Journal of Visual Languages and Computing 28 (2015) 195–211

Contents lists available at ScienceDirect

Journal of Visual Languages and Computing

journal homepage: www.elsevier.com/locate/jvlc

Specifying model transformations by direct manipulation using concrete visual notations and interactive recommendations ☆

Iman Avazpour [a,*], John Grundy [a,*], Lars Grunske [b]

[a] Centre for Computing and Engineering Software and Systems (SUCCESS), Swinburne University of Technology, Hawthorn 3122, VIC, Australia

[b] Institute of Software Technology, Universität Stuttgart, Universitätsstraß e 38, D-70569 Stuttgart, Germany

ARTICLE INFO

Article history:
Received 15 October 2014
Received in revised form
27 January 2015
Accepted 19 February 2015
Available online 27 February 2015

Keywords:
Model driven engineering
Model transformation
Visual notation
Recommender system
Concrete visualizations

ABSTRACT

Model transformations are a crucial part of Model-Driven Engineering (MDE) technologies but are usually hard to specify and maintain for many engineers. Most current approaches use meta-model-driven transformation specification via textual scripting languages. These are often hard to specify, understand and maintain. We present a novel approach that instead allows domain experts to discover and specify transformation correspondences using concrete visualizations of example source and target models. From these example model correspondences, complex model transformation implementations are automatically generated. We also introduce a recommender system that helps domain experts and novice users find possible correspondences between large source and target model visualization elements. Correspondences are then specified by directly interacting with suggested recommendations or drag and drop of visual notational elements of source and target visualizations. We have implemented this approach in our prototype tool-set, CONVErT, and applied it to a variety of model transformation examples. Our evaluation of this approach includes a detailed user study of our tool and a quantitative analysis of the recommender system.

1. Introduction

Model transformation plays a significant role in the realization of Model Driven Engineering (MDE). Current MDE approaches require specifying correspondences between source and target models using textual scripting languages and the abstract representations of meta-modeling languages. Although these abstractions provide better generalization, and hence code reduction, they introduce difficulties for many potential transformation specification users. This is because in order to effectively use them, users have to possess in-depth knowledge of transformation languages and meta-modeling language syntax. These are often very far removed from the actual concrete model syntax for the target domain. Moreover, taking into account large models being used in today's software systems, many transformation specifications are very complex and challenging to specify and then maintain, even for experienced transformation script and meta-model users [1–3]. Although some approaches have been developed to mitigate these problems, such as by using visual abstractions [4,5], by-example transformations [3,6,7], graph transformations [8–11], automatic inference of bi-directional

☆ This paper has been recommended for acceptance by Shi Kho Chang.

* Corresponding author.
E-mail addresses: iavazpour@swin.edu.au (I. Avazpour), jgrundy@swin.edu.au (J. Grundy), lars.grunske@informatik.uni-stuttgart.de (L. Grunske).

http://dx.doi.org/10.1016/j.jvlc.2015.02.005
1045-926X/

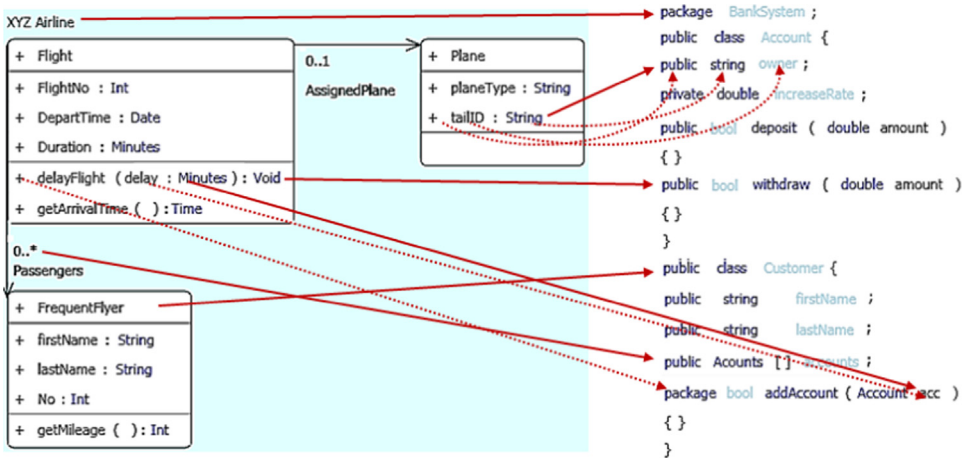

Fig. 1. Example of correspondence relations between a Class Diagram and Java code. Dashed arrows show more fine-grained correspondences.

transformations [12,13], and automated assistance for mapping correspondence deduction [14], none of these fully address the problems nor do so in an integrated, visual, human-centric and highly extensible way.

We introduce a new approach that helps to better incorporate user's domain knowledge by providing them with familiar concrete model visualizations for use during model visualization and transformation generation. This approach follows the three principles of *direct manipulation* [15], i.e. (1) it provides support for generating concrete visualizations of example source and target models; (2) these visualizations allow user interaction in the form of drag and drop of their concrete visual notation elements; and (3) interactions are automatically translated into transformation code and hence direct coding in complex transformation scripting languages is avoided. In addition, to better aid users in finding correspondences in large model visualizations, an automatic recommender system is introduced that provides suggestions for possible correspondences between source and target model elements. Complex model transformation code is automatically generated from the user's interaction with concrete visual notations and suggested recommendations.

This paper is organized as follows: Section 2 gives a motivating example, our key research questions and the requirements being addressed by the research reported in this paper. Section 3 briefly discusses key related work. Section 4 outlines our approach to model transformation generation followed by a usage example in Section 5. Section 6 describes the architecture and implementation of our approach in CONcrete Visual assistEd Transformation (CONVErT) framework. Section 7 describes our evaluation and user-study setup and is followed by a discussion in Section 8. Finally Section 9 concludes the paper with a summary.

2. Motivation

Assume Tom, a software developer, is working in an MDE-using team and has received a system analysis report for an application. Being an expert in UML diagram interpretation and a Java coder, he is familiar with concrete syntax of the diagrams and Java code. He is interested in transforming specific parts of UML diagrams provided by the analysis directly to his programming code, to increase team productivity, code quality and to ease software evolution. For example, he wants to create a model to code translator in order to transform specific features and parts in the analysis diagrams to specific Java code templates. For Tom, as an expert in the domain, corresponding elements in the UML diagram and in his Java code are obvious. He can clearly spot and relate classes, methods, and even statement snippets in both program code and class diagram. For example, he can easily relate an attribute in a class diagram to a property in Java code and their fine-grained elements (i.e. types, names and access identifiers). Some such model element correspondences are depicted in Fig. 1, using concrete visualizations of the UML model (a class diagram) and code model (Java textual syntax).

Fig. 2. Example of correspondence relations between geo-located bubbles and pie pieces in a pie chart.

As another example, consider Jerry, an urban planner, who is preparing a report on traffic congestion in part of a city. He is used to viewing volumes of vehicles crossing intersections on screen using a geo-spatial visualization. An Example of this visualization is shown on the left side of Fig. 2. Here, the volume of vehicles are represented on a map using bubbles. In his report, he would like to reflect the volume of vehicles passing set of intersections in a particular time instance by a pie chart. Being an expert in this domain, he has a solid understanding of this map-based visualization and pie charts and therefore their corresponding relationships are obvious to him. He would like to relate the number reflected to each bubble to a pie piece in a pie chart and generate new visualizations for his report as shown in Fig. 2.

Given that Tom and Jerry may not have experience or knowledge of transformation languages, meta-modeling,