Tool Support for Automatic Model Transformation Specification using Concrete Visualisations

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

Lars Grunske
Institute of Software Technology
Universität Stuttgart
Universitätsstraße 38, D-70569 Stuttgart, Germany
lars.grunske@informatik.uni-stuttgart.de

***Abstract*—Complex model transformation is crucial in several domains, including Model-Driven Engineering (MDE), information visualisation and data mapping. Most current approaches use meta-model-driven transformation specification via coding in textual scripting languages. This paper demonstrates a novel approach and tool support that instead provides for specification of correspondences between models using concrete visualisations of source and target models, and generates transformation scripts from these by-example model correspondence specifications.**

I. Introduction

Successful application of Model-Driven Engineering (MDE) relies on model transformations. In MDE, development, maintenance and evolution of software is performed by transformations on models. With current MDE approaches, specification of these transformations is performed using textual scripting languages and abstract representations of meta-modelling languages. These textual representations are hard to specify and maintain for non-software engineering users [1], and although abstractions provide better generalisation and thus code reduction, their syntax is often far removed from the actual model syntax [4]. As a result, current approaches introduce pragmatic barriers for non-software engineering users to adopt MDE as their main approach. Moreover, the variety of models being used in today's software systems and their often large scale, adds to the complexity of transformation specification and maintenance, even for expert users [1], [2].

To address this complexity, several approaches have been developed including use of visual abstractions (e.g. [3]), by-example transformations [4], graph transformations (e.g. [5]), automatic inference of bi-directional transformations (e.g. [6]), and automated assistance for mapping correspondence deduction (e.g. [7]). However, none of these fully address the problems nor do so in an integrated and highly extensible way.

This paper demonstrates the approach implemented in our CONcrete Visual assistEd Transformation (CONVErT) framework [8]. CONVErT provides users with familiar concrete visualisations of source and target models in order to leverage their domain knowledge in transformation specification. As a result, users can specify complex model element mappings between concrete visual notational elements using interactive drag-and-drop and reusable, spreadsheet-like mapping formulae. It also automatically creates high-level abstractions for transformation generation from the concrete visualisations. Using this abstraction and model examples, CONVErT helps users decide, find and explore possible model correspondences by providing automated recommendations using an interactive recommender system. Complex, scalable and reusable model transformation implementations will then be generated from these visually specified, by-example model mappings. Following sections better describe CONVErT's approach and implementation architecture.

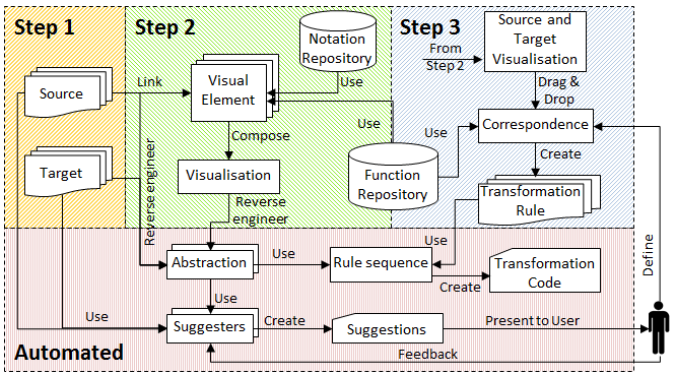

Fig. 1. CONVErT's approach and usage scenario.

II. CONVErT's approach and usage example

CONVErT's approach to model transformation is outlined in Figure 1. It consists of three steps: 1) Providing source and target model examples to the framework. 2) Creating (or reusing) concrete visualisation for the provided examples. 3) Using these model visualisations, specifying source and target mapping correspondences, which includes 1-to-1, 1-to-many, many-to-1 and many-to-many element correspondences.

Input examples can be provided to CONVErT as Comma Separated Value (CSV) or XML files. CONVErT uses a graph lattice to reverse engineer an abstraction (meta-model) from provided examples. A crawler analyses the inputs and inserts the new structures and elements it faces to the lattice. This abstraction is used for transformation script generation and in calculating recommendations of mappings to users.

CONVErT's guidance mechanism uses a group of similarity heuristics (we call "Suggesters") to calculate similarities between source and target model elements. These suggesters

978-1-4799-0215-6/13 © 2013 IEEE

check source and target models for value, name tag and structural similarities. Each suggester returns a similarity matrix representing similarity scores between elements of source and target models. The recommender system then analyses these scores and prepares a list of recommendations according to the confidence score assigned to each suggester. If a recommendation is selected by the user, a feedback loop increases the confidence score of the suggester(s) which came up with that recommendation. If it is otherwise rejected, the confidence scores will be updated accordingly. Due to performance issues of structural similarity suggesters, the reverse engineered abstraction of input models are used for similarity calculation. Other suggesters use actual values of the input models.

If a visualisation has been previously defined, it can be reused. If not, users can specify visualisations by linking model elements to provided visual notations. These notations can be generated by designers and be adopted according to the application domain. CONVErT provides facilities for drag and dropping input elements to visual notations to specify model to visualisation correspondences. It also examines the input model and the notation, and provide series of automatically-generated recommendations to the user.

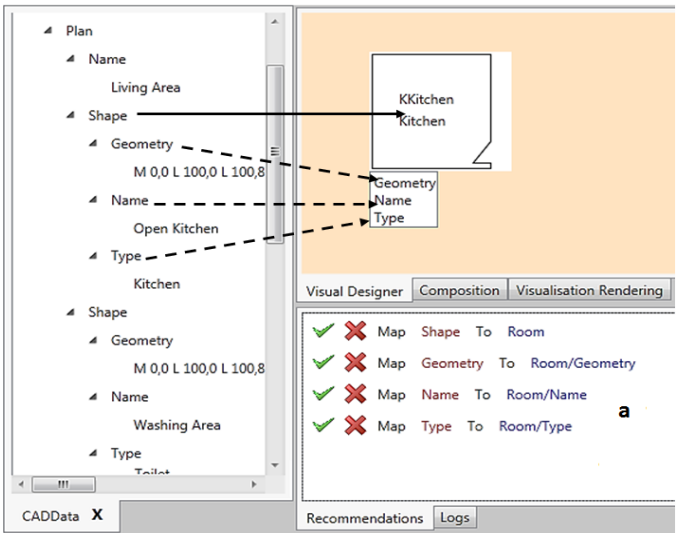

Fig. 2. Mapping example elements to notation elements. Arrows depict drag and drop directions.

For example, consider a Computer Aided Design (CAD) application in which architect Carrie needs to create a new visualisation for a provided design model. To specify correspondences between shapes of the input model and a Room notation she drags and drops shape element of the input model on a room notation (see solid black arrow in Figure 2) or if it has been recommended, selects it in the recommendation list (see *a* in Figure 2). CONVErT analyses this interaction and triggers a transformation rule for transforming that portion of the input model (Shape element in XML input) to the room notation's model. Each notation may have internal elements which can be viewed in a pop-up window by right clicking on the notation. For example, the room notation of Figure 2 has a Geometry, Name and Type. These internal elements can also be linked by drag and dropping elements or choosing from recommendations. These correspondences will be included in the transformation rule template that has been triggered.

The notations that are defined in CONVErT represent a model element-to-visual notation transformation rule. CONVErT uses these notations to generate a model-to-visualisation transformation script. To have a complete transformation script, the prepared collection of defined notation rules should be scheduled according to their call sequence. Usually, this is achieved by asking users to write code for this script, similar to procedural programming. In our approach however, CONVErT can generate this scheduling and the transformation code by user composition of notations.

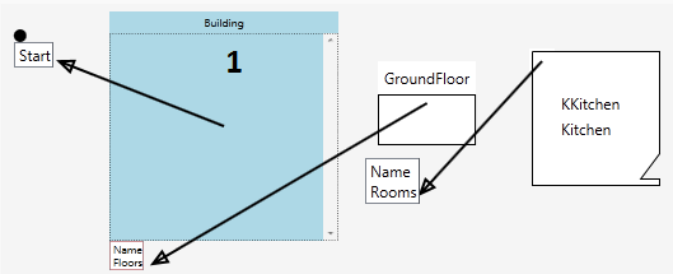

Fig. 3. Composition of notations for CAD visualisation.

Composition of notations specifies how a full model visualisation is formed from a set of sub-element visualisations. For instance in the CAD example, assume Carrie has defined visual notations to represent building, floor and room. She can compose these defined notations by linking them according to their specific place holder elements as depicted by Figure 3. These place holders are notation elements which will be replaced by calls to transformation rules of other notations. For example in composition of Figure 3 since the room notation is linked to *Rooms* element of floor notation, a call to the transformation rule embedded in the room notation will be placed in floor notations *Rooms* element.

The notation mapped to *Start* element in Figure 3 defines the transformation rule that has to be called first. For example in Figure 3, the transformation rule in building notation (marked by *1*) is the first rule to be called. It then calls the floor transformation rule, and the scheduling continues accordingly for other linked notations. This linking results in the scheduling of model element-to-visual notation transformation rules and thereby CONVErT can generate the model to visualisation transformation script.

The transformation script generated by CONVErT for each composition generates Windows Presentation Foundation (WPF) visual elements. For example, by using the compositions specified in Figure 3, a complete XSLT script to generate concrete visualisations of CAD models will be generated for rendering those model examples to visualisations similar to the visualisation of Figure 4. Note that the generated XSLT transformation script can be reused and applied to all examples

of the CAD input to provide an automatic concrete visual notation renderer. These generated concrete 2D visualisations are implemented as WPF elements and allow interaction with their composing notations. The individual elements of a concrete visualisation can be dragged, dropped on other elements, and right clicking them reveals their internal elements.

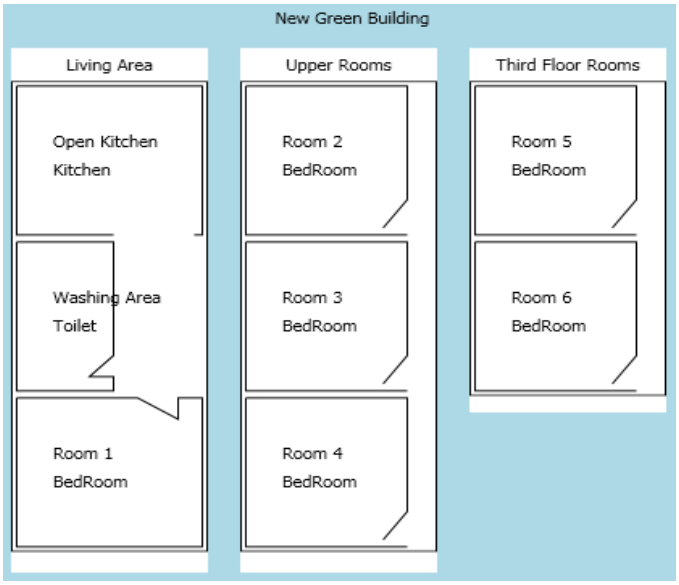

Fig. 4. Example of a generated CAD visualisation.

Once visualisations of both source and target models are available, users can generate transformation rules between source and target models by (a) drag and dropping elements in those visualisations or (b) selecting from provided recommendations. CONVErT analyses user interactions and either creates new transformation rules or inserts those correspondences into other transformation rules.

Figure 5 shows an example of creating a transformation rule for mapping a 2D room shape (source model visual notation) to a building structure node (target model notation). To create this rule, architect Carrie needs to drag a room notation element to a building node notation element, as depicted by solid black arrow, and match their internal elements, as shown by dashed black arrows (or select them in automatically suggested recommendations). By dragging one notation to another, CONVErT's automatic recommender updates the list of suggested correspondences to provide suggestions related to that rule and hence better guide users with targeted recommendations. For example, if Carrie defines a 2D room to tree node rule (by either drag and drop or selecting from suggesters), the suggestion list will be updated to demonstrate how internal elements of these notations (like Name, Type, Geometry and Color) can be linked (see *a* in Figure 5). This intelligent assistance greatly helps in mapping large models and models with many correspondences and complex visualisations.

A wide variety of mapping functions are available in CONVErT (see *b* in figure 5) to specify more complex correspondences in both model-to-visual notation and visual-

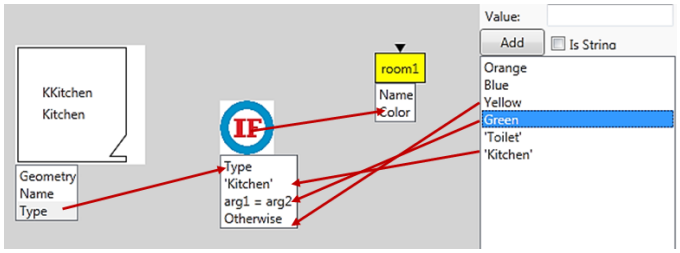

Fig. 6. Using conditions to map 2D room notation to room node notation of a structure chart. Arrows depict drag and drop direction.

to-visual transformation rules. For example, in the mapping of room notation to structure node notation of Figure 5, room notations have explicit *Type* attribute whereas building nodes representing rooms in floors are color-coded. To generate these colors, Carrie can use condition functions and specify colors based on types. By dragging a room notation to a room node (or any notational element to another) their default notation representations will be shown on a separate view to better provide space for using functions. In this example, Carrie can navigate to that canvas and specify conditions as depicted by Figure 6. The condition function in the Figure tests whether room's *Type* is equal to for example *Kitchen*. For this situation, Carrie can specify different colors (in this case Green) and drag it to the condition expression and specify what color to be used otherwise. The value provided by the condition will be then assigned to the element of the target (in this case tree node's color) as depicted by arrows in Figure 6.

CONVErT uses a concrete representation for transformation rules. Saving a transformation rule between visualisations results in creation of a concrete representation for that rule based on source and target elements being involved in the transformation. For instance in the example above, the visual representation of the rule transforming a 2D room to a tree node is marked by *c* in Figure 5.

Due to use of automatically generated abstractions in transformation rule templates, the two visualisations do not need to represent the same data. For example in Figure 5 the CAD visualisation represents a green building design but the tree representation is for a city council building. However, visualisations of the same dataset will help CONVErT's recommender system to produce more accurate recommendations.

Once the required rules for transforming all parts of source and target visualisations are defined, CONVErT uses these abstractions to schedule rules and generates the transformation script. Applying this script to source visualisations will transform the data represented by source to visualisation of the target. For example, in the CAD example, applying the full transformation script on the source of Figure 5 will result in the visualisation of Figure 7.

III. Summary

This paper demonstrated the use of CONVErT for specifying and generating complex model transformations using concrete model visualisations. CONVErT provides facilities

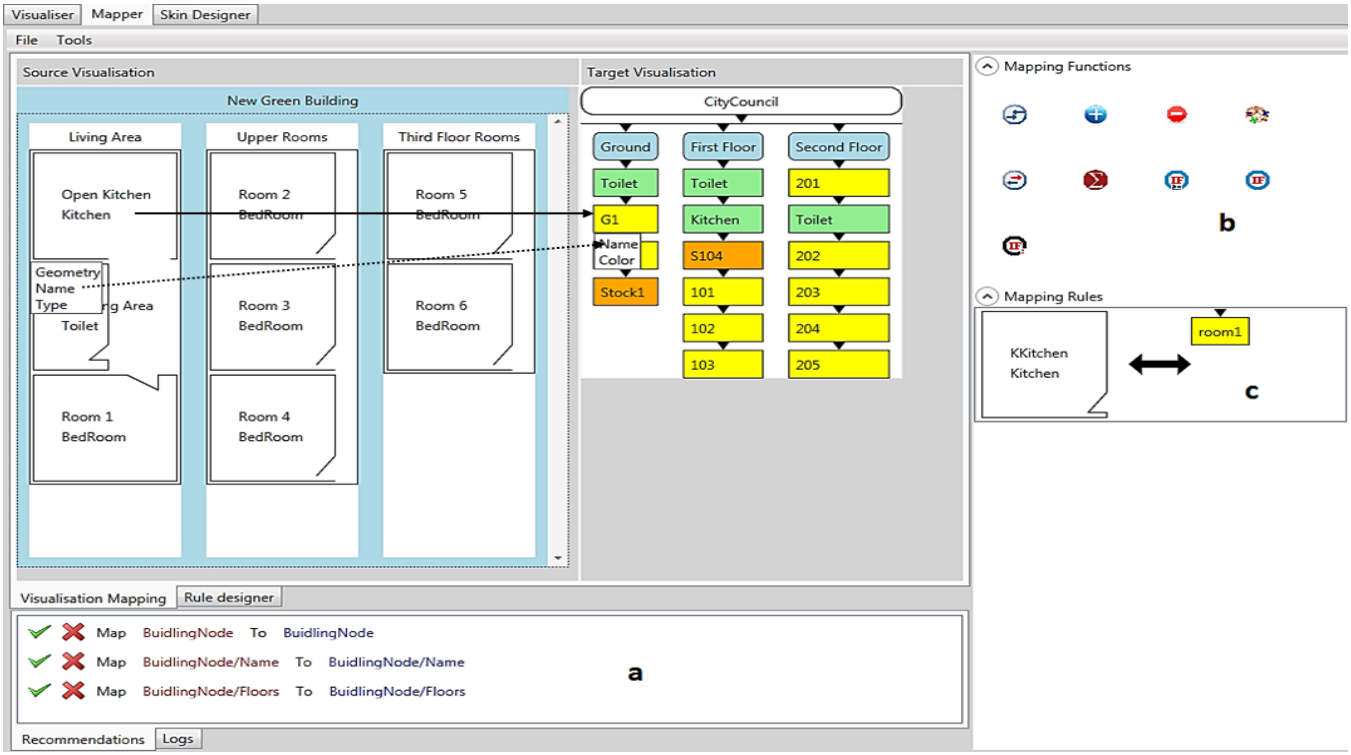

Fig. 5. Using CONVErT to transform 2D CAD drawings to a tree-based layout.

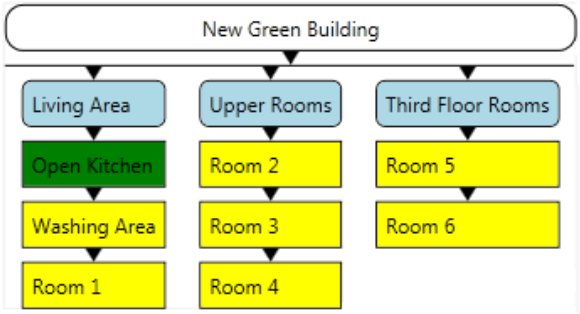

Fig. 7. Resulting tree structure chart.

for specifying correspondences on concrete and familiar visualisations of source and target models. It reverse engineers required abstraction from model examples and provides recommendations for possible source and target correspondences. As a result, the required knowledge and skills for performing transformations is reduced. CONVErT has been evaluated by a user study involving twelve participants from staff and students at Swinburne university. These participants were divided into two groups and were asked to create a visualisation for a given model example and then use their visualisation as source to transform it to another model visualisation. Experiment set-ups were the same, however, each group used different models and visualisations. Results of this study revealed that participants liked and were more effective using the concrete visualisation and drag and drop approach to specifying correspondences and transformation rules. Full results are on our website [9].

Acknowledgments

Avazpour was supported by Swinburne University Vice Chancellor's Research Scholarship (VCRS). We thank Dr Markus Lumpe, Dr Robert Amor, Ayman A. Amin and anonymous reviewers for constructive comments.

References

[1] 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.

[2] 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.

[3] Y. Sun, J. White, and J. Gray, "Model transformation by demonstration," in *Model Driven Engineering Languages and Systems*, ser. Lecture Notes in Computer Science, A. Schürr and B. Selic, Eds. Springer Berlin Heidelberg, 2009, vol. 5795, pp. 712–726.

[4] A. Raffio, D. Braga, S. Ceri, P. Papotti, and M. Hernandez, "Clip: a visual language for explicit schema mappings," in *Data Engineering, 2008. ICDE 2008. IEEE 24th International Conference on*, 2008, pp. 30–39.

[5] G. Rozenberg and H. Ehrig, *Handbook of graph grammars and computing by graph transformation*. World Scientific London, 1999, vol. 1.

[6] S. Hidaka, Z. Hu, H. Kato, and K. Nakano, "A compositional approach to bidirectional model transformation," in *Software Engineering - Companion Volume, 2009. ICSE-Companion 2009. 31st International Conference on*, 2009, pp. 235–238.

[7] S. Bossung, H. Stoeckle, J. Grundy, R. Amor, and J. Hosking, "Automated data mapping specification via schema heuristics and user interaction," in *Proceedings of the 19th IEEE/ACM International Conference on Automated Software Engineering.*, 2004, pp. 208–217.

[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.

[9] I. Avazpour, J. Grundy, and L. Grunske. CONVErT website. [Online]. Available: https://sites.google.com/site/iavazpour/tools-manuals