Computers in Human Behavior 51 (2015) 734–741

Contents lists available at ScienceDirect

Computers in Human Behavior

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

A collaborative *testbed* web tool for learning model transformation in software engineering education

D. Rodríguez-Gracia, J. Criado, L. Iribarne *, N. Padilla

Applied Computing Group, University of Almería, Spain

ARTICLE INFO

Article history:
Available online 20 December 2014

Keywords:
MDE
Model transformation
M2M
ATL
EMF
Learning tool

ABSTRACT

Software Engineering provides mechanisms to design, develop, manage and maintain social and collaborative software systems. At present, the Software Engineering *Curricula* includes teaching Model-Driven Engineering (MDE) as a new paradigm that enables higher productivity, attempting to maximize compatibility between systems. Modern learning methods MDE require the use of practical approaches to analyze new model-transformation techniques. Model transformations are carried out by using very high-level languages, like the ATL language. This model transformation language is built as a *plugin* for the Eclipse framework, and users who want to collaborate and develop software with it, have some difficulties executing ATL transformations outside this platform. To handle models at runtime, it is interesting to perform the transformations in a standalone way. In this context, we have developed a testbed web tool which aims to be useful for learning model transformation techniques. The tool offers a Graphical User Interface to test and verify the involved model transformations. The proposal is useful as a collaborative scenario for learning MDE and model transformation issues and techniques in Software Engineering education.

1. Introduction

Nowadays, the *Software Engineering* (SE) educators have to deal with the difficulty of teaching students not only the theoretical concepts (Offutt, 2013) but also the engineering processes for actual projects. It is often not easy to find representative examples that illustrate the software engineering process and the difference of abstraction between software engineering programs and computer science programs (Parnas, 1999). Nevertheless, the best way to get in touch with these techniques is testing them by means of practical examples and using the right tools when students are learning SE.

At present, the *Curriculum* in software engineering includes teaching *Model-Driven Engineering* (MDE) as a new paradigm in the software development that enables higher productivity attempting to maximize compatibility between systems. The previous statements can also be applied in the case of MDE, because this methodology requires the use of practical approaches that allow both the educators and software engineering students to analyze model transformation techniques.

Within MDE, model transformations are the main mechanism for the development of software systems, because these operations allow us to automate the management of the models which have been defined to describe them. In *Model-Driven Architecture* (MDA), model transformations have traditionally been used at design time to build software from the *Computation Independent Model* (CIM) level, going through *Platform Independent Model* (PIM) and *Platform Specific Model* (PSM) levels, to the code level. Furthermore, model transformations have also been used to refine models of a particular level based on certain modifications of the system along its life cycle. However, at present, some systems require to adapt themselves at runtime due to the changes in the system context or due to new requirements that were not detected in the design phase (Blair, Bencomo, & France, 2009).

The most powerful method for implementing model transformations is the use of transformation languages. ATLAS Transformation Language (ATL) (Jouault, Allilaire, Bézivin, & Kurtev, 2008) is one of the most widely used model transformation languages. It is usually executed using the specific plugin within the Eclipse platform. This fact implies that learning, design, implementation and execution of ATL model transformations depend on the platform, which is not always desirable. It may be interesting to be able to run the transformations outside such framework, allowing more

* Corresponding author.
E-mail address: luis.iribarne@ual.es (L. Iribarne).

http://dx.doi.org/10.1016/j.chb.2014.11.096
0747-5632/

open access to model transformation techniques and encouraging the use of such transformations to adapt systems at runtime.

In this context, the teaching–learning process can be improved if it is carried out collaboratively between the different actors involved in the process. In this regard, *Computer-Supported Collaborative Learning* (CSCL) may provide the strategies required to successfully achieve such process. CSCL is based on the development of software applications in which the collaboration has a special emphasis. In this paper, we describe a tool available on the web that aims to bring software engineering and model transformation techniques to SE education. This tool is part of a series of applications that together conform a CSCL environment. In this socio-technical environment, two types of products appear: those products used for learning a specific feature of the domain of the SE (such as the tool described in this paper), and those ones used to support collaboration tasks (*e.g.*, a collaborative editor, a subversion repository, etc.).

This paper focuses solely on describing the product which has been developed for the collaborative learning of model transformation techniques. For this purpose, the tool makes use of ATL and Eclipse Modeling Framework (EMF) (Steinberg, Budinsky, Merks, & Paternostro, 2008) libraries to provide model transformation and model validation services. These capabilities have been tested by implementing a sequence of transformations at runtime. This transformation sequence results in an adaptation process which is in charge of dynamically generating a non-preset Model-to-Model (M2M) transformation from a repository of rules, which is responsible for adapting component-based software systems. Thus, the tool allows us to study how model transformations work based on the execution of this adaptation process, which operates in a standalone way without depending on the Eclipse platform.

As mentioned above, the main use case of the tool is to provide an execution environment for testing the adaptation of component-based systems. Therefore, any software system which is built from components can be a use case of the tool and, consequently, of the underlying adaptation process. Therefore, some application examples are the smart home software systems, smart TV applications, component-based robotic systems, widget-based user interfaces, etc. All these example scenarios offer a component-based architecture that may have the necessity of being adapted at runtime and hence the proposed tool can be used to learn how model transformations can be applied within this context.

The rest of the article is organized as follows. Section 2 reviews the context of the tool and the implemented adaptation process. Then, Section 3 describes the tool design and implementation details. Later, Section 4 gives some transformation examples for the better understanding of how the tool works and discusses the results. Section 5 shows an overview of related works and, finally, Section 6 presents the conclusions of the work.

2. Adapting component-based software systems

In order to understand the developed tool, it is necessary to describe the scenario from which the model transformation sequence that is executed emerged. The aim of our sequence of transformations is to adapt component-based software systems at runtime. In our research work, component-based software systems are represented in four levels, from the task specification to the running software architectures as it is explained in (Criado, Iribarne, Padilla, Troya, & Vallecillo, 2012) (see Fig. 1). The highest level of abstraction that describes our architectures is the task and concepts level which matches the CIM level in MDE. The next one is the abstract architectural model level which corresponds to the PIM level in MDE. It represents the software architecture in terms of what kind of components it must contain, what the relationships between them are like, and what specifications these components have. Then, the concrete architectural model level corresponds to the PSM level in MDE and it describes what concrete components, which have been selected from a repository, best fulfill the abstract definition of the software architecture. Finally, the code level in MDE is represented by the final software architectures, which are made up of the source code that generates the running software system.

The tool focuses on the execution of the mentioned transformation sequence, which is performed at the abstract level of the architecture definitions. Its goal is to adapt an architectural model using an M2M transformation not defined a priori which is built at runtime by selecting some transformation rules defined in a repository (Rodríguez-Gracia, Criado, Iribarne, Padilla, & Vicente-Chicote, 2012). The transformation that adapts the architectural models is horizontal and it occurs in the PIM level. In addition, this kind of transformation is endogenous, because the source and the target models are defined according to the same metamodel (Mens & Van Gorp, 2006).

Our adaptation process comprises a sequence of M2M transformations which, taking as inputs (a) an initial architectural model, (b) a model with the context information and (c) a repository model containing the transformation rules, generates (d) the adapted architectural model as output (Fig. 2). Although the purpose of this paper is not to describe the adaptation process in depth, it is necessary to briefly introduce the involved transformations:

(a) **ContextProcessing** is an M2M transformation in charge of processing the context information and resolving the adaptation operations that must be executed.
(b) **RRR** is an M2M transformation which is responsible for rating the transformation rules of the repository.
(c) **RuleSelection** is an M2M transformation process which selects the highest rated rules.
(d) **RSL** is an M2M transformation that updates the attributes of the rule repository based on the selected rules.
(e) **RuleTransformation** is a Higher-Order Transformation (HOT) (Tisi, Jouault, Fraternali, Ceri, & Bézivin, 2009) which is in charge of translating the selected adaptation rules into ATL rule model.
(f) **ATLExtraction** is a Textual Concrete Syntax (TCS) (Jouault, Bézivin, & Kurtev, 2006) extraction process responsible for generating the ATL code from the ATL rule model.
(g) **ArchitecturalModelTransformation** is the M2M transformation created dynamically as result of the transformation sequence and it is in charge of adapting the initial architectural model by applying the selected transformation rules.

These transformations within the adaptation sequence are invoked in the correct order from the web tool and the generated results are shown to the user by means of a graphical user interface. These results include: the adapted architectural model, the updated values of the repository of rules and the log information related to the model transformations executed.

3. A web tool for testing model transformations

The sequence of transformations described above provides an appropriate scenario to learn the behavior of model transformations. However, it is essential to have a tool to carry out this adaptation process, not only running the transformations involved, but also providing the user with a test scenario which allows him/her to vary the input conditions and see the results that are produced as output.