Chapter 4
A WYSIWYG Approach to Support Layout Configuration in Model Evolution

Yu Sun
University of Alabama at Birmingham, USA

Jeff Gray
University of Alabama, USA

Philip Langer
Johannes Kepler University, Austria

Gerti Kappel
Vienna University of Technology, Austria

Manuel Wimmer
Vienna University of Technology, Austria

Jules White
Virginia Tech, USA

ABSTRACT

Model evolution has become an essential activity in software development with the ongoing adoption of domain-specific modeling, which is commonly supported and automated by using model transformation techniques. Although a number of model transformation languages and tools have been developed to support model evolution activities, the layout of visual models in the evolution process is not often considered. In many cases, after a transformation is performed, the layout of the resulting model must be manually rearranged, which can be time consuming and error-prone. The automatic layout arrangement features provided by some modeling tools usually do not take a user's preferences or the semantics of the model into consideration, and therefore could potentially alter the desired layout in an undesired manner. This chapter describes a new approach to enable users to specify the model layout as a demonstrated model transformation. We applied the Model Transformation By Demonstration (MTBD) approach and extended it to let users specify the layout information using the concept of "What You See Is What You Get" (WYSIWYG), so that the complex layout specification can be simplified.

DOI: 10.4018/978-1-61350-438-3.ch004

INTRODUCTION

With the ongoing adoption of Domain-Specific Modeling (DSM) (Gray et al., 2007), models are emerging as first-class entities in many domains and play an increasingly significant role in every phase of software development (i.e., from system requirements analysis and design, to software implementation and maintenance). In the DSM context, whenever a software system needs to evolve, the models used to represent the system should evolve accordingly. For instance, system design models often need to be changed to adapt to new system requirements (Greenfield & Short, 2004). As an additional example, it is sometimes necessary to apply model refactoring (France et al., 2003) to optimize the internal structure of the implementation models (i.e., models used to generate implementation code through code generators). Furthermore, models used to control the deployment of a software system are occasionally scaled up for the purpose of improving performance (Sun et al., 2009a).

Although manual model evolution is often tedious and error-prone, automating complex model evolution tasks using model transformation technologies has become a popular practice (Gray et al., 2006). A number of executable model transformation languages (e.g., QVT (http://www.omg.org/cgi-bin/doc?ptc/2005-11-01, 2010), ATL (Jouault et al., 2008)) have been developed to enable users to specify model transformation rules, which take an input model and evolve it to produce an output model automatically.

Open Problems

Although the implementation of model evolution concerning the abstract syntax has been well-supported, the layout of models is rarely considered in the traditional model evolution process. Most evolution efforts focus only on the semantic aspects of the evolution (e.g., adding or removing necessary model elements and connections, modifying attributes of model elements), and often ignore model layout configuration concerns during the evolution (e.g., positions of model elements, font, color and size used in labels). For instance, executing a set of model transformation rules to add model elements and connections will sometimes lead to placing all the newly created elements in a random location in the model editor.

Ignoring the desired layout after model evolution has a strong potential to undermine the readability and understandability of the evolved model, and may even unexpectedly affect the implicit semantics under certain circumstances. For example, users may accidentally misunderstand the system because of a disordered layout (e.g., a sequence of actions to be executed is represented by a set of nodes with arrows indicating the sequence, but a disordered arrangement of the nodes may lead to a challenge in identifying the correct execution order). Furthermore, the positions of model elements and connections may correspond to special coordinates in the real world, such that an unoptimized layout could lead to unexpected problems for the actual system (e.g., the configuration of the actual hardware devices and cables might be based on the positions of model elements and connections representing them, or the color of the elements might represent the running status of the actual devices). It may be possible to incorporate the layout information related with the implicit semantics into the metamodel as part of the abstract syntax, but a change to the metamodel may trigger further model migration problems (Sprinkle, 2003). Although it is very direct to manually adjust the layout, it becomes a tedious, timing-consuming task when a larger number of model elements are involved in the model evolution process. Therefore, while the semantic concerns of model evolution have been implemented and automated, it is indispensible to realize the automatic configuration of the layout as part of the model evolution process.

The most commonly used approach to automatically arrange the layout of models is to apply