Overview

The aim of this Use Case is to implement a set of mappings already defined between a pair of metamodels using the ATL language. More specifically, we want to implement the transformation for obtaining a Service Process model from an Extended Use Case model [2]. Both of them are defined in the framework of the SOD-M methodology, a model driven method for service-oriented web applications development (SOD-M [1]). The method starts from a high level business model and allows obtaining a service composition model that simplifies the mapping to a specific web service technology. The process is summarized in Figure 1. In this Use Case we focus in the mapping from the Extended Use Case Model to the Service Process Model.

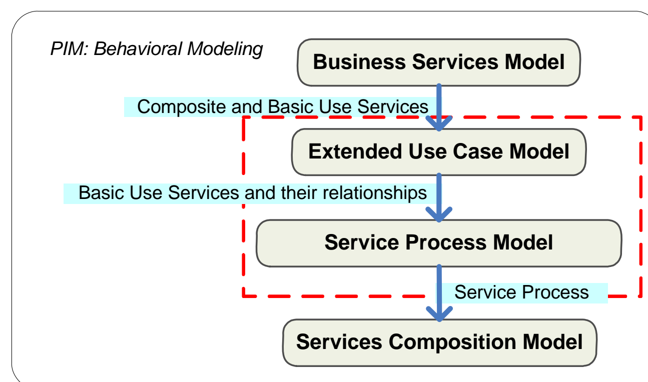

Figure 1. Behavioral Modeling process in SOD-M

To implement this transformation, we need to provide with some extra information about the elements of the source model. These extra data express some relations between the elements of the source model. So, we use a weaving model to define an annotation model [3] that contains this additional information. The approach is depicted in Figure 2.

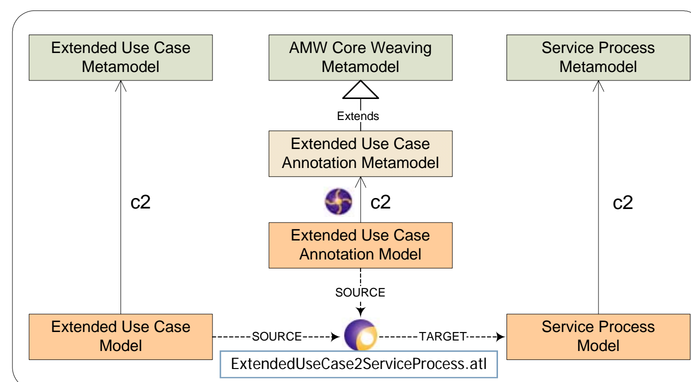

Figure 2. Use Case Overview

For each execution of the ATL program, i.e., for each source model considered, we define a weaving model conforming to a new weaving metamodel. The weaving model contains the information needed to execute the transformation, that is, the value of the *parameters* needed to execute some of the rules of the ATL program. Thus, both the source and the weaving model are taking as inputs by the transformation in order to generate the target model.

Installation

This use case requires Eclipse + EMF + ATL + AM3 + AMW to be installed.

- Download ATL/AM3: http://www.eclipse.org/m2m/atl/download/
- Download AMW: http://www.eclipse.org/gmt/amw/download/

If you want to use the latest version, you can download ATL and AMW sources from eclipse CVS. You can find the instructions on the ATL wiki:
http://wiki.eclipse.org/index.php/ATL/How_Install_ATL_From_CVS/.

After installing the tools, download the archive file containing this use case and unzip it into a new ATL project.

SOD-M Project Content

The project contains 2 scenarios: one (*Simplified.Models* folder) being a reduced version of the other (*Complete.Models* folder). Both of them are defined during the development of a Web Information System for conference management. Specifically they are used to model the behaviour of the Web system. Next, we detail the content of the ATL downloadable project:

Transformation File

- AnnotatedEUC2ServiceProcess.atl → the ATL transformation valid for both scenarios

Metamodel Files

- Metamodels/ExtendedUseCase.ecore (.km3) → The Extended Use Case metamodel. It is based on the UML2.0 Use Case metamodel.
- Metamodels/ServiceProcess.ecore (.km3) → The Service Process metamodel. It is based on the Activity Diagrams UML 2.0 metamodel.
- Metamodels/EUCAnnot_base_extension.km3 → Extension of the core weaving metamodel for annotating Extended Use Case models.
- Metamodels/AnnotatingEUC.ecore → The Weaving metamodel. Later on we will describe how to use the AMW user interface to obtain this metamodel from the extension above.

Model Files for Simplified Scenary

- Simplified.Models/Input.xmi → Extended use Case model in which just one Service has been included, the *View Submitted Articles* service.
- Simplified.Models/AnnotatedEUC.amw → Annotation model for the previous model.
- Simplified.Models/Output.xmi → Service Process Model. It is the obtained executing the transformation.

Model Files for Complete Scenary (*Complete.Models* folder)

- CompleteModels/Input.xmi → Extended Use Case model in which the three services offered by the system have been included: *View Subbmitted Articles*, *Edit Author Data* and *Submit Article.*
- CompleteModels/AnnotatedEUC.amw → Annotation model for the previous model.
- Complete.Models/Output.Test.xmi → Service Process model obtained as output of the transformation.

Build Files

- **build.xml** → ANT project comprising the tasks needed to execute the transformation. They are parameterized in order to support the two possible configurations (Simplified and Complete scenarios)
- **build.properties** → value of the parameters for the build file

The Metamodels

Three different metamodels are used in this project: the Extended Use Case and the Service Process metamodels and a new extension of the core weaving metamodel for annotating Extended Use Case models.

Extended Use Case metamodel

The *Extended Use Case* model represents the different services offered by a Web Information System, as well as the functionalities required to carry out each service. We use the KM3 language to define a new DSL. It is a simplified version of the UML 2.0 Use Case metamodel in which we add three new concepts.

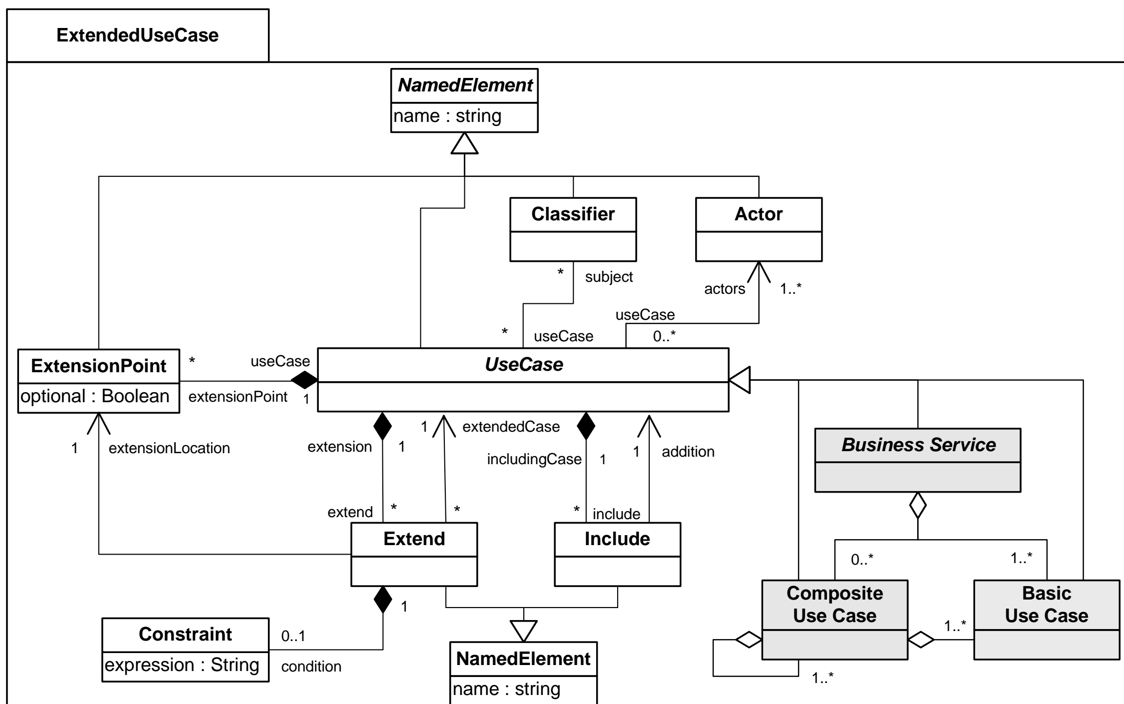

Figure 3. Extended Use Case metamodel

We define a *Business Service* as a complex functionality offered by the system, which satisfies a specific need of the consumer. The consumers of the system are represented in this model as actors. To represent the portions of the functionality of a business service we define the *Basic Use Case* and the *Composite Use Case* classes. The former represents a basic unit of behaviour of the web application, like *registering as a costumer*, while the later is an aggregation of either basic or composite use cases.

Service Process metamodel

This metamodel is a simplified version of the UML activity package. In the service process model we represent the activities that must be carried out for delivering a business service. The activities of this model are called *Service Activities*. The service activities are obtained transforming the basic use cases identified in the previous model

into *Service Activities* of a given *Activity* process. So, the services activities represent a behaviour that is part of the execution flow of a business service. The *ServiceActivity* class is shadowed in Figure 4 which shows the Service Process metamodel.

Figure 4. Service Process metamodel

Extended Use Case Annotation metamodel

If we want to use an Extended Use Case model as input in a model transformation, we need some extra information. For instance, if a Use Case includes two or more Use Cases, the order in which they should be executed in the including Use Case should be specified, since the mapping rule for *<<include>>* relationships gives the two options shown in the picture below.

Figure 5. Include Relationships Mapping Rule

These data are not relevant to the model itself, so we use another model to collect them. More specifically, since these data represents relations between the elements of the Extended Use Case model, we use a weaving model [4]. This process is known as *annotation* and the weaving model is known as the *annotation model* [3]. Then each link in the weaving model represents an annotation for the woven model. All this given we define a new weaving metamodel for annotating Extended Use Case models. It is shown in Figure 6.

Juan M. Vara Mesa
Kybele Research Group
University Rey Juan Carlos

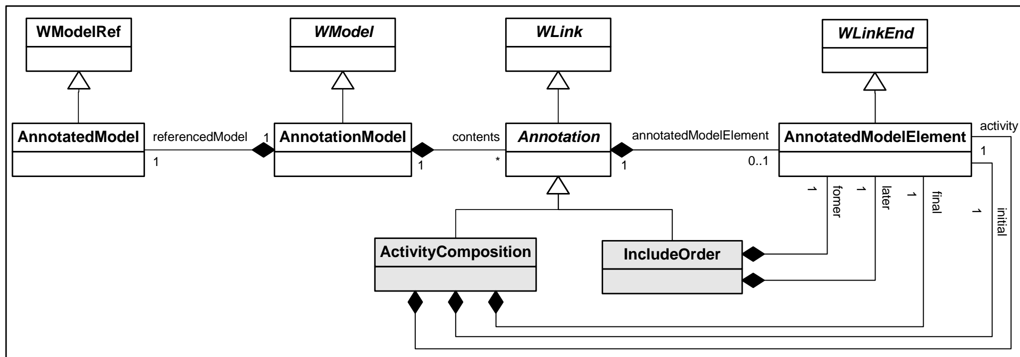

Figure 6. Weaving metamodel for annotating Extended Use Case models

For instance, the *IncludeOrder* annotation helps in the mapping of several *include* relationships attached to the same Use Case. It relates the two different included Use Cases, defining which one should be executed first. That is which Service Activity should precede the other.

The models

The source model

Here we will refer just to the *main* source model. As a matter of fact there will be two source models, this one and the annotation model, that can be thought of as an auxiliary source model. In this case we use part of a case study we have already developed: a web system for conference management. The corresponding Extended Use Case model is shown in Figure 7

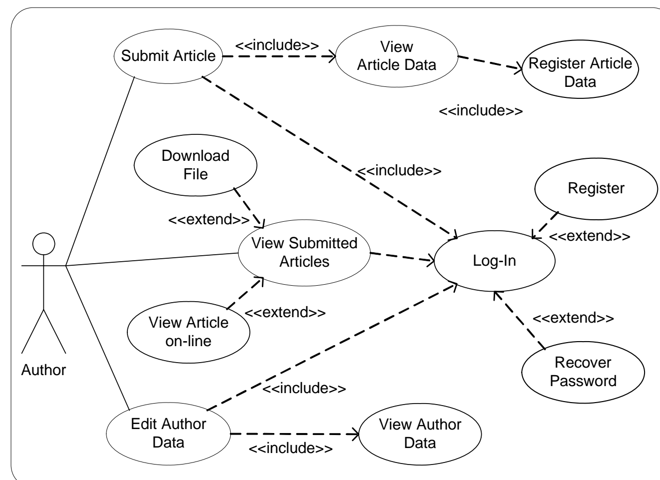

Figure 7. Extended Use Case model of the Web system for conference management

The system offers three different services: *Submit an Article*, *View Submitted Articles* and *Edit Author Data*. In order to provide with this complex services, some basic services are needed, as the *Log-In* or the *Register* ones. To model the relation between the different Use Cases two types of associations are used: *<<include>>* and *<<extend>>*. The former implies that the behaviour of the included Use Case is inserted in the including Use Case, while the later specifies how and when the behaviour defined in the extending use case can be inserted into the behaviour defined in the extended use case.

The Target model

The Service Process model for our case study is shown in Figure 8. Each complex service identified in the previous model is mapped to an activity, while the basic services that it uses are represented as service activities. This way, we have three different activities that use a set of service activities. For instance the Log-In service activity one is used by the three activities.

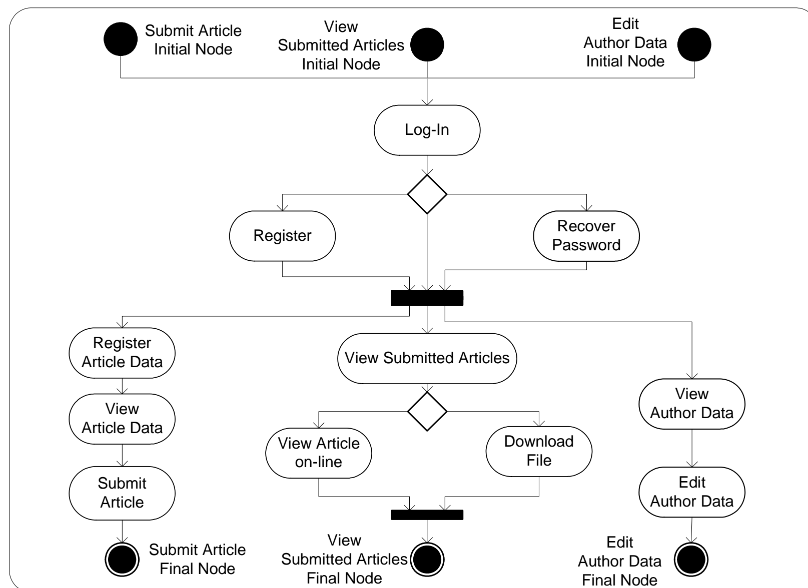

Figure 8. Service Process model of the Web system for conference management

Notice that the previous model did not show which the complex services were, those that had to be mapped to activities. This kind of information is not conceptually relevant to be part of the corresponding metamodel. So it will be collected in the annotation model.

Executing the Use Case

Defining the annotation model

As mentioned, we have to annotate the *main source* model to provide with the additional information needed to execute the transformation. We will use the Weaver Wizard (*File → New → Weaving Model*) to create a weaving model that will comprise those annotations. The different steps are shown in the picture below:

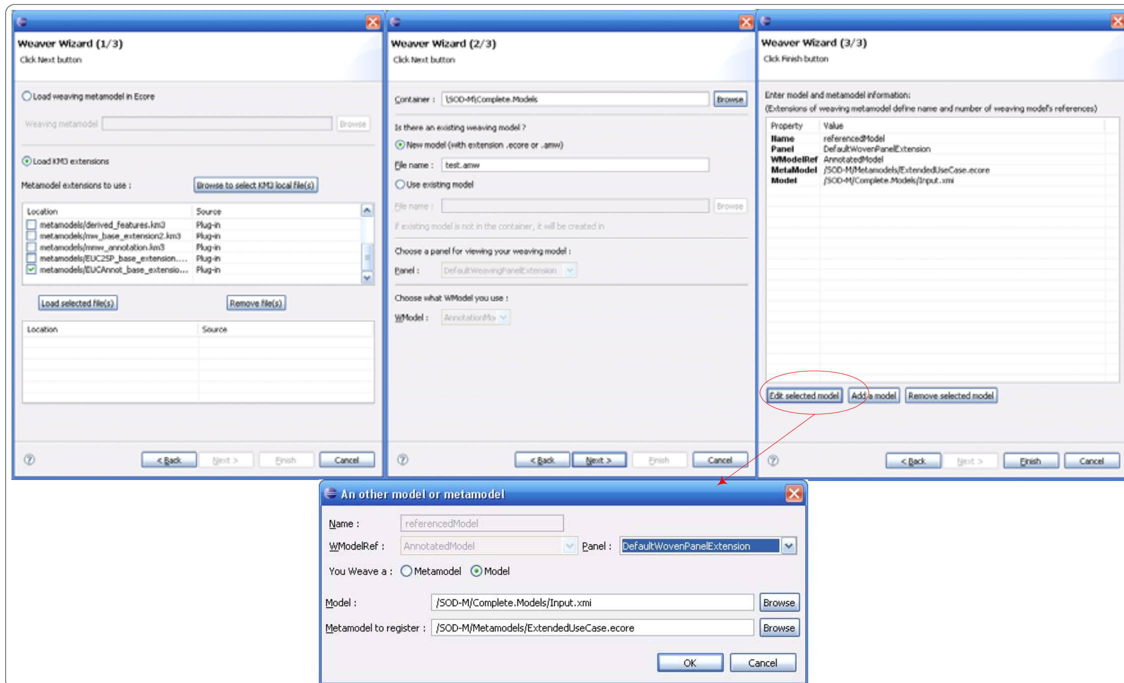

Figure 9. Weaver Wizard for new weaving models

- Weaver Wizard (1/3): first of all, you have to select which extensions of the core weaving metamodel you want to use for defining the new weaving model (in our case the *EUCAnnot_base_extension.km3* extension is used). There are two ways of loading the extension: using the *Browse to select KM3 file(s)* button and then localize the file in your Eclipse workspace, or developing a new plug-in[1] that extends the AMW plug-in (in the *org.eclipse.weaver.metamodelextensionID* extension point). Using the later, the new extension of the weaving metamodel will be directly available in the wizard (as shown in Figure 9).
- Weaver Wizard (2/3): now you must assign a container as well as a name for the new weaving model.
- Weaver Wizard (3/3): finally, you have to choose which will be the woven model (that is, the annotated model) and, in case it is not a metamodel, the corresponding metamodel. To do so, press the *Edit Selected Model* button that opens the *An other model or metamodel* window.

[1] You can find the AMW plug-in for using the annotation metamodel for annotating Extended Use Case models downloads section of this Use Case (download the file)

At the end of the wizard the AMW graphical interface looks like the screen capture of Figure 10(a). It presents an empty annotation model on the right panel (the weaving model) and the Extended Use Case model to annotate on the left one (the woven model).

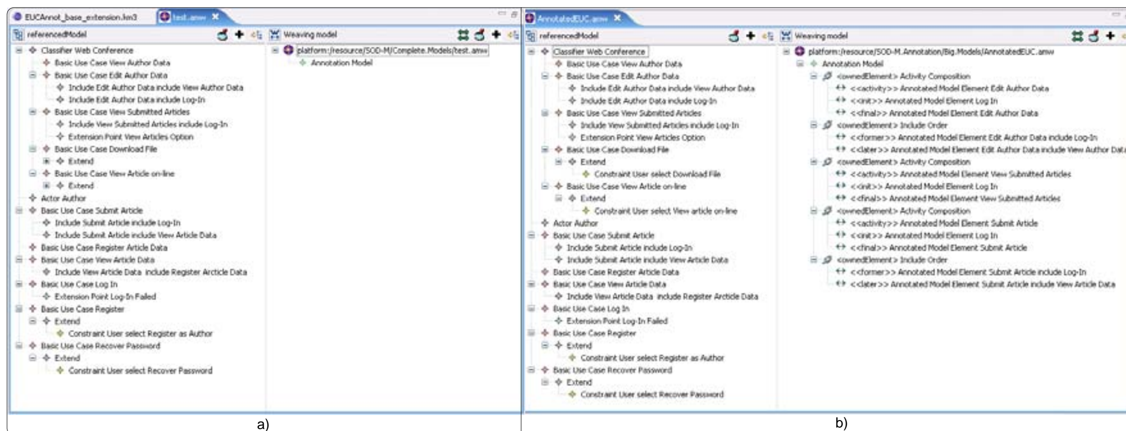

Figure 10. (a) Empty and (b) Final annotation models

From now on, we annotate the woven model by adding new annotations in the weaving model. Next we describe the steps to add a new *IncludeOrder* annotation. This annotation indicates that the *Log-In* Basic Use Case must precede the *View Article Data* Basic Use Case:

- Right click in the weaving panel → New Child → IncludeOrder
- Drag and drop the *Submit Article include Log-In* Include from the left panel to the new annotation you have just created in the right panel
- A menu will ask you if this will be the *former* or the *later AnnotatedModelElement*. Choose the first option since you have to Log-In before being able to View the article data.
- After doing the same with the *Submit Article include View Article Data* element, it is automatically identified as the *later AnnotatedModelElement*.

You have to follow these steps in order to add an *IncludeOrder* annotation for each pair of *include* relationships attached to a same Use Case. The annotation defines the order in which the included Use Cases should be executed to carry out the including Use Case.

In the same way, we add a new *ActivityComposition* annotation for each service provided by the system. It serves to identify the complex services provided by the system, as well as its entry and exit points. Here we add the one corresponding to the *Submit Article* service:

- Right click in the weaving panel → New Child → ActivityComposition
- Drag and drop the *Submit Article* Basic Use Case to the new link in the right panel.
- This time, the menu will ask if this Basic Use Case should be identified as the *activity*, the *init* or the *final* annotated model element. We choose the first option since we want to a *Submit Article* activity to be created in the Service Process model (the output model).
- To define the entry and exit points for this service we drag and drop the Log-In (*init*) and Submit Article (*final*) Basic Use Cases. This way we are indicating that the Submit Article service starts by log-in the system and finishes by effectively submitting the article (we can identify this step with the action of pushing the *enter* bottom)

Following the described steps for each service provided by the system, as well as for every pair of include relationships attached to a same Use Case, we obtain the weaving model (or annotation model) shown in Figure 10(b). Notice that if you click over any element in the weaving model on the right, the AMW interface points out automatically the corresponding element/s in the woven model on the left.

Obtaining the annotation metamodel

As we will show later, to execute the transformation we need a metamodel for each one of the input models. Up to know we have an extension for the core weaving metamodel (*EUCAnnot_base_extension.km3*) but it is not a metamodel itself. To obtain a proper metamodel for the annotation model we use the facilities provided by the AMW plug-in as shown in Figure 11. We just have to make right click over the root of the weaving model and choose the *Save weaving metamodel in Ecore format* option.

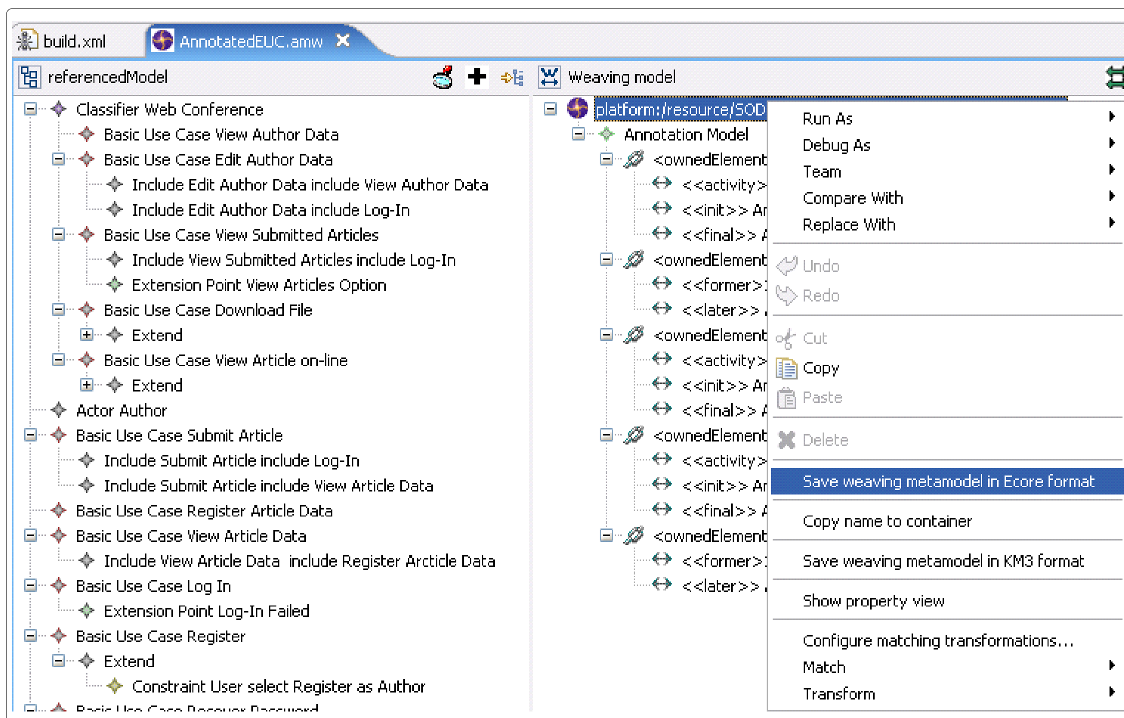

Figure 11. Saving weaving metamodels

Coding the transformation

Once we have defined the *main* input model (the Extended Use Case model) and the *auxiliary* input model (the annotation model), it is time to code the model transformation program. Here we will focus just in showing an example of how to use the information provided by the annotation model. Obviously, skills in the coding of ATL programs are supposed to the reader.

The annotation of the input model allows us to add the missing data we need to execute the transformation. In order to use this information, we just have to include some helpers in the ATL program. For instance, to map an *include* object we have to know if it is related with other *include* objects (remember the ambiguity about the mapping of include relationships). Thus, we define the following helper that navigates the weaving model (the annotation model) looking for *IncludeOrder* annotations that refers to the given *include* object.

```
helper context ExtendedUseCase!Include def: getIncludeOrderLink() : AMW!IncludeOrder =
        AMW!IncludeOrder.allInstances()->asSequence()->
                select(link | link.former.element.ref = self.__xmiID__)->first();
```

The helper shown above navigates the weaving model to find an *IncludeOrder* annotation, whose *former* element is the same that was being mapped when the helper was invoked. To identify the link, the _xmiID_ property is used.
Then we define two different rules for mapping *include* objects and we include a guard in one of them to distinguish which one should be used in each specific case. The guard invokes the helper we have just shown.

```
rule IncludeSimple2ServiceProcess {
	from
		inc : ExtendedUseCase!Include(inc.getIncludeOrderLink().oclIsUndefined())
	to
		edge : ServiceProcess!ControlFlow (
			name <- ('[' + inc.addition.getFinalActionName() + ' to '+ inc.includingCase.name + ']'),
			source <- inc.addition.getFinalAction(),
			target <- inc.includingCase
		)
}
```

Executing the transformation

To launch the transformation we have opted for using an *ANT project* [5]. An ANT project is basically a way of automation the execution of business processes using XML. Each one is coded in a *build.xml* file. Next, we detail its content for the present Use Case (you can find it in the downloadable Eclipse project).

- First, we give a name to the project using the corresponding attribute of the *project* element. This element also comprises a *default* attribute that points to the task that will be executed by default whenever the project is launched. We will go deep into this matter later.

```
<project name="AnnotatedEUC2ServiceProcess" default="transformation">
	<description>
			This project launch the ATL transformation for obtaining a Service Process model
			from an Extended Use Case model
	</description>
	<!-- Project properties -->
	<property file="build.properties" />

	<!-- ================================ 
	<target name="transformation" depends="loadModels">

	<target name="loadModels">

</project>
```

Figure 12. Summarized build file

- Then, the *description* element allows adding a short description of the aim of this project
- We can define some parameters or properties for the project and store then in something akin to a parameter file: the *property file* (*build.properties* for this project). Here we use this feature to support the execution of the transformation for two different sets of input models as well as for modularize the build file. (see the *Readme.txt* file in the downloadable Eclipse project).
- Next, two *target* elements are found. Each target is a set of tasks you want to be executed.

```
<!-- ================================ 
<target name="transformation" depends="loadModels">
	<am3.atl path="${atl.file}">
	<am3.saveModel model="${target.model.name}" path="${target.model.file}" />
</target>

<target name="loadModels">
	<am3.loadModel modelHandler="EMF" name="${source.metamodel.name}" metamodel="MOF" path="${source.metamodel.file}" />
	<am3.loadModel modelHandler="EMF" name="${source.model.name}" metamodel="${source.metamodel.name}" path="${source.model.file}" />
	<am3.loadModel modelHandler="EMF" name="${weaving.metamodel.name}" metamodel="MOF" path="${weaving.metamodel.file}" />
	<am3.loadModel modelHandler="EMF" name="${weaving.model.name}" metamodel="${weaving.metamodel.name}" path="${weaving.model.file}" />
	<am3.loadModel modelHandler="EMF" name="${target.metamodel.name}" metamodel="MOF" path="${target.metamodel.file}" />
</target>
```

Figure 13. Content of the *transformation* and *loadModels* targets

- The first one, so-called *transformation* is the model transformation we want to carry out. It will be the task executed by default, as it is stated by the *default* attribute of the *project* element. Moreover, the *transformation* task depends on other task: the *loadModels* task. This is due to the fact that you can only execute the transformation when you have already loaded the models.
    - The *transformation* target includes two tasks: the first one defines which will be the ATL program executed and the second one allows storing the result in a newly created file.
- A *loadModel* task is added in the *loadModels* target for each model (and metamodel) involved in the transformation. The value of the *name* attribute is the one that should be used whenever the corresponding model has to be referred in any other part of the build file.

Finally, we can launch the ANT project. Right click in the *build* file and select *Run as → Ant Build*. Go to the JRE tab and choose the *Run in the same JRE as the worspace*

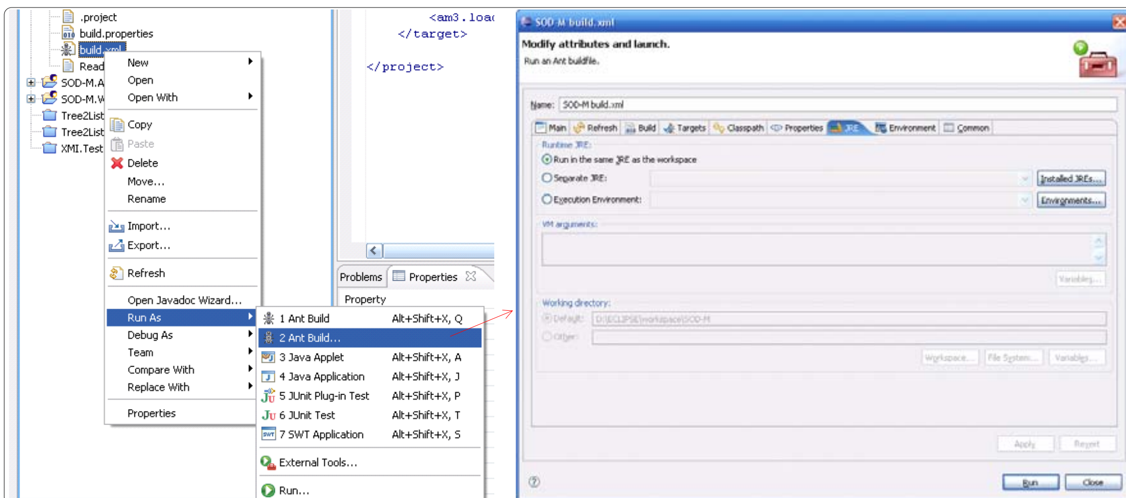

Figure 14. Running ANT tasks

References

[1] De Castro, V., Marcos, E., López Sanz, M. *A Model Driven Method for Service Composition Modeling: A Case Study*. International Journal of Web Engineering and Technology, Vol. 2(4), Eds: Inderscience Enterprise Ltd, pp. 335-353, 2006.

[2] De Castro, V., Vara, J.M., Marcos, E. *Model Transformation for Service-Oriented Web Applications Development*. Workshop Proceedings of 7th International Conference on Web Engineering. July 2007, pp. 284-198.

[3] Didonet Del Fabro, M., *Metadata Management Using Model Weaving and Model Transformation*. Ph D Thesis. Université de Nantes. September 2007.

[4] Didonet Del Fabro, M., Bézivin, J. and Valduriez P. *Weaving Models with the Eclipse AMW plugin*. In: Eclipse Modeling Symposium, Eclipse Summit Europe 2006, Esslingen, Germany.

[5] Holzner, S., *Ant: the definitive guide (2º Ed.)*. O'Reilly (2005).