2012 IEEE International Symposium on Visual Languages and Human-Centric Computing, Innsbruck, Austria, Sept 30-Oct 4 2012, © IEEE CS Press

CONVErT: A Framework for Complex Model Visualisation and Transformation

Iman Avazpour and John Grundy
Faculty of ICT, Centre for Computing and Engineering Software and Systems,
Swinburne University of Technology, Hawthorn, VIC 3101, Australia
{iavazpour, jgrundy}@swin.edu.au

***Abstract*—Model Driven Engineering (MDE) has become a commonly used approach in software engineering. It promotes using models as primary artefacts and proposes methods for transforming them to desired software products. However, the specification of models and their transformations in MDE with current techniques is not user-friendly, due to excessive use of high level abstract models and textual representation of transformation languages. This paper briefly describes CONVErT, an approach and tool developed for user-centric transformation generation using concrete model visualisations.**

I. Introduction

Model Driven Engineering (MDE) promotes using models as primary artefacts and proposes methods for transforming them to different domains or different abstraction levels. To accomplish this, users are required to define high level abstractions (meta-model) of their models and specify transformations using textual representation of available transformation languages. Transformations usually include correspondences and relations between elements of participating Left Hand Side (LHS - the source) and Right Hand Side (RHS - the target) models that have to be specified on their abstract meta-models. The way these correspondences and relations are specified creates a pragmatic barrier for many users (average modellers). This is because meta-models are not user-friendly artefacts and often get very complex [1], [2]. In addition, textual representations are often hard to maintain, especially when dealing with large and complex models.

To improve understandability of the abstract notation for average users, previous techniques used concrete syntax in conjunction with abstract syntax (metamodels) [1], [5]. The approach presented here, however, uses the actual visual elements as parts of transformation specification (transformation rules). By-example approaches have also been used to eliminate the need for defining metamodels and input model abstractions [3], [4]. However, they do not integrate visualisation. Our approach allows the user to define visualisation for desired LHS and RHS models regardless of the abstract level of input models and use the defined visualisation for transformation generation.

II. CONVErT (CONcrete Visual AssistEd Transformation) framework

CONcrete Visual AssistEd Transformation (CONVErT) is the framework developed for concrete model transformation. CONVErT contains an integrated collection of techniques to support specification and generation of model transformations in a more user-centric manner. The intention in CONVErT's design was to use same transformation routines using drag and drop of elements, for the task of transforming input model examples to desired visualisations and then visualisation to visualisation.

In order to generate a transformation using CONVErT, users provide a set of source and target model examples to specify correspondences. They then use visual elements and model context to transform input models to visualisations by drag and dropping model elements on visual notations. The visualisations are transformation aware, i.e. they include transformation templates and abstractions for transforming the context to visual elements (and back). They also include the data required for rendering element's shape. Each visual element can therefore take the role of a transformation rule. Figure 1 depicts few examples of these visual elements.

Users can save visual elements when defined. Saving a visual element will result in its composing parts (transformation templates, abstractions and data) being recorded. The visual element will then be kept in a palette for custom defined visualisations or transformation rules (depicted by 2 in Figure 2.a). For example, a visual element which is the result of transforming a model attribute to java property can be saved and reused whenever such a transformation rule is required.

A number of transformation functions have been integrated into the framework to handle more complex transformation tasks. The user can drag them to the designer canvas and link elements to/from their input/output ports to form the desired transformation rule (an example is depicted in Figure 2.b where a merging function is being used for merging two values to create bar chart name). Each function has the template of the task to be performed encoded inside along with its reverse. The system uses the interaction of user with the visual representation of the function (drag and drop of element

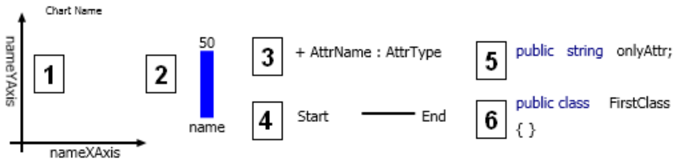
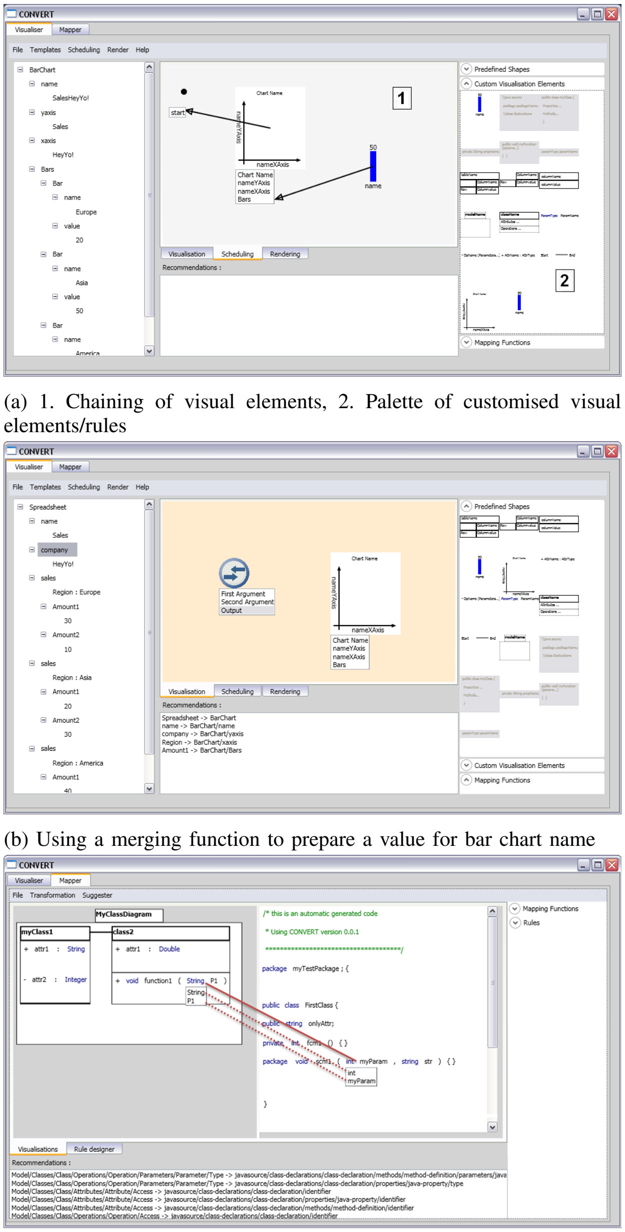


Fig. 1. Examples of visual elements: 1. A chart, 2. A bar, 3. UML attribute, 4. UML association link, 5. Java property and 6. Java class notation.



(a) 1. Chaining of visual elements, 2. Palette of customised visual elements/rules

```
Spreadsheet -> BarChart
name -> BarChart/name
company -> BarChart/yaxis
Region -> BarChart/xaxis
Amount1 -> BarChart/Bars
```

(b) Using a merging function to prepare a value for bar chart name

```
/* this is an automatic generated code

 * Using CONVERT version 0.0.1

 ************************************/

package myTestPackage ; {


public class FirstClass {

public string onlyAttr;

private int fcm1 () { }

package void scm1 ( int myParam , string str ) { }


}
```

```
Model/Classes/Class/Operations/Operation/Parameters/Parameter/Type -> javasource/class-declarations/class-declaration/methods/method-definition/parameters/java
Model/Classes/Class/Operations/Operation/Parameters/Parameter/Type -> javasource/class-declarations/class-declaration/properties/java-property/type
Model/Classes/Class/Attributes/Attribute/Access -> javasource/class-declarations/class-declaration/identifier
Model/Classes/Class/Attributes/Attribute/Access -> javasource/class-declarations/class-declaration/properties/java-property/identifier
Model/Classes/Class/Attributes/Attribute/Access -> javasource/class-declarations/class-declaration/methods/method-definition/identifier
Model/Classes/Class/Operations/Operation/Access -> javasource/class-declarations/class-declaration/identifier
```

(c) Correspondence specification between two visualisation (Class Diagram and Java code)

Fig. 2. Screenshots of CONVERT.

on/from function) to complete the template code required for the task. Reverse generation of a function is however limited to the tasks that are reversible. If not, a default value will be provided.

For a complete visualisation (and hence transformation specification), a visual element (transformation rule) chaining and scheduling mechanism has been applied that allows the user to compose multiple elements to generate more complex ones, yet only use drag and drop approach (indicated by 1 in Figure 2.a where a bar chart is being developed by chaining its composing elements). Each rule in the chain is represented by a visual element which is representation of the rule's resulting model. Visual elements in the chain contribute to composing a metamodel for the chain with their abstraction data. This metamodel acts as a dynamic example of the resulting model (which will be updated as the chain is altered) and will be used for transformation validation and to alert the user if the resulted model does not conform to the metamodel learnt from input examples. It is also possible to visualise the resulting (partial/complete) model so that the user can see a snapshot of the result as the chain completes. The defined rule chain together with the generic metamodel learned from the examples are used to generate reverse transformations, so the transformation cycle (source to source visualisation to target visualisation to target and back) is automatically defined once the user defines one direction. Users can generate and save multiple redundant rules, but only the rules in the chain will be used for code generation and since they are checked and validated (both visually and against metamodel) the system will not allow redundant results.

Once the visualisation for input source and target models are defined (e.g. in form of Charts, Source code, Boxes and lines, etc.), correspondence specification between visualisations will be performed with simple drag and drop of visual elements. Figure 2 part (c) shows an example where a visualisation of a class diagram and java source code have been rendered. For example, the user can drag and drop a parameter in the function declaration of a class diagram, to a parameter on a function in java code (showed by red solid line) to start generating UMLClass.Function.Parameter to JavaClass.Function.Parameter transformation rule. The contained elements of the parameters (showed in popups) can be dragged accordingly (depicted by dashed red lines) to form a complete transformation rule. If the rule requires more complex processing, user can transfer elements to designer canvas and use the defined functions to complete rule generation.

III. Conclusion

This paper introduced our approach to improve user-friendliness of model transformation specifications. It briefly described CONcrete Visual AssistEd Transformation (CONVErT) which is our framework for providing user-centric transformations and is capable of generating transformation code in XSLT. The users of CONVErT will use drag and drop of visual notations to develop complex transformation specifications.

References

[1] R. Gronmo, "Using concrete syntax in graph-based model transformations," Ph.D. dissertation, University of Oslo, Norway, 2010.

[2] G. Kappel, E. Kapsammer, H. Kargl, G. Kramler, T. Reiter, W. Retschitzegger, W. Schwinger, and M. Wimmer, "Lifting metamodels to ontologies: A step to the semantic integration of modeling languages," in *MoDELS'06*. Springer, 2006, pp. 528–542.

[3] M. Strommer, M. Murzek, and M. Wimmer, "Applying model transformation by-example on business process modeling languages," in *ER'07*, 2007, pp. 116–125.

[4] D. Varró, "Model transformation by example," in *MoDELS'06*. Springer, 2006, pp. 410–424.

[5] E. Visser, "Meta-programming with concrete object syntax," in *GPCE'02*. Springer, 2002, pp. 299–315.