Tutorial: Model Transformation Chain

Kelly Garcés[1,2]

[1] AtlanMod team, EMN-INRIA
[2] ASCOLA team, EMN-INRIA, LINA

1 Tutorial purpose

– Transform high-level specifications of ATL transformation chains into Ant scripts.

2 Pedagogical objectives

– Put in practice the transformation process development described in the lecture Transforming Models with ATL.
– Take a look at ATL advanced features (i.e., inheritance, lazy rules, called rules)

3 Motivation

Model transformation is a main axis in Model Driven Engineering (MDE). In general, an MDE system executes model transformation chains. The chain specifies: 1) when to execute a transformation, 2) what models the transformations need/yield, and 3) how the transformations interact each other. Fig. 1 shows a particular transformation chain that refines models conforming to a metamodel (i.e., OutMM). Each transformation takes the models InM1 (conforming to InMM1) and InM2 (conforming to InMM2) as input, and yields a model OutM as output. Unlike the first transformation, the subsequent transformations take the previously produced model as an additional input.

In particular, the developers can instrument ATL transformation chains using Ant projects. These projects are supposed to execute at least three kind of tasks:

1. am3.loadModel loads a model (i.e., terminal model or metamodel).
2. am3.atl executes an ATL transformation.
3. am3.save persists a model.

Fig. 2 gives the syntax of an Ant project that instruments the transformation chain described in Fig. 1:

Let's suppose the developer has to specify large ATL transformation chains using Ant projects. This task consumes a lot of development time, and it is error-prone. How can we reduce the effort to specify ATL transformation chains?

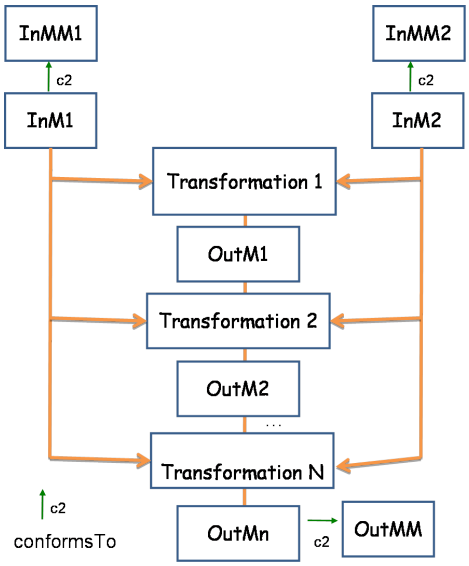


Fig. 1. A model transformation chain

```
<project name="myChain" default="TransformationN">

 <target name="loadMetamodels">
           <am3.loadModel modelHandler="EMF" name="InMM1" metamodel="\%EMF" path="Metamodel/InMM1.ecore"/>
                ...
           <am3.loadModel modelHandler="EMF" name="OutMM" metamodel="\%EMF" path="Metamodel/OutMM1.ecore"/>
 </target>

 <target name="loadModels" depends="loadMetamodels">
           <am3.loadModel modelHandler="EMF" name="InM1" metamodel="InMM1" path="Models/InM1.xmi"/>
     </target>
     ...
  <target name = 'Transformation2' depends = 'loadModels, Transformation1'>
    <am3.atl path = '/ATL/Transformation2.asm' vm = ''>
      <inModel name = 'InMM1' model = 'InMM1'/>
      <inModel name = 'InMM2' model = 'InMM2'/>
      <inModel name = 'OutMM' model = 'OutMM'/>
      <inModel name = 'InM1' model = 'InM1'/>
      <inModel name = 'InM2' model = 'InM2'/>
      <inModel name = 'OutM3' model = 'OutM3'/>
      <outModel name = 'OutMN' model = 'OutMN' metamodel = 'OutMM'/>
    </am3.atl>
    <am3.saveModel model = 'OutM3' path = 'Models/OutM3.xmi'/>
  </target>
 ...
</project>
\end{lstlisting}
```

Fig. 2. A model transformation chain using Ant

4 The Cadena Language

We have implemented the Cadena language to answer the question raised above. Cadena allows to specify simple model transformation chains. Each transformation (to be executed) takes several models as input, and yields one model as output. Listing. 1.1 presents the syntax of a Cadena program. This instruments a chain with 3 transformations.

Listing 1.1. A model transformation chain using Cadena, large version

```
1  Chain myChain {
2
3    OutM1 = Transformation1(InM1:InMM1, InM2:InMM2)
4    OutM2 = Transformation2(InM1:InMM1, InM2:InMM2, OutM1)
5    OutM3 = Transformation3(InM1:InMM1, InM2:InMM2, OutM2)
6
7  }
```

Listing. 1.2 presents a shorter Listing. 1.1 version. We use Transformation2 call as argument in Transformation3 call, instead of OutM2.

Listing 1.2. A model transformation chain using Cadena, short version

```
1 Chain MyChain {
2
3   inM1 = InM1 : InMM1;
4   inM2 = InM2 : InMM2;
5
6   OutM1 = Transformation1(inM1, inM2);
7   OutM3 = Transformation3(inM1, inM2,   Transformation2(inM1,inM2,OutM1)
        ↪);
8
9 }
```

5 Challenge

We may specify transformation chains using Cadena faster than before, but how to make these specifications executable?

Fig. 3 shows an MDE solution. An injector puts Cadena programs into xmi format. Cadena2AntScript transforms Cadena models into Ant script models. Finally, an extractor extracts the Ant script model into an executable Ant script. In this tutorial, we propose you to develop the transformation Cadena2AntScript.

Prerequisites We provide a TCS project that contains all the files required to specify Cadena programs (i.e., metamodel, injector, extractor, syntax, samples, etc.). We moreover provide the following files:

- The Ant script metamodel
- The file build.xml that automatizes the process described in Fig. 3

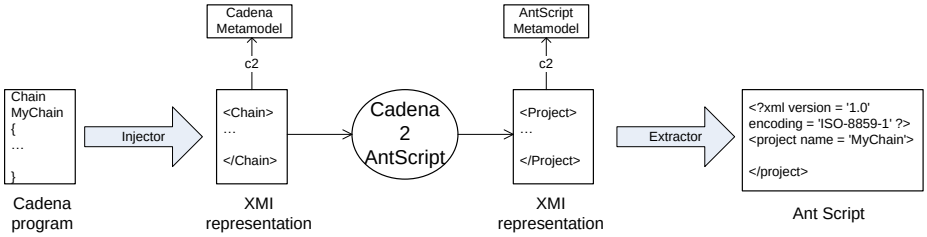


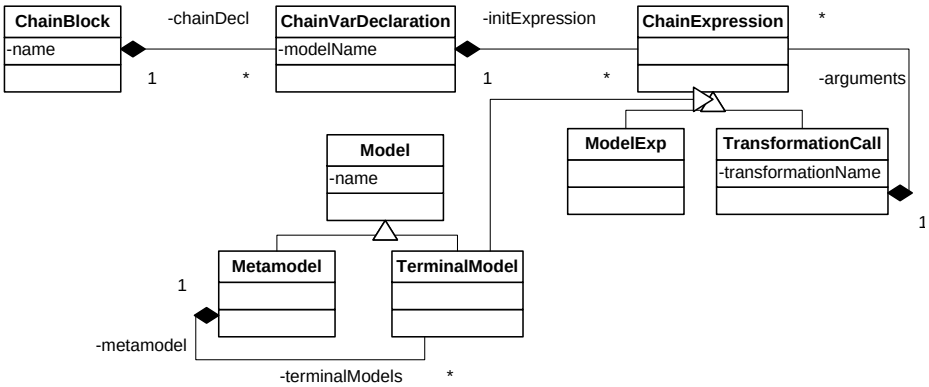
Fig. 3. Transforming Cadena programs to Ant scripts, overview

Instructions

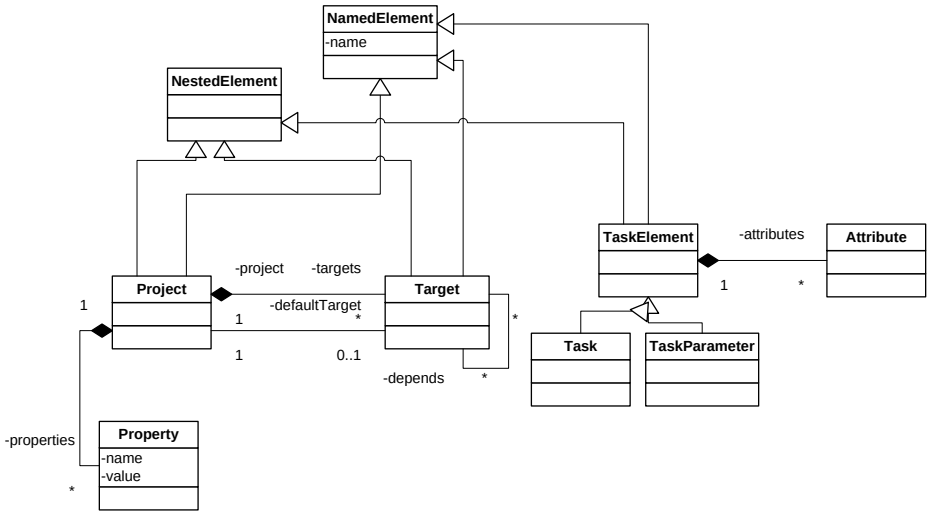
1. Take a look to the Cadena and Ant script metamodels (see Section 6).
2. Figure out what elements described in Fig. 2 and Listing. 1.2 conform to the concepts of Cadena and Ant script metamodel.
3. Write the correspondences between the Cadena and Ant script concepts (see Annex 1).
4. Implement the ATL transformation rules.
5. Check and run the build.xml file

ATL Styles We can implement ATL transformations in three styles: declarative, imperative or hybrid. We encourage the declarative style, however the transformation development (in practice) involves a hybrid flavor. We show you how to implement ATL programs using all these styles.

6 Metamodels



(a) The Cadena metamodel



(b) The Ant script metamodel