In Proceedings of the 2013 IEEE Symposium on Visual Languages and Human-Centric Computing © IEEE 2013

Using Concrete Visual Notations as First Class Citizens for Model Transformation Specification

Iman Avazpour, John Grundy
Centre for Computing and Engineering Software and Systems
Faculty of ICT, Swinburne University of Technology, Hawthorn 3122, VIC, Australia
Email: {iavazpour, jgrundy}@swin.edu.au

***Abstract*—Model transformations are an important part of Model Driven Engineering (MDE). To generate a transformation with current MDE approaches, users are required to specify (or provide) complex meta-models and then engage in quite low-level coding in textual transformation scripting languages. This paper introduces a new approach to visualising source and target models that allows specifiers of complex data transformations to use the resultant visual notations for specifying transformations by example using drag and drop. We demonstrate the applicability of our new approach by an example case study.**

I. Introduction

Model Driven Engineering (MDE) promotes the development, maintenance and evolution of software by performing transformations on models. Transformations are therefore an integral part of MDE [1]. Current techniques for specifying transformations introduce high barriers for non-expert users, since they require users to learn and use complex meta-models and engage in low-level coding in complex transformation languages [2]. This problem has pushed both research and practice towards Model Transformation By-Example (MTBE) approaches rather than traditional programmatic solutions[3]. Users of such approaches provide multiple examples of source and target models and specify correspondences between them. The system then tries to infer meta-models and, in some approaches, possible model transformation rules. However, to use such approaches users must familiarise themselves with the syntax of the correspondence specification language. This is often problematic as no visual approach for specifying correspondences on actual familiar notations exists [3]. Also, since the rule derivation is semi-automatic in some MTBE approaches, multiple example pairs need to exist so the system can infer more accurate transformations [4].

This paper describes our research that takes the by-example approach a step further by using concrete visual notations representing source and target model examples. It addresses the complexity of model transformation by providing familiar visual notations to end users and allowing users to specify mapping correspondences between visualised model elements using a drag and drop approach. These specified-by-example correspondences thus form transformation rules, and are hence used as transformation specifications for low-level transformation script generation. In this approach, model visualisation examples are not required to represent the same underlying model data formats, i.e. they do not need to be in pairs.

II. Our approach

We use a three step procedure for specifying and generating complex model visualisations in our approach, as in figure 1. Step one is to design and create required notational elements to be used in the visualisation. These notational elements form a notation repository and can be reused. Step two is mapping examples of model input data to the reusable notational elements. A range of visualisation functions and conditions are available in our tool to be used in this step to create more complex data-to-visual notation mappings. Step three is to compose these model element-to-notational element mappings together to create complete visualisations for models that conform to the examples used. Once visualisations for both the source model and the target model have been created through these three steps, examples of these models can be visualised and transformation rules between the models are then created. This is done by drag and dropping concrete visual notational representations of the model elements between the two model visualisations.

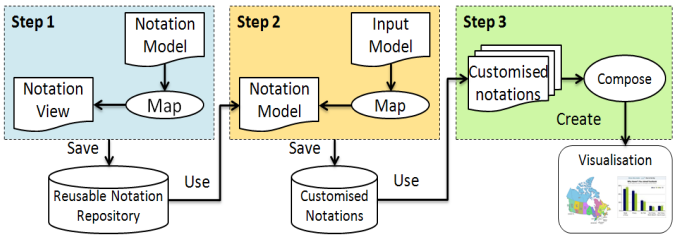


Fig. 1. Three steps of our visualisation approach.

This model-to-visualisation specification and model-to-model specification using concrete, by example approach, has been implemented in our CONcrete Visual assistEd Transformation (CONVErT) framework. CONVErT generates eXtensible Stylesheet Language Transformation (XSLT) implementations for both model-to-visualisation and model-to-model transformations, and includes a transformation engine capable of running these transformations.

In the following subsections the three steps of our approach are described using demonstrative example of creating a simplified version of the famous Minard's map (figure 2) visualisation. We show how portions of the underlying map model data, representing troop movements, can be transformed to a pie chart visualisation with our approach. For simplicity, we have omitted temperature information and have used locations relative to the map Canvas instead of actual GPS coordinates. Due to space limitations, the whole visualisation specification procedure for the pie chart is not provided. The full worked

specification of this visualisation, and further demonstrations with the latest release of CONVErT, are all available from our website [5].

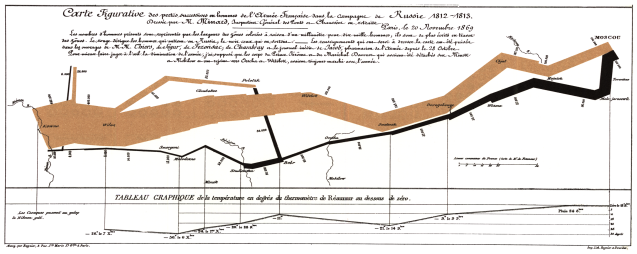
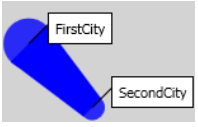


Fig. 2. Minard's map, depiction of French army's invasion of Russia.

1) Specify Basic Notational Elements for Reuse: To generate arbitrary visual or textual notations, we follow a Model View Controller (MVC) approach [6]. Visual (or indeed textual) notations in our approach are represented by a collection of shapes, text and graphics that define the View of the notation (which can be provided separately by a designer). A data part specifies what is to be represented by the notation (the Model). The Model needs to be mapped to the View; therefore, a direct mapping specification is provided (which acts as the Controller). We have designed a simple annotation language for defining mappings between this Model and View. It is composed of two annotations which specify the relationship type of Model elements with the View. It includes "linkto" for one-to-one mappings and "callfor" for one to many. Using these annotations a Controller transformation is generated to transform the Model data to the specific View.



(a) View

```
<TroopsData>
 <TroopsEntered>422000</TroopsEntered>
 <TroopsLeft>422000</TroopsLeft>
 <City1X>50</City1X>
 <City1Y>150</City1Y>
 <City1Name>Kawno</City1Name>
 <City2X>100</City2X>
 <City2Y>110</City2Y>
 <City2Name>Kawno East</City2Name>
 <Color>Red</Color>
</TroopsData>
```

(b) Data (Model)

Fig. 3. Troops movement notation

For the Minard map visualisation example, we follow a similar recreation approach to that used by Humphrey [7]. Our version of this visualisation uses two notations. The first notation depicts troop movements from one position to another. Based on coordinates and the number of troops at the starting point and the number arriving at the destination, a shape will be drawn as specified in the notation View (figure 3(a)). The algebras for creating this shape are provided in [7]. CONVErT provides a way to specify and reuse this in C#. This notation requires a data part that represents start and destination location names and coordinates, number of troops starting, and number arriving at destination, and the color of the shape which represents whether troops were advancing or retreating. The data part for this notation will be similar to Figure 3(b). Elements of this model are all in one-to-one relation with notation attributes. Therefore, they will all be specified in the controller transformation using "linkto" annotations.

The second notation shows the map with its description on top (figure 4(a)). The map notation should host troop movement notations, therefore, when the user is annotating

(a) Map View

```
<MapData>
<Description>Discription goes here</Description>
<Movements>Troop Movements</Movements>
</MapData>
```

(b) Map Model

```
<StackPanel Orientation="Vertical" Height="290" Width="716">
 <TextBlock Height="27" TextAlignment="Center" Background="White">
  <TextBlock.Text linkto="Description">Description of the map
    </TextBlock.Text>
  </TextBlock>
 <Canvas Height="263" Width="716">
  <Canvas.Background>
   <ImageBrush ImageSource="NapoleonMap.bmp" />
  </Canvas.Background>
  <local:Minard Height="263" Width="716" callfor="Movements">
  </local:Minard>
 </Canvas>
</StackPanel>
```

(c) Annotated notation View

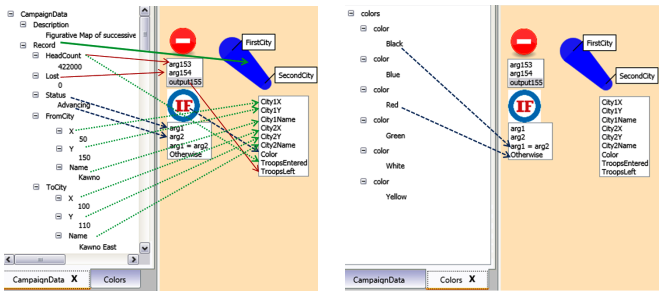
Fig. 4. Annotating Map notation's View.

the provided View, the "callfor" annotation is provided in the View according to Figure 4(c). The Minard class declared in XAML is derived from the Canvas and allows for hosting of other visual elements.

2) Map Source Data to Visual Notation: Step two involves mapping example model data to the specified notational elements. This step is used to provide the system with information in order to create transformation rules for transforming specific parts of input models to the notation's Model data. By default a tree-like representation of the input model data (for XML and Comma Separated inputs) is provided for users so that they can drag elements of that representation to the provided notation. For example in our Minard map visualisation, the provided input data to be visualised is an XML data file (the input model) which includes a map description and a list of troop movement records which include start and destination location names and coordinates, number of troops starting and lost during the journey, and a status string which defines whether they were advancing or retreating. These inputs can be in multiple files, for simplicity of description however, we have merged input files in a single file.

To generate the transformation rule for visualising each movement record as a movement notation, users need to drag and drop a record element from input to a troops movement notation. Figure 5 shows mapping specification of records to troop movement notation. Considering input data and the annotation's Model data, we can conclude that coordinates, name of locations, and number of troops at start are in a one-to-one relationship with their corresponding elements on the notation's Model data. Therefore their correspondences can be specified by direct drag and drop of input data elements on the notation's Model elements as shown by green arrows in figure 5(a). However, the notation requires the number of troops at the destination, whereas the input data record provides number of troops lost during the journey. Also the status of the movement is declared by *Advancing* or *Retreating* strings in the input while this has been defined by colours in the notation.

To specify these correspondences, a set of reusable functions and conditions are provided by CONVErT to allow generation of more complex model element-to-visual notation mappings. For example, troops movement status can simply be defined by using a condition which provides a colour according



(a) Correspondences (b) Colour condition

Fig. 5. Specifying correspondences between troop movement records and troop movement notation. Arrows indicate drag and drop directions.

to the status string. To generate this, the CONVErT user drops a condition on the Canvas and links corresponding elements, as shown in figures 5(a) and 5(b), by the navy dashed arrows. Colour names are provided as a separate input file. Similarly, for specifying the number of troops at the destination, the user is provided with a subtraction function which subtracts troops *Lost* from number of troops starting (*HeadCount*) to calculate the required value. The user then maps its output to troops arriving at the destination element (red arrows in figure 5(a)).

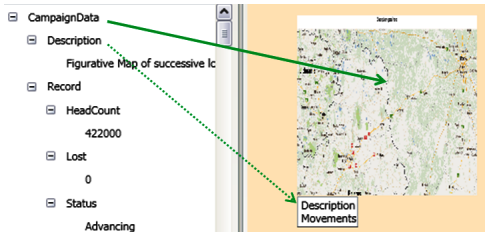


Fig. 6. Specifying correspondences between input data and map notation.

Specification of the map notation is more straight forward. User need to drag a campaign data element from the input model on the map and link its description to the *description* element of the map notation, as shown by figure 6. The movements element is the place holder for troop movement notations which will be linked in the notation composition step. Once data mapping is complete in CONVErT, the visual notation specification for these model elements are saved. This results in creation of a customised notation and a transformation rule that transforms input model to the notation's Model (and its reverse where possible).

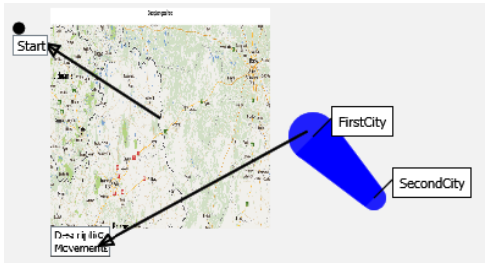


Fig. 7. Composing troop movement and map notations to generate complete visualisation. Arrows are provided by CONVErT.

3) Compose Basic Notational Elements: Step three allows users to link, combine and embed visual individual model data-to-visual notation mappings in order to create a complete visualisation for a whole model. Linking a notation to an element results in scheduling of model element-to-visual notation transformation rules, since each visual element also hosts a transformation rule for transforming a portion of input data to the notation's Model data. In our example, the two notations can be composed by linking the movement notation and the map notation, as shown by figure 7. Linking a notation to a *start* element will define the top-most (first to be run) transformation rule for the full model transformation specification. The generated transformation is then applied to all input data to produce a full map visualisation (figure 8).

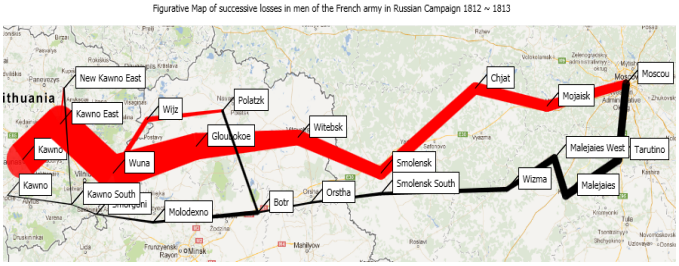


Fig. 8. Resulting visualisation of Minard's map in CONVErT.

Once visualisations for both source and target models are available, users can then drag and drop elements between these model visualisations to form model-to-model transformation rules. These mappings are then used to generate a complete model transformation script that can transform examples of models of the source model format to models of the target model format (for examples see [8] and CONVErT website [5]). In the map visualisation example, let's assume that there is a requirement for visualising the number of troops lost during the campaign at each key location as a pie chart, transformed by-example from this map visualisation. Note that the pie chart does not have to posses similar information and here it represents a sales records of a company. To perform this data mapping, CONVErT allows us to view both source (map) and target (pie chart) concrete visualisations side-by-side. Here we need to generate a rule for transforming the map to a chart and a rule for creating a pie piece from each troop movement notation. To generate the first rule, the user is required to drag and drop map notation on the chart area. Right clicking on notations in these visualisations reveals their internal elements. The internal elements of the map correspond to elements of the chart, therefore, they should be linked as well. Figure 9 shows these correspondences by green arrows. Saving these will generate a map to chart transformation rule.

To generate the pie piece notation from a troop movement notation, user drags a movement notation onto a pie piece. Each pie piece includes a value, a name and a color. The Minard's map visualisation does not include a specific data element for number of troops lost at each movement. Therefore this value needs to be calculated from available data in the visualisation using CONVErT's mapping functions. Similar to visualisation step, mapping functions are available for mapping generation between visualisations. Once the user drags a visual notation onto another, CONVErT provides a default View for those notations in a separate window (see figure 10). Here the we have used a subtraction function for calculating the value of troops lost during the journey and a merging function which merges two city names and includes a " to " between them to generate the name of each pie piece. One-to-one

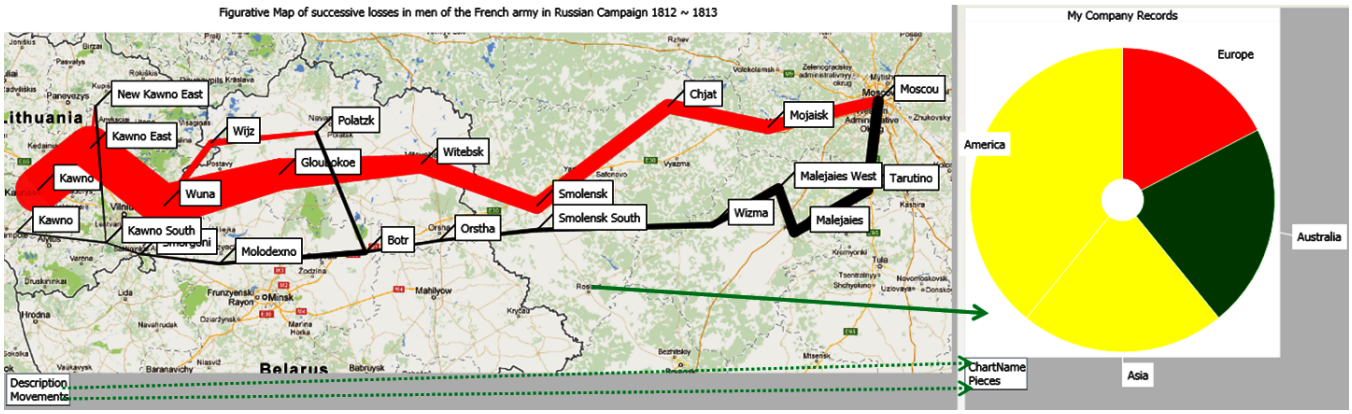


Fig. 9. Specifying correspondences between map and chart in concrete visualisations. Arrows depict drag and drop direction.

correspondences (like Color) can be defined here or on actual visualisations.

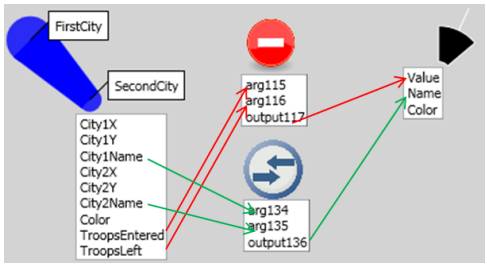


Fig. 10. Specifying number of troops lost and location names, using CONVErT mapping functions.

Once the two mappings are defined, a meta-model is automatically reverse engineered from both source and target visualisations. Using this meta-model, CONVErT knows how to compose/schedule transformation rules automatically. Once done, CONVErT generates the transformation specification and the resulted visualisation. The resulting visualisation of this example is depicted in figure 11.

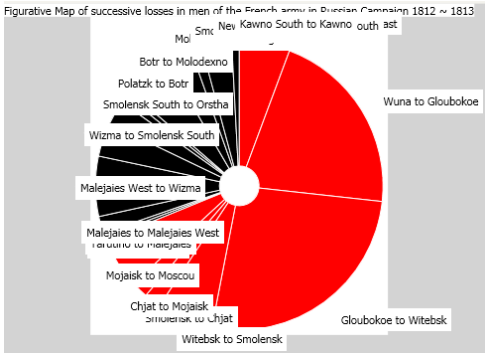


Fig. 11. The resulting Pichart.

III. Conclusions and Future Work

We have described a new concrete visual approach to model transformation specification. Users of our approach specify transformation rules by drag and drop of elements in concrete visualisations generated from example underlying data. A proof of concept tool implementing this approach, CONVErT, has been prototyped and evaluated by a demonstrative case study and user evaluation. Our user study revealed that by allowing users to define correspondences on source and target visualisations, they were capable of easily spotting correspondences and creating transformation rules. The full results of this evaluation are available in CONVErT's website [5].

CONVErT uses the default layout mechanisms of WPF and XAML for visualisations. We are working on integrating better layout algorithms to provide more powerful and flexible layout support. For large scale models, visualisations may become crowded and complex. New mechanisms are being tested to filter visualisations and allow partial visualisations according to users' queries. For bijective transformations our approach generates a reverse translation automatically. Fully round-trip transformations are not possible in non-bijective cases e.g. where conditions are used. Similarly, lossy transformations are not as yet supported.

Acknowledgments

Avazpour was supported by Swinburne University Vice Chancellor's Research Scholarship (VCRS). We thank Dr Vivienne Farrell for help with questionnaire and user studies.

References

[1] R. France and B. Rumpe, "Model-driven development of complex software: A research roadmap," in *2007 Future of Software Engineering*, ser. FOSE '07. IEEE, 2007, pp. 37–54.

[2] J. C. Grundy, J. G. Hosking, R. Amor, W. B. Mugridge, and Y. Li, "Domain-specific visual languages for specifying and generating data mapping systems," *J. Vis. Lang. Comput.*, vol. 15, no. 3-4, 2004.

[3] G. Kappel, P. Langer, W. Retschitzegger, W. Schwinger, and M. Wimmer, "Conceptual modelling and its theoretical foundations," A. Düsterhöft, M. Klettke, and K.-D. Schewe, Eds. Springer-Verlag, 2012, ch. Model transformation by-example: a survey of the first wave, pp. 197–215.

[4] M. Kessentini, H. Sahraoui, M. Boukadoum, and O. Omar, "Search-based model transformation by example," *Software & Systems Modeling*, vol. 11, pp. 209–226, 2012.

[5] I. Avazpour and J. Grundy. Convert website. [Online]. Available: https://sites.google.com/site/iavazpour/tools-manuals

[6] T. Reenskaug, "Models-views-controllers," *Technical note, Xerox PARC*, vol. 32, p. 55, 1979.

[7] M. C. Humphrey, "Creating reusable visualizations with the relational visualization notation," in *Proceedings of the 11th IEEE Visualization 2000 Conference (VIS 2000)*. IEEE, 2000, pp. 53–60.

[8] I. Avazpour and J. Grundy, "Convert: A framework for complex model visualisation and transformation," in *2012 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, 2012, pp. 237–238.