2004 : ASModule}
GRAVE:      local stack = [OUT!Login.java]
GRAVE: ***** END Stack Trace
INFO: Execution terminated due to error (see launch configuration to allow continuation after errors).
GRAVE: ERROR: cannot convert d:\Login.java : class java.io.File from EMF.
java.lang.RuntimeException: ERROR: cannot convert d:\Login.java : class java.io.File from EMF.
    at org.eclipse.m2m.atl.engine.vm.SimpleDebugger.error(SimpleDebugger.java:185)
    at org.eclipse.m2m.atl.engine.vm.StackFrame.printStackTrace(StackFrame.java:85)
    at org.eclipse.m2m.atl.engine.vm.StackFrame.printStackTrace(StackFrame.java:81)
    at
org.eclipse.m2m.atl.drivers.emf4atl.ASMEMFModelElement.emf2ASM(ASMEMFModelElement.java:186)
    at org.eclipse.m2m.atl.drivers.emf4atl.ASMEMFModelElement.get(ASMEMFModelElement.java:134)
    at org.eclipse.m2m.atl.engine.vm.ASMOperation.realExec(ASMOOperation.java:288)
    at org.eclipse.m2m.atl.engine.vm.ASMOperation.exec(ASMOOperation.java:161)
    at org.eclipse.m2m.atl.engine.vm.nativelib.ASMOclAny.invoke(ASMOclAny.java:133)
    at org.eclipse.m2m.atl.engine.vm.nativelib.ASMOclAny.invoke(ASMOclAny.java:91)
    at org.eclipse.m2m.atl.engine.vm.ASMOperation.realExec(ASMOOperation.java:230)
    at org.eclipse.m2m.atl.engine.vm.ASMOperation.realExec(ASMOOperation.java:325)
    at org.eclipse.m2m.atl.engine.vm.ASMOperation.exec(ASMOOperation.java:161)
    at org.eclipse.m2m.atl.engine.vm.nativelib.ASMOclAny.invoke(ASMOclAny.java:133)
    at org.eclipse.m2m.atl.engine.vm.nativelib.ASMOclAny.invoke(ASMOclAny.java:91)
    at org.eclipse.m2m.atl.engine.vm.ASMOperation.realExec(ASMOOperation.java:230)
    at org.eclipse.m2m.atl.engine.vm.ASMOperation.exec(ASMOOperation.java:161)
    at org.eclipse.m2m.atl.engine.vm.ASMInterpreter.<init>(ASMInterpreter.java:289)
    at org.eclipse.m2m.atl.engine.AtlLauncher.launch(AtlLauncher.java:155)
    at org.eclipse.m2m.atl.engine.AtlLauncher.launch(AtlLauncher.java:105)
    at org.eclipse.m2m.atl.engine.AtlLauncher.launch(AtlLauncher.java:81)
    at org.eclipse.m2m.atl.adt.launching.AtlRegularVM.runAtlLauncher(AtlRegularVM.java:351)
    at org.eclipse.m2m.atl.adt.launching.AtlRegularVM.runAtlLauncher(AtlRegularVM.java:453)
    at org.eclipse.m2m.atl.adt.launching.AtlRegularVM.launch(AtlRegularVM.java:425)
    at
org.eclipse.m2m.atl.adt.launching.AtlLaunchConfigurationDelegate.launch(AtlLaunchConfigurationDelegate.java:
35)
    at org.eclipse.debug.internal.core.LaunchConfiguration.launch(LaunchConfiguration.java:766)
    at org.eclipse.debug.internal.core.LaunchConfiguration.launch(LaunchConfiguration.java:608)
    at org.eclipse.debug.internal.ui.DebugUIPlugin.buildAndLaunch(DebugUIPlugin.java:899)
    at org.eclipse.debug.internal.ui.DebugUIPlugin$7.run(DebugUIPlugin.java:1102)
    at org.eclipse.core.internal.jobs.Worker.run(Worker.java:55)
```

We suppose this is due to the fact that ATL has not been designed to manage complex Java types as `java.io.File` and similar.

At present, we ignore values coming from those kind of types and we take into account only simple Java types.

Within the TT implementation activities we have planned to investigate on this issue in order to find a more structured solution.



## 5.2 Bug Reporting

Please use the project Bugzilla for bug reporting at ['http://demos.txt.it/momocs\\_bugzilla'](http://demos.txt.it/momocs_bugzilla)

.

## 5.3 FAQ

**Q:** Do users have to write ATL rules on their own in order to modernize Xirup models?

**A:** TT simplifies the editing process providing a set of facilities such as syntax templates and the operations view; however, users still have to write ATL rules or, at least, there should be a person managing TT and creating proper transformations to be stored in the KB and reused by all other MOMOCs users.

However, we're going to build a set of "transformation patterns" and related user-interfaces to provide parametric transformations that target common modernization issues.

**Q:** How can I customize ongoing parametric transformations?

**A:** Xirup metamodel provides a tagging mechanism: by using tags, model elements can be identified and updated according to the rules specified in the transformation.

**Q:** Why are XirupCopy, UML2Xirup and Xirup2UML transformations TT-embedded instead of being simply stored into the KB?

A: We can say that they are some kind of “rail lines” addressing and covering the major steps of the Xirup modernization process.

They are the actual foundations which transformations that really perform the modernization and effectively need to be stored into the KB can be based upon.

## 6 Appendixes

### 6.1 Ongoing features

Several ongoing features are planned to be implemented from the next TT release. Below, a list of bullets shows the major ones.

- Editing facilities improving with little graphical support
- Creation of further cheat-sheets covering major learning issues (e.g. ATL superimposition and refining techniques, but this also depends on end-users needs and evaluation)
- Help contents documentation filling
- Creation of transformation patterns and related user interfaces

### 6.2 ATL Execution modes

From [ATL\_DOC]:

#### Default execution mode:

The execution of an ATL module is organized into three successive phases: a module initialization phase, a matching phase of the source model elements, and a target model elements initialization phase.

The module initialization step corresponds to the first phase of the execution of an ATL module. In this phase, the attributes defined in the context of the transformation module are initialized. Note that the initialization of these module attributes may make use of attributes that are defined in the context of source model elements. This implies these new attributes to be also initialized during the module initialization phase.

During the source model elements matching phase, the matching condition of the declared matched rules are tested with the model elements of the module source models. When the matching condition of a matched rule is fulfilled, the ATL engine allocates the set of target model elements that correspond to the target pattern elements declared in the rule. Note that, at this stage, the target model elements are simply allocated: they are initialized during the target model elements initialization phase. The last phase of the execution of an ATL module corresponds to the initialization of the target model elements that have been generated during the previous step. At this stage, each allocated target model element is initialized by executing the code of the bindings that are associated with the target pattern element the element comes from. The imperative code section that can be specified in the scope of a matched rule is executed once the rule initialization step has completed. This imperative code can trigger the execution of some of the called rules that have been defined in the scope of the ATL module

### **Refining execution mode:**

The refining execution mode introduces specific semantics for the implicit generation of copied model elements.

An ATL module executed in refining mode follows the three successive phases of the default execution mode. The execution of the first phase, the module initialization phase, remains unchanged compared to the default execution mode. During the source model elements matching phase, the ATL engine only evaluates the matching conditions of the explicitly specified matched rules. This implies that, at this stage, the only target model elements that are allocated are those that are generated by these explicit transformations rules.

The differences with the default execution mode appear during the execution of the initialization phase of the target model elements. In refining mode, this phase has to deal with the initialization of the explicitly generated target model elements, but also with the allocation and the initialization of the target model elements that are implicitly generated.

For this purpose, each time an already allocated target model element is initialized with a reference to a non-allocated model element, the ATL engine allocates and initializes this new target model element. If the newly created model element also refers to another non-allocated model element, this process is repeated recursively.

Note that with the described semantics, no target model element will be generated for a source model element that is neither matched by an explicit rule, nor referred, directly or indirectly, by an explicitly generated target model element

## 6.3 XirupCopy transformations

This section provides some screenshots and examples showing the main characteristics of the two XirupCopy transformations that have been created and supplied along with TT.

The concept which they lay on is the same, but they are defined for two different compilers, ATL 2004 and ATL 2006.

Section 6.3.3 also gives an example of porting activities for XirupCopy 2004 with different Xirup metamodel releases.

### 6.3.1 XirupCopy2004

XirupCopy2004 is explicitly defined to be executed by ATL compiler 2004 and, as already described, copies all elements from the source model to the target model.

In the next page, some indicative screenshots are displayed.

```
module XIRUPCopy2004; -- Module Template
create OUT : Xirup from IN : Xirup ;

--XSM SECTION

--ExternalModel element
rule ExternalModel {
  from
    s : Xirup!ExternalModel
  to
    t : Xirup!ExternalModel {
      id <- s.id,
      tag <- s.tag,
      metric <- s.metric,
      externalModelReference <- s.externalModelReference,
      name <- s.name,
      type <- s.type,
      document <- s.document
    }
}

--ExternalModelType element
rule ExternalModelType {
  from
    s : Xirup!ExternalModelType
  to
    t : Xirup!ExternalModelType {

      id <- s.id,
      tag <- s.tag,
      metric <- s.metric,
      externalModelReference <- s.externalModelReference,
      name <- s.name,
      description <- s.description
    }
}

--ExternalModelReference element
rule ExternalModelReference {
  from
    s : Xirup!ExternalModelReference
  to
    t : Xirup!ExternalModelReference {
      id <- s.id,
      tag <- s.tag,
      metric <- s.metric,
      externalModelReference <- s.externalModelReference,
      name <- s.name,
      description <- s.description
    }
}
```

Figure 52: XirupCopy2004 (1)

```
rule CompositeConstraintParameter{
  from
    s : Xirup!CompositeConstraintParameter
  to
    t : Xirup!CompositeConstraintParameter {
      id <- s.id,
      tag <- s.tag,
      metric <- s.metric,
      externalModelReference <- s.externalModelReference,
      name <- s.name,
      description <- s.description,
      type <- s.type,
      variable <- s.variable,
      expression <-s.expression
    }
}

rule ConstraintTemplate {
  from
    s : Xirup!ConstraintTemplate
  to
    t : Xirup!ConstraintTemplate {
      id <- s.id,
      tag <- s.tag,
      metric <- s.metric,
      name <- s.name,
      description <-s.description,
      language <- s.language,
      expression <- s.expression,
      severity <- s.severity,
      constrainttemplate <- s.constrainttemplate,
      constraint <- s.constraint,
      constraintparameter <- s.constraintparameter
    }
}

--COMPONENT SECTION
```

Figure 53: XirupCopy2004 (2)



### 6.3.2 XirupCopy2006

This transformation performs the same mappings as XirupCopy2004, but it's designed to exploit an interesting new ATL2006 feature, called "rule inheritance". This approach was followed since it gives a more coherent vision of the transformation if we take into account the metamodel structure. For example, it avoids re-mapping elements once they have been already matched by a parent rule. A small example is shown below:

```
abstract rule BasicElement{  
    from  
        s : Xirup!BasicElement  
    to  
        t : Xirup!BasicElement(  
            id <- s.id,  
            tag <- s.tag,  
            metric <- s.metric  
        )  
}  
  
rule Model extends BasicElement{  
    from  
        s: Xirup!Model  
    to  
        t: Xirup!Model(  
            name <- s.name,  
            description <- s.description,  
            element <- s.element,  
            tagtype <-s.tagtype  
        )  
}
```

```
    )  
}  
  
rule Method extends BasicElement {  
    from  
        s : Xirup!Method  
    to  
        t : Xirup!Method (  
            name <- s.name  
        )  
}
```

Of course Method class extends BasicElement class, that's why the rule inheritance works. It's clear that this approach easily lead to an elegant and straightforward way to express the mappings.

With ATL2004 Compiler, the three rules above would be:

```
rule Model {  
    from  
        s: Xirup!Model  
    to  
        t: Xirup!Model(  
            id <- s.id,  
            tag <- s.tag,  
            metric <- s.metric,  
            name <- s.name,  
            description <- s.description,  
            element <- s.element,
```

```
        tagtype <-s.tagtype
    )
}

rule Method{
    from
        s : Xirup!Method
    to
        t : Xirup!Method (
            id <- s.id,
            tag <- s.tag,
            metric <- s.metric,
            name <- s.name
        )
}
```

As it's clearly observable, the mappings expressed in bold character (coming from the abstract BasicElement) must be explicitly allocated in each rule that manage all BasicElement child elements (in this example Method, Model).

Abstract rules defined within XirupCopy2006 are:

- BasicElement
- Type
- Relation
- InternalElement
- Element
- Instance
- AbstractConstraint

- ConstraintParameter
- InfoInstance
- InfoType
- AbstractActivitivy

Next pages show some useful screenshots.

```
-- @atlcompiler atl2006
module XIRUPCopy2006; -- Module Template
create OUT : Xirup from IN : Xirup ;

--ABSTRACT RULE SECTION:
--Please notice that a full rule inheritance is not possible since ATL doesn't
--support multiple inheritance.

-- Basic element abstract rule
abstract rule BasicElement{
  from
    s : Xirup!BasicElement
  to
    t : Xirup!BasicElement {
      id <- s.id,
      tag <- s.tag,
      metric <- s.metric
    }
}

-- Type element abstract rule
abstract rule Type extends BasicElement {
  from
    s : Xirup!Type
  to
    t : Xirup!Type {
      constrainttemplate <- s.constrainttemplate,
      constraint <- s.constraint
    }
}

-- Relation element abstract rule
abstract rule Relation extends BasicElement{
  from
    s : Xirup!Relation
  to
    t : Xirup!Relation {
    }
}
```

Figure 54: XirupCopy2006 (1)

```
--InfoType abstract rule
abstract rule InfoType extends Type {
  from
    s : Xirup!InfoType
  to
    t : Xirup!InfoType {
      externalModelReference <- s.externalModelReference --this is mapped since it derives from Element inheritance
    }
}

abstract rule AbstractActivity extends Type {
  from
    s : Xirup!AbstractActivity
  to
    t : Xirup!AbstractActivity {
      next <- s.next,
      previous <- s.previous,
      diagram <- s.diagram,
      externalModelReference <- s.externalModelReference --this is mapped since it derives from Element inheritance
    }
}

--XSH SECTION

--ExternalModel element
rule ExternalModel extends Element {
  from
    s : Xirup!ExternalModel
  to
    t : Xirup!ExternalModel {
      name <- s.name,
      type <- s.type,
      document <- s.document
    }
}

--ExternalModelType element
rule ExternalModelType extends Element {
  from
    s : Xirup!ExternalModelType
  to
    t : Xirup!ExternalModelType {
      name <- s.name,
      type <- s.type,
      document <- s.document
    }
}
```

Figure 55: XirupCopy2006 (2)

### 6.3.3 Work case example

This section let the user have a look of some screenshots taken from a real transformation work case, the one that has been done to port the XirupCopy2004 transformation with the Xirup model version 0.0.1 to the XirupCopy2004 transformation with the Xirup model version 1.0.0.

ComponentTypeAggregation now extends ComponentType:

Features *name*, *library*, *requires*, *offers*, *infotype*, *attributetype*, *externalModelReference* and *behaviour* added to “ComponentTypeAggregation” rule due to inheritance from *ComponentType*. Moreover, *subcomponents* field has been added.

No more *ComponentType.parts* in “ComponentType” rule.

```
rule ComponentType{
  from
    compIn: Xirup!ComponentType
  to
    compOut: Xirup!ComponentType(
      id <- compIn.id,
      tag <- compIn.tag,
      metric <- compIn.metric,
      name <- compIn.name,
      constraint <- compIn.constraint,
      constrainttemplate <- compIn.constrainttemplate,
      externalModelReference <- compIn.externalModelReference,
      name <- compIn.name,
      library <- compIn.library,
      offers <- compIn.offers,
      requires <- compIn.requires,
      attributetype <- compIn.attributetype,
      parts <- compIn.parts,
      infotype <- compIn.infotype,
      behaviour <- compIn.behaviour
    )
}

rule ComponentTypeAggregation{
  from
    compTAIn: Xirup!ComponentTypeAggregation
  to
    compTAOut: Xirup!ComponentTypeAggregation(
      id <- compTAIn.id,
      tag <- compTAIn.tag,
      metric <- compTAIn.metric,
      name <- compTAIn.name,
      constraint <- compTAIn.constraint,
      constrainttemplate <- compTAIn.constrainttemplate,
      componenttype <- compTAIn.componenttype,
      exposed <- compTAIn.exposed,
      connection <- compTAIn.connection,
      required <- compTAIn.required
    )
}
```

Deleted

Figure 56: Old XirupCopy2004

```
--ComponentType element
rule ComponentType {
  from
    -- select only the type which has been added to the model by manual editing and beware to specify
    -- that we want to map only Type element, not ComponentType which is actually a subclass
    s: Xirup!ComponentType (not s.oclIsTypeOf(Xirup!ComponentTypeAggregation))
  to
    t: Xirup!ComponentType(
      id <- s.id,
      tag <- s.tag,
      metric <- s.metric,
      constrainttemplate <- s.constrainttemplate,
      constraint <- s.constraint,
      name <- s.name,
      library <- s.library,
      requires <- s.requires,
      offers <- s.offers,
      infotype <- s.infotype,
      attributetype <- s.attributetype,
      behaviour <- s.behaviour,
      externalModelReference <- s.externalModelReference
    )
}

--ComponentTypeAggregation element
rule ComponentTypeAggregation {
  from
    s: Xirup!ComponentTypeAggregation
  to
    t: Xirup!ComponentTypeAggregation(
      id <- s.id,
      tag <- s.tag,
      metric <- s.metric,
      constrainttemplate <- s.constrainttemplate,
      constraint <- s.constraint,
      name <- s.name,
      library <- s.library,
      requires <- s.requires,
      offers <- s.offers,
      infotype <- s.infotype,
      attributetype <- s.attributetype,
      behaviour <- s.behaviour,
      externalModelReference <- s.externalModelReference,
      subcomponents <- s.subcomponents,
      exposed <- s.exposed,
      connection <- s.connection,
      required <- s.required
    )
}
```

Inserted (red box)  
and deleted (blue  
box) features

Figure 57: New XirupCopy2004



ComponentInstance is changed:

*subcomponents* and *supercomponent* features instead of *parts*

```
rule ComponentInstance{
  from
    instIn: Xirup!ComponentInstance
  to
    instOut: Xirup!ComponentInstance(
      id <- instIn.id,
      tag <- instIn.tag,
      metric <- instIn.metric,
      externalModelReference <- instIn.externalModelReference,
      name <- instIn.name,
      needs <- instIn.needs,
      provides <- instIn.provides,
      constraint <- instIn.constraint,
      infoinstance <- instIn.infoinstance,
      type <- instIn.type,
      attribute <- instIn.attribute,
      parts <- instIn.parts
    )
}

--ComponentInstance element
rule ComponentInstance {
  from
    s: Xirup!ComponentInstance
  to
    t: Xirup!ComponentInstance(
      id <- s.id,
      tag <- s.tag,
      metric <- s.metric,
      externalModelReference <- s.externalModelReference,
      name <- s.name,
      needs <- s.needs,
      provides <- s.provides,
      constraint <- s.constraint,
      infoinstance <- s.infoinstance,
      type <- s.type,
      subComponents <- s.subComponents,
      superComponent <- s.superComponent,
      attribute <- s.attribute
    )
}
```

Deleted

Inserted

**Figure 58: Another example of porting XirupCopy transformation**

## 7 Annexes

### 7.1 Acronyms and Glossary

Acronym	Meaning
XIRUP	eXtreme end-User dRiven Process
XSM	XIRUP system model
TBMS	To be modernised system
MS	Modernised system
UML	Unified modelling language
UC	Use case
OCL	Object constraint language
KB	Knowledge base
EMF	Eclipse modelling framework
ATL	Atlas Transformation Language
TT	XSM Transformation Tool
RCP	Rich Client Platform

### 7.2 Reference Documents

[2007a] MOMOCS team, D11 XIRUP Methodology Requirements

[2007b] MOMOCS team, D12 Methodology Requirement Analysis Report: industrial solution and Telecommunications Cases

[2007c] MOMOCS team, D13 State of the art analysis on Actual Mainstream Methodologies

[2007d] MOMOCS team, D21 XIRUP Supporting tools Requirements

[2007e] MOMOCS team, D31 XIRUP Methodology Specifications

[2007f] MOMOCS team, D41 XIRUP Supporting Tools Specifications

[ATL] ATLAS Group (INRIA & LINA), "ATL Project" <http://www.sciences.univ-nantes.fr/lina/atl/> and <http://www.eclipse.org/m2m/atl/>

[ATL\_DOC] ATL Documentation:

<http://www.eclipse.org/m2m/atl/doc/>

[http://www.eclipse.org/m2m/atl/doc/ATL\\_Starter\\_Guide.pdf](http://www.eclipse.org/m2m/atl/doc/ATL_Starter_Guide.pdf)

[http://www.eclipse.org/m2m/atl/doc/ATL\\_User\\_Manual%5Bv0.7%5D.pdf](http://www.eclipse.org/m2m/atl/doc/ATL_User_Manual%5Bv0.7%5D.pdf)

[http://wiki.eclipse.org/ATL\\_Superimposition](http://wiki.eclipse.org/ATL_Superimposition)

<http://www.eclipse.org/m2m/atl/atlTransformations/>

[Eclipse] Eclipse Project, <http://www.eclipse.org>

[Eclipse\_ref] <http://www.eclipse.org/articles/using-perspectives/PerspectiveArticle.html>

[EMF] Eclipse Modeling Framework, <http://www.eclipse.org/modeling/emf/>

[Europa] Eclipse Europa project, <http://www.eclipse.org/europa/>

[ECORE] Ed Merks and Dave Steinberg, "From Models to Code with the Eclipse Modeling Framework", [http://www.eclipsecon.org/2005/presentations/EclipseCon2005\\_Tutorial11final.pdf](http://www.eclipsecon.org/2005/presentations/EclipseCon2005_Tutorial11final.pdf)

[OCL] <http://www.omg.org/technology/documents/formal/ocl.htm>

[UML2Tools] <http://www.eclipse.org/modeling/mdt/?project=uml2#uml2>