Automated chaining of model transformations with incompatible metamodels

Francesco Basciani *
Computer Science Department, University of L’Aquila, Italy

* Join work with: Davide Di Ruscio, Ludovico Iovino, Alfonso Pierantonio.
Special thanks to Juri Di Rocco
Roadmap

- Background
- Automating Model Transformation Chain
- Adapters
- Implementation - Demo
- Conclusions and future work
BACKGROUND
Overview

Hand write the whole transformation

USE
Overview

Chaining of Transformation

Repository

Discovering

RE-USE

AUTOMATED CHAINING OF MODEL TRANSFORMATIONS WITH INCOMPATIBLE METAMODELS
Repository structure

It is a structured repository of artifacts that are properly organized to enable their reuse.

It is constructed from a metamodel, general enough to allow to store artifacts of any type:

- models;
- metamodels;
- transformations;
- ecc…

Keeping the relationship among them.
Repository graph representation

Example of repository:

- **Nodes** represent metamodels
- **Arcs** represent Transformation

It is represented as a graph
Type of Composition*

- **Internal Composition**
  
  A new transformation is defined as the transformations to be composed.

- **External Composition**
  
  Chaining separate model transformations and passing models from one transformation to another

* With the term composition we refer to external composition. Moreover, the terms composition and chaining are used interchangeably.
Type of Composition*

• **Internal Composition**
  
  A new transformation is defined as the transformations to be composed.

• **External Composition**
  
  Chaining separate model transformations and passing models from one transformation to another.

* With the term composition we refer to external composition. Moreover, the terms composition and chaining are used interchangeably.
Chain of Transformation

The common way to compose transformations is to chain them by passing models from one transformation to another.

In order to chain transformations it is necessary to ensure the pre- and post-conditions of the considered transformations and to verify the metamodels compatibility condition.
In order to create a chain, the main condition to ensure is that the metamodel output of a transformation must be conform to the metamodel input of the next.
AUTOMATING MODEL TRANSFORMATION CHAINING PROCESS
Model Transformations Chaining Process

What we propose is to **automate** the process of creating a chain so that the user needs to provide only:
- source model;
- target metamodel.

1. Discovery of required model transformations
2. Execution of the derived model transformations chain
3. Derivation of the model transformations chain
Model Transformations Chaining Process

Discovery of required model transformations refers to the repository and search the transformations that can be compatible for a chain;

1. **Discovery** of required model transformations
2. **Derivation** of the model transformations chain

**Execution** of the derived model transformations chain
Model Transformations Chaining Process

Derivation of the model transformations chain
given directed graph representation solve the user's request and thus from the source metamodel search one or more paths that leave from the source and arrive to the target metamodel (user supplied).
The transformations present within the chain of transformations are performed in order.
Dealing with Multiple chains

The activity 1 might give place to different possible chains and the users have to select one of them.

If the user gives as input a Grafcet model and wants to generate a target PNML. The request can be satisfied by two possible chains, i.e.:

- T1 → T2;
- T1 → T3 → T4.

In these cases it is necessary to consider one or more evaluation criteria to select one of the derived chains:

- Coverage of the source metamodels;
- Usage;
- Number of transformations;
- Execution time.
Compatibility & Composability

Essential to create a chain are the notions of compatibility and composability:

Definition 1 - (Metamodel compatibility):
Let $MM_1$ and $MM_2$ be two metamodels, then $MM_1$ and $MM_2$ are compatible if $MM_1 \subseteq MM_2$ or $MM_2 \subseteq MM_1$.

Definition 2 - (Transformation composability):
Let $T_1 : MM_1 \rightarrow MM_2$ be a model transformation from the metamodel $MM_1$ to the metamodel $MM_2$.
Let $T_2 : MM_3 \rightarrow MM_4$ be a model transformation from the metamodel $MM_3$ to the metamodel $MM_4$.
Then, $T_1$ and $T_2$ are composable as $T_1 \circ T_2(T_2 \circ T_1)$ if $MM_4 \subseteq MM_1$ ($MM_2 \subseteq MM_3$).

Problem: Restricting the definition of transformation chains only for the cases of compatible metamodels can reduce the number of chains that might be potentially obtained.
Example

Suppose that the user wants to:
- **start** from a model **Grafcet**
- **get** a **PNML** model.

- **Nodes** represent metamodels
- **Arches** represent Transformation
Incompatibility

Due to the **definition of transformation composability** it is impossible to have a chain

\[ \text{Graf cet} \rightarrow \ldots \rightarrow \text{PNML} \]

Because starting from the transformations \( \text{Graf cet} \_\_ \_ \_ \_ \text{PetriNet}_{1.0} \) and \( \text{PetriNet}_{2.0} \_\_ \_ \_ \_ \text{to} \_\_ \_ \_ \_ \text{PNML} \)

\[ \text{PNML} \not\equiv \text{Graf cet} \]

and

\[ \text{PetriNet}_{1.0} \not\equiv \text{PetriNet}_{2.0} \]
PetriNet\textsubscript{1.0} vs PetriNet\textsubscript{2.0}

PetriNet\textsubscript{2.0} is a simple evolution of PetriNet\textsubscript{1.0}. In particular, the differences between the two versions are:

\begin{itemize}
  \item $\delta_1$ - pull up of the attribute \textit{name} to the new abstract metaclass \textit{NamedElement};
  \item $\delta_2$ - renaming of the metaclass \textit{Net} as \textit{PetriNet}.
\end{itemize}
ADAPTERS
Generation of the Adapter transformation

Our main contribution is focused in proposing an approach that (under certain conditions) permits to chain model transformations defined on incompatible metamodels.
Generation of the Adapter transformation

This is done by means of an adapter transformation that can be automatically synthesized from a delta model that represent the differences between the output and input metamodels of the transformations to be chained.
Coupled-evolution Problem

This discussion relates to the coupled-evolution problem. In such contexts metamodel manipulations can be classified by their corrupting or not-corrupting effects on corresponding artifacts as:

- **non-breaking changes**: changes which do not break the conformance of models to the corresponding metamodel;
- **breaking and resolvable changes**: changes which break the conformance of models even though they can be automatically co-adapted;
- **breaking and unresolvable changes**: changes which break the conformance of models which can not automatically co-evolved and user intervention is required.
Similarity function

In order to perform in automatic way an adaptation we have to introduce the similarity function used to calculate a similarity value among all the metamodels stored in the repository.

The function is launched every time a new transformation is stored (or deleted, or updated) into the repository.

This function assigns a value of similarity between the input and target metamodels of the transformation respect to all other metamodels already present in the repository.

For our purposes of tests on the prototype we used the function of similarity of “Falleri” that makes use of the Similarity Flooding Algorithm to calculate the similarity between two graphs.
Similarity function

The similarity values are maintained in a table like this:

<table>
<thead>
<tr>
<th></th>
<th>Grafnet</th>
<th>PetriNet\textsubscript{1.0}</th>
<th>PetriNet\textsubscript{2.0}</th>
<th>XML</th>
<th>PNML</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grafnet</td>
<td>1</td>
<td>0.20</td>
<td>0.30</td>
<td>0.29</td>
<td>0.26</td>
</tr>
<tr>
<td>PetriNet\textsubscript{1.0}</td>
<td>-</td>
<td>1</td>
<td>0.89</td>
<td>0.20</td>
<td>0.28</td>
</tr>
<tr>
<td>PetriNet\textsubscript{2.0}</td>
<td>0.30</td>
<td>0.89</td>
<td>1</td>
<td>0.30</td>
<td>0.30</td>
</tr>
<tr>
<td>XML</td>
<td>0.29</td>
<td>0.20</td>
<td>0.30</td>
<td>1</td>
<td>-</td>
</tr>
<tr>
<td>PNML</td>
<td>0.26</td>
<td>-</td>
<td>-</td>
<td>0.28</td>
<td>1</td>
</tr>
</tbody>
</table>

If the similarity value between two considered metamodels is higher then a fixed threshold (i.e. 0.80) such metamodels are further analyzed.

Then, if the resulting delta model consists of non-braking and breaking and resolvable changes, then corresponding adapter transformations can be generated.
Adapter $\text{PetriNet1.0, PetriNet2.0}$

Referring to our example (differences between PetriNet$_{1.0}$ and PetriNet$_{2.0}$), that the similarity value exceeds the threshold.

Furthermore, by analyzing the differences between the two metamodels we have:

$\delta_1$ - pull up of the attribute *name* to the new abstract metaclass *NamedElement*;
$\delta_2$ - renaming of the metaclass *Net* as *PetriNet*.

Where:
$\delta_1$ is a non-breaking change;
$\delta_2$ is breaking and resolvable change.

So our approach can be applied and the adapter generation is completely automated without requiring user intervention.
Adaptation from PetriNet 1.0 to PetriNet 2.0

All the rules of the PetriNet\textsubscript{2.0\texttt{--to--PNML}} transformation that refer to the pulled up attribute name are still valid and do not require any adaptation.

\begin{verbatim}
8 rule PetriNet {
  from
    g : Grafcet!Grafcet
  to
    p : PetriNet!Net
  (    location <- g.location,
       name <- g.name,
       elements <- g.elements,
       arcs <- g.connections
    )
}
\end{verbatim}

\begin{verbatim}
7 rule PNMLDocument {
  from
    e : PetriNet!PetriNet
  to
    n : PNML!PNMLDocument
      (        location <- e.location,
                xmlns <- uri,
                nets <- net
      ),
    uri : PNML!URI
      (            value <- 'http://www.informatik.hu-be
      ),
    net : PNML!NetElement
      (              name <- name,
                    location <- e.location,
                    id <- e.location,
                    type <- type_uri,
                    contents <- e.elements.union(e.arcs)
      )
}
\end{verbatim}

Grofcet\texttt{--to--PetriNet}_{1.0}  

PetriNet\texttt{2.0\texttt{--to--PNML}}
Adapter PetriNet1.0,PetriNet2.0

The rules of PetriNet\textsubscript{2.0\_to\_PNML} whose source patterns refer to PetriNet elements can be still applied on Net elements after an adaptation phase that simply copies all the Net elements to target PetriNet ones (Rename).

Thus the rule PNMLDocument can be applied on Net elements generated by Grafcet\textsubscript{\_to\_PetriNet}\textsubscript{1.0} after an adaptation step that generates PetriNet elements from the generated Net ones.
Adapter\textsubscript{PetriNet1.0,PetriNet2.0} 

More precisely, the adaptation step can be performed by means of the Adapter transformation 

(Fragment of the adapter transformation $Adapter_{PetriNet1.0,PetriNet2.0}$)
Adapter PetriNet1.0, PetriNet2.0

With this adapter the repository is update:
Screen – 1/ Upload of the source model

In the first step the user has to upload his model

Transformation Wizard

Upload input model
Upload your input model and click upload

File upload widget

Upload your xmi model

Link to the online web application: www.mdeforge.org
Screen – 2/ Selection of target metamodel

Once the model has been uploaded, the system is able to detect the metamodel the model conforms to. Afterwards, the user has to select the target metamodel as shown in this screenshot:

The list of metamodels shown to the user is retrieved from the repository used to store transformations and all the metamodels required to execute them. The system will only search in the repository the metamodels target of all these transformations.

Once the target metamodel is selected, the whole activity 1 of the process is executed.
Screen – 3/ Transformation chain selection

In the third step, all the possible chains that satisfy the user request are shown.

To help users in the selection, each chain is characterized by different attributes, like the number of single transformations that will be executed, the **coverage** with respect to the source metamodel, **how many times** the chain has been already executed, and its average **execution time**. Once the chain is selected, then the user can trigger its execution and wait for the generated model.
Screen – 4/ Report about the executed chain

Once the chain has been executed some statistical data are stored in the system and shown to the user as report.

**Transformation Chain Wizard**

<table>
<thead>
<tr>
<th>Chosen Chain</th>
<th>Chain Details</th>
<th>Execution Counter</th>
<th>Execution Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chain 1</td>
<td>Grafcet2Petrinet1.0 → Petrinet1.0toXML → XMLtoPNML</td>
<td>5</td>
<td>200 ms</td>
</tr>
</tbody>
</table>
CONCLUSION AND FUTURE WORK
Discussion and Future work 1/

The main strengths of the approach are related to the possibility of chaining transformations that would be discarded if is considered only the notion of metamodel compatibility.

However, the approach we proposed can be enhanced in different directions:

Management of breaking and unresolvable changes:

With a breaking and unresolvable change (BUR) we might exclude some feasible transformation chains.

Managing BUR is complex and require user intervention also in “coupled-evolution” area.

(this is why we decided to do not manage BUR changes at a first stage and investigate them as a future work)
Discussion and Future work 2/

**Metamodelf similarity functions:**
Our implementation is independent from the similarity function in the sense that, when a new similarity function is considered to be more appropriate, then we have to refining only one method of a dedicated Java class.

**Missing the management of semantic aspects of the transformations:**
We are based only on structural aspect of transformations and metamodels. Dealing with semantics of model transformations and in MDE in general is complex and represents an interesting and challenging research direction to pursue for extending the proposed approach.

**Experimentation and evaluation:**
The approach has been validated by considering the transformations available in the ATL Zoo. To better assess the validity it is necessary to consider an extended set of data (whose definition is a complex task per se).
Applications

This work was been accepted to MODELS 2014 Conference


References


Link to the online web application: www.mdeforge.org