A Collaborative *testbed* Web Tool for Learning Model Transformation in Software Engineering Education

D. Rodriguez-Gracia, J. Criado, L. Iribarne[1], N. Padilla

Applied Computing Group, University of Almeria, Spain

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.

Keywords: MDE, Model Transformation, M2M, ATL, EMF, Learning tool.

[1]Email address: luis.iribarne@ual.es

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, 2009).

The most powerful method for implementing model transformations is the use of transformation languages. ATLAS Transformation Language (ATL)

(Jouault, 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 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, 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 , 2012) (see Figure 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.

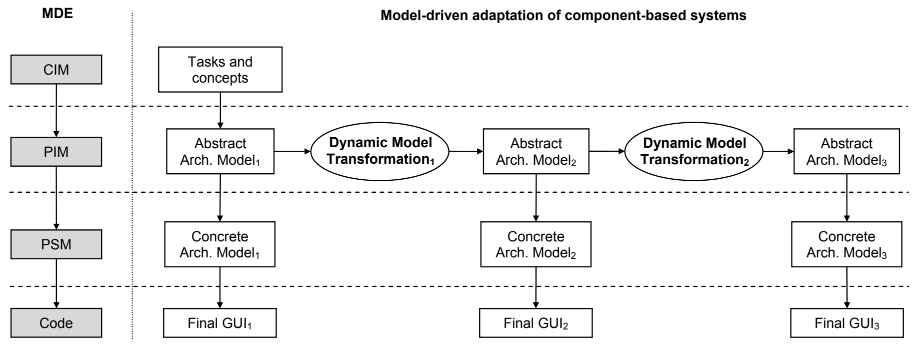


Figure 1: Model Transformation of Component-based systems

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 (Rodriguez, 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, 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 (Figure 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, 2009) which is in charge of translating the selected adaptation rules into ATL rule model.

(f) **ATLExtraction** is a Textual Concrete Syntax (TCS) (Jouault, 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.

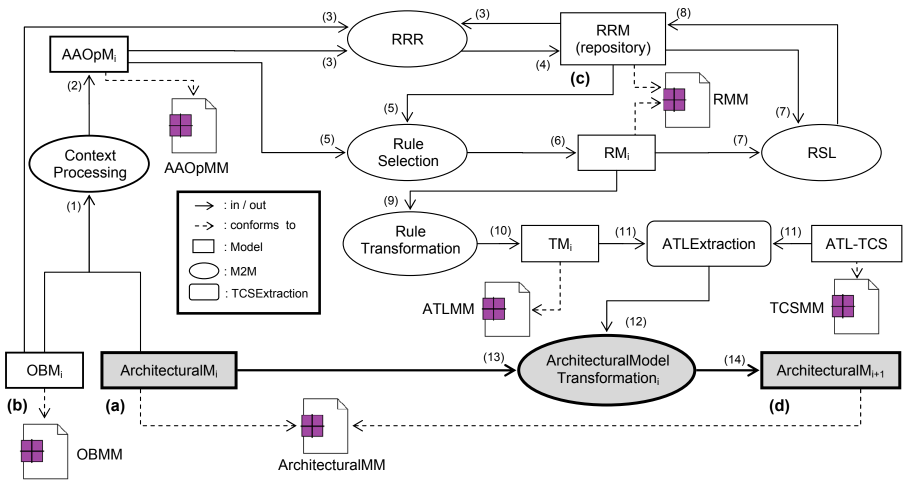


Figure 2: The adaptation schema executed by the tool

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

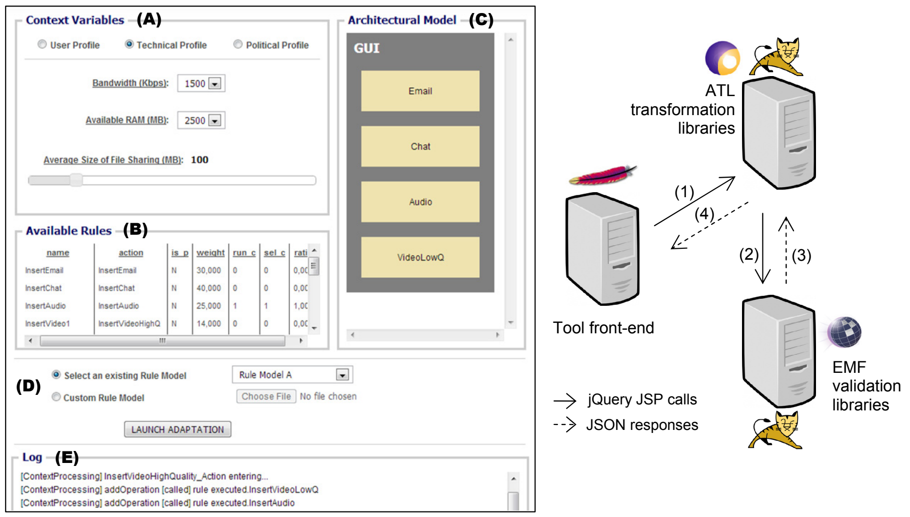
The sequence of transformations described above provides an appropriate scenario to learn the behavior of model transformations. However, it is essential to have a tool to carry out this adaptation process, not only running the transformations involved, but also providing the user with a test scenario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

The developed tool is strongly linked to the one of the adaptation domain in which our research work is based. In such domain, architectural models represent graphical user interfaces as part of research projects of the Spanish Ministry and the Andalusian Government that require adaptation of Graphical User Interfaces (GUI) at runtime. In this context, it is useful to have component-based user interfaces that adapt their functionality depending on the circumstances. We intend to develop smart graphical user interfaces (SmartGUIs) which learn from the user interaction, modifying and adapting their behavior.

Specifically, the tool describes an adaptation scenario in which a cooperative task is performed (Iribarne, 2012). In this task, three users with different roles participate and, at a certain point, they need to communicate. Therefore, the user interfaces should be adapted, incorporating the communication components (textual chat, audio, video, etc.) that allow them to interact. In addition, the communication components that are incorporated into the GUIs depend on the user's profile and certain context variables.

In order to make the tool accessible from any platform, a graphical user interface (Figure 3) has been developed in the web environment. Using this GUI, students can test the input of different values of context variables (A). They can select between different user profiles, considering that each one is associated with a set of available components, which affects the transformation process. Moreover, they can vary not only the values of the bandwidth and the memory available in the system, but also the value of the average size of the files that are being exchanged. The values affect the output of the transformation sequence, since the generated architectural model must be adapted to the interaction and the available resources.

Furthermore, the web tool shows some information about the repository rules that are being used by the adaptation process (B). In this part of the GUI, the attribute values of the rules involved in the model-to-model transformation processes are displayed. In addition, when the transformation sequence is performed, students can see the updated values of the affected rules. On the other hand, the right side of the GUI shows the current architectural model (C). In this part of the user interface, students can see the architectural model that is obtained at the end of the transformation sequence. This architectural model describes the component-based graphical user interface as mentioned above, so it is not the final view of the GUI, but just a representation of it.



User Profile [✓ Technical Profile] Political Profile

Bandwidth (Kbps): 1500

Available RAM (MB): 2500

Average Size of File Sharing (MB): 100

name	action	is_p	weight	run_c	sel_c	rati
InsertEmail	InsertEmail	N	30,000	0	0	0,0C
InsertChat	InsertChat	N	40,000	0	0	0,0C
InsertAudio	InsertAudio	N	25,000	1	1	1,0C
InsertVideo1	InsertVideoHighQ	N	14,000	0	0	0,0C

[✓ Select an existing Rule Model] Rule Model A

Custom Rule Model Choose File No file chosen

```
[ContextProcessing] InsertVideoHighQuality_Action entering...
[ContextProcessing] addOperation [called] rule executed.InsertVideoLowQ
[ContextProcessing] addOperation [called] rule executed.InsertAudio
```

Figure 3: Graphical user interface of the web tool

In order to provide more flexibility in the test scenario, the tool gives us the possibility of selecting which rule repository is going to be used (D). We can either select a predefined rule repository model from a series of predefined models, or provide our own repository model. Once the modifications have been made in the context variables, and the rule repository is selected, we can start the execution of the transformation sequence by pressing "Launch Adaptation" button. When the transformation processes have been executed, the tool displays the adapted architectural model, the updated rule repository, and some log information about the rules that have been performed in the M2M transformations. This piece of information, shown at the bottom of the user interface (E), allows us to check if the process is working properly. Additionally, this part of the tool helps students to better understand the implemented model transformations.

In summary, the tool consists of five areas, two for the modification of the input data and three for the visualization and analysis of information generated as output. In parts (A) and (D), we can change both values of the context variables as input rule repository, respectively. Furthermore, in parts (B), (C) and (E), we can see the updated values of the repository of rules, the adapted architectural model, and the log information related to the model transformations executed.

The tool has been implemented following a three-tier server architecture so that it can be executed from any web platform without installing any local application or Eclipse plugin. The graphical user interface described above performs the functions of the front-end tool and is deployed in an Apache Web Server. Another server offering the M2M transformation services has been developed and deployed in a Tomcat Web Server. Finally, a third server that performs the validation processes is also deployed in a Tomcat Web Server (see Figure 4).

This server architecture allows us to separate the tool functionalities and make them independent. With this aim, ATL and TCS libraries have been deployed in a separated server to provide functions responsible for executing each M2M transformation and TCS extraction of the adaptation process. Additionally, EMF libraries have been deployed in a different server that provides functions which are called from the server in charge of the model transformations.

The steps to operating the tool are summarized as follows. In the first place, the tool is accessed by means of the front-end represented by the described GUI. Then, when the adaptation process is launched, the transformation services are called asynchronously from the front-end server in order to execute the sequence of model transformations (step 1 in Figure 3). These services are called through *Java Server Pages* (JSP) requests. Within the transformation server, for each new model that is generated, the validation server is called to check that this model is built according to its metamodel (step 2).

The validation services are also called through JSP requests and their function is to check that *Object Constraint Language* (OCL) (Cabot, 2012) constraints and other structural definitions specified in the metamodels are fulfilled. Once the validation has been run, the server sends a response in *JavaScript Object Notation* (JSON) format (step 3), and the sequence of transformations continues if the model has been successfully validated; otherwise, it will provide an error message. When all M2M transformations have been performed and once checked that the models have been generated correctly, the result is returned to show the user the adapted architectural model, the updated rule repository and the log information about the transformation sequence (step 4).

4. Case study

In order to provide students with a test scenario, this section shows a case study in which the tool is executed with specific input values. Let us remember that the sequence of model transformations implemented by the tool aims to adapt component-based GUIs, so it is necessary to briefly describe the assumptions of this scenario.

In our domain, a user interacts with a GUI which is made of the following components: an email component, a chat, an audio component, a low-quality video, a high-quality video, a file sharing component (in the same way as Dropbox) and a digital blackboard. Moreover, the user profile has such an impact on the components that it neither has the file sharing component nor the blackboard available. Similarly, the "technical" profile does not have the high quality video component available either. In addition to the user profile, the context variables related to the available bandwidth and memory, and the average size of shared files also affect the output of the transformation process. Therefore, the sequence of model transformations will modify the input model according to the context variables and the current state of the architecture.

Let us suppose a user who starts with a graphical user interface with three single, simple components: an Email component, a Chat component and an Audio component. The user profile is a "technical" profile, the available bandwidth is 750 kbps, the available memory is 1,500 MB and the average size of the shared files is 50 MB. Then the following changes in the values of the context variables occur: the new available bandwidth is 1,500 kbps, the available memory is 3,000 MB and the user is sharing files with an average size of 200 MB. Consequently, the new adapted architectural model will incorporate the low quality video and file sharing components, because the available resources have increased and the file sharing component will be beneficial for the user, since large files are being exchanged. We can see the transformation example in Figure 4.

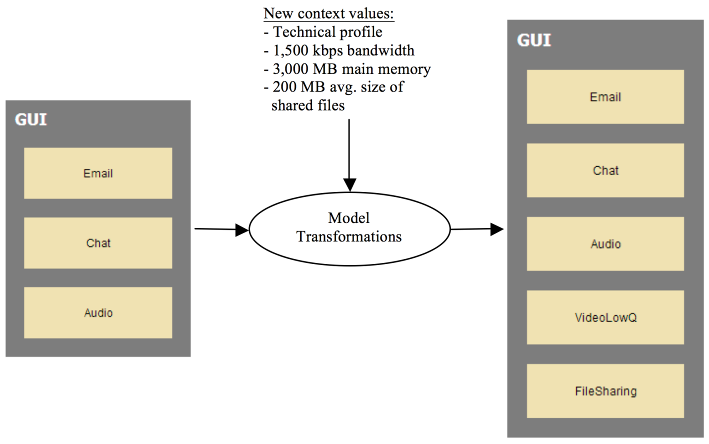


Figure 4: Model transformation, Example #1

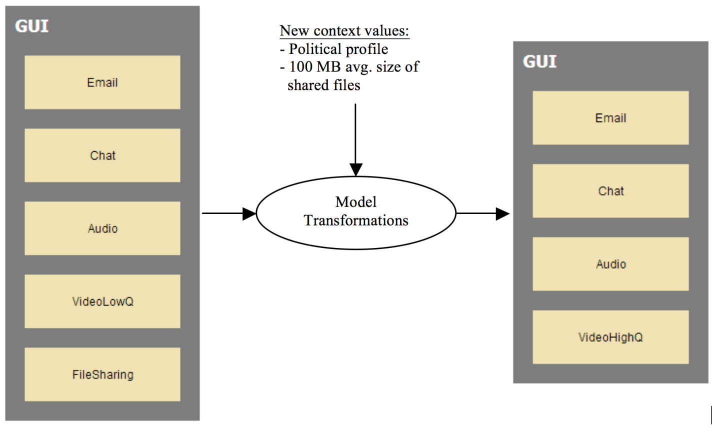


Figure 5: Model transformation, Example #2

In the next transformation example, let us suppose that the user who interacts with the graphical user interface changes from a "technical" profile to a "political" profile and, currently, the user is sharing files with an average size of 100 MB rather than 200 MB. Then, on applying the sequence of model transformations implemented by the tool, the resulting architectural model will remove the file sharing component and the low quality video component and will insert the high quality video component (see Figure 5). This is because, for the technical profile, the high quality video component is not available, and for the new value of the average size of the shared files, the file sharing component is not needed (as it is possible to continue sharing smaller files using the chat component).

The adaptation actions that can be performed depends on the available rules of the repository which is managed by the tool. For this reason, we implemented in the tool an option to be able to select between different repositories. It is also possible to upload a custom repository with ATL transformation rules. Thus, the tool can be used to test the behavior of our own transformation rules. Both previous examples show the corresponding transformations obtained by using an example repository, which is composed by the rules shown in Table 1. According to the example domain, there is a rule for inserting each component

Table 1: Available rule repository

Rule ID	**Adaptation action**
#1	add email
#2	add chat
#3	add audio
#4	add videoLQ
#5	add videoHQ
#6	add fileSharing
#7	remove email
#8	remove chat
#9	remove audio
#10	remove videoLQ
#11	remove videoHQ
#12	remove fileSharing

as well as a rule for its deletion.

With an illustrative purpose, we added to this repository the two rules shown in Figure 6. The first rule (`rule13`) is in charge of inserting two components: a simple component representing a low quality video element, and a complex component (containing two simple components) for the management of the video recording. Therefore, this rule is like the rule #4, but with an additional behavior. The second rule (`rule14`) is intended to delete the email component if there is a chat component in the architecture.

In this case, if we use the previous repository (Table 1) with the addition of the rules of Figure 6, we obtain another output model as a result, which is different from the one shown in Figure 4. Taking the same values for the input context variables and the same input model, the obtained model incorporates a low-quality video component, a recording manager and a file sharing component; otherwise, the resulting model removes the email component, as we can see in Figure 7. We assume that the rule #13 is selected instead the rule #4 and that the rule #14 is selected in addition to the rule #6. The context processing and the selection of the rules are two modules of the adaptation schema (Figure 2), and are not explained because the internal behavior of the adaptation is out of the scope of this paper.

```
rule rule13() {
  to
    t1 : AMM!SimpleAbstractComponent (
      component_name <- 'VideoLowQ',
      component_parent <- thisModule.getComponent('GUI') ),
    t2 : AMM!ComplexAbstractComponent (
      component_name <- 'RecManager' ),
    t3 : AMM!SimpleAbstractComponent (
      component_name <- 'LayoutSelection'
      component_parent <- t2 ),
    t4 : AMM!SimpleAbstractComponent (
      component_name <- 'OutputConfig'
      component_parent <- t2 )
}

rule rule14() {
  from
    f : AMM!SimpleAbstractComponent (
          f.component_name = 'Email' and
          thisModule.existComponent('Chat') )
    to drop
}
```

Figure 6: New transformation rules added to the repository

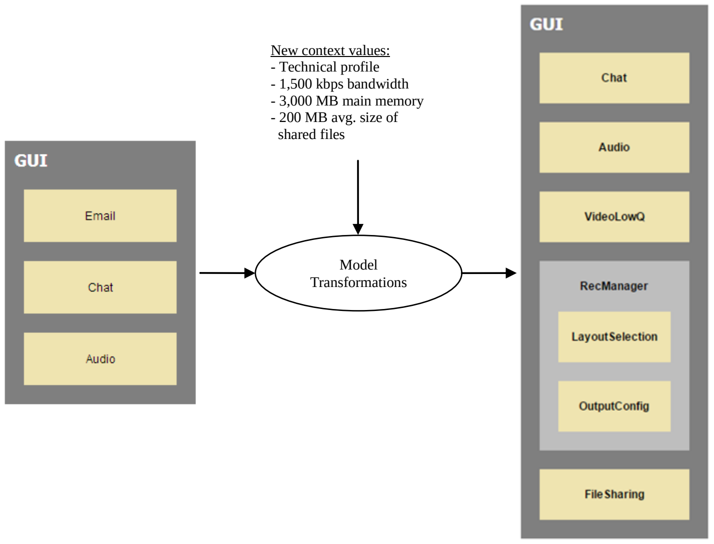


Figure 7: Model transformation, Example #3 (with new transformation rules)

Thus, we demonstrated how the results of model transformations are affected by the inputs to these processes. In our example scenario of adaptation, the inputs are the context variables and the component-based (input) models. We also illustrated the importance of the rules which form part of the model transformation process by introducing two new rules to the repository of available rules. In this sense, we shown how our tool offers the possibility of modifying the rule repository and use it in the transformation process. Users can try the tool and deal with e-learning model transformation techniques by visiting the following link: http://acg.ual.es/isoleres/adaptation.

5. Related work

There is a wide range of transformation languages such as ATL (Jouault, 2008), ETL (Kolovos, 2008), RubyTL (Sanchez, 2006) or TGG (Shurr, 1995). Nevertheless, not all languages provide powerful tools for the specification of the transformations or may be useful for teaching and for its application in EIS education. Regarding model transformation tools, most of them are implemented as plugin within the Eclipse environment or require the use of specific software for their handling, execution and for testing purposes to assist the user in learning.

Wires tool (Rivera, 2009) provides a graphical and executable language to implement ATL transformations. The tool also offers mechanisms to enable modular composition of complex model transformation chains. With this tool, you can define a sequence of transformations by connecting the inputs and outputs of each transformation with the required item. In contrast, the sequence of transformations is already pre-established in our tool, and our goal is to show their behavior, so that the user can change the values of the input models and analyze their results.

The work in (Guerra, 2010) presents an Eclipse tool to define model transformation specifications by using a visual concrete syntax. This tool is developed with *Graphical Modeling Framework* (GMF) (Gronback , 2008) and generates

ETL transformations. This tool is useful for building visual transformation languages, as it makes their understanding and teaching easier. However, it does not provide extra support for the execution of the transformation or for the visualization of the models on which it operates.

Regarding the model validation, the work presented in (Bezivin, 2006) describes how the ATL model transformation tool itself may be used to validate the models generated in a transformation process. In our case, the model validation is performed by using the EMF libraries, so that the models are validated according to their metamodel, due to the structural constraints and the OCL constraints defined within the metamodel through *OCLInEcore*.

Furthermore, the USE tool (Gogolla, 2007) allows models with OCL constraints to be validated contrary to the developer's assumption. This tool shows not only a graphical user interface to navigate through the models and the constraints but also some log information about the executed model checking. Another work making a comparison of tools for OCL can be found in (Toval, 2003).

It is possible to find some related work that provides a test scenario. In (Moring, 2009), a tool implemented within DiVA project to test a dynamic customer relationship management (DCRM) system is shown. In this tool, the system analyzes the context and explicitly constructs a suitable configuration using *Aspect-Oriented Modeling* (AOM) techniques at runtime. The tool validates this configuration by using traditional MDE techniques, such as invariant checking or simulation. Moreover, the system automatically generates a safe reconfiguration script to actually adapt the running business system. The difference is that our tool uses model transformations to perform reconfigurations, rather than *Dynamic Software Product Lines* (DSPL).

6. Conclusions

This paper presents a collaborative web tool that can be used for the experimentation by students of Software Engineering (SE) courses. The tool has been developed for the implementation of a sequence of model transformations and allows us to perform the involved model transformations and model validations as part of an adaptation process for component-based graphical user interfaces. The transformation and validation services are deployed on a three-tier server architecture and are called asynchronously by the web tool.

Among other features, such services can be reused and invoked by different web applications that require the execution of ATL model transformations and EMF model validations. For a better understanding of the tool, we presented a case study with three execution examples, which shows how an initial model is adapted to variations in the context variables introduced by means of the tool.

Using this developed tool, we achieved two objectives. On the one hand, it is a validation tool of our proposed adaptive model transformation at runtime and, on the other hand, it is a practical approach to MDE. Thus, the final goal of the proposal presented in this paper is the tool can be used as an educational-learning object, in which the users may experience (in a practical way) the model transformation concepts, and perform a sequence of operations at runtime, allowing the users to analyze the obtained results.

As mentioned above, our research work focuses on the development of a CSCL (Computer-Supported Collaborative Learning) environment for SE learning. This environment is made up of a set of SE domain-specific and general-purpose tools aimed to support the teaching-learning collaborative processes. Therefore, a key issue is the integration of the tool described in this article with the existing ones in the CSCL environment.

Finally, we want to develop some satisfaction and opinion surveys that will be carried out on students using the web tool. In addition, we intend to incorporate the possibility of dynamically defining the context variables and their range in order to make the tool more open and less restricted to the scenario.

Acknowledgments

This work was funded by the EU ERDF and the Spanish Ministry of Economy and Competitiveness (MINECO) under Project TIN2013-41576-R, and the Spanish Ministry of Education, Culture and Sport (MECD) under a FPU grant (AP2010-3259), and the Andalusian Regional Government (Spain) under Project P10-TIC-6114. This work was also supported by the CEiA3 and CEIMAR consortiums.

References

J. Offutt, "Putting the Engineering into Software Engineering Education", IEEE Software, 30(1), 2013, pp. 93–95.

D. Parnas, "Software Engineering Programs Are Not Computer Science Programs", IEEE Software, 16(6), 1999, pp. 19–30.

G. Blair, N. Bencomo and R.B. France, Models@Run.time, Computer, 40(10), 2009, pp. 22–27.

F. Jouault, F. Allilaire, J. Bzivin and I. Kurtev, "ATL: A model transformation tool", Science of Computer Programming, 72(1-2), 2008, pp. 31–39.

D. Steinberg, F. Budinsky, E. Merks and M. Paternostro, "EMF: Eclipse Modeling Framework", Addison-Wesley Professional, 2008.

J. Criado, L. Iribarne, N. Padilla, J. Troya and A. Vallecillo, "An MDE approach for Runtime Monitoring and Adapting Component-based Systems: Application to WIMP User Interface Architectures", in 38th Euromicro Conference on Software Engineering and Advanced Applications, 2012, pp. 150–157.

D. Rodrguez-Gracia, J. Criado, L. Iribarne, N. Padilla and C. Vicente-Chicote, "Runtime Adaptation of Architectural Models: An Approach for Adapting User Interfaces", in 2nd International Conference on Model and Data Engineering, 2012, pp. 16–30.

T. Mens and P. Van Gorp, "A Taxonomy of Model Transformation", Electronic Notes in Theoretical Computer Science, 152, 2006, pp. 125–142.

M. Tisi, F. Jouault, P. Fraternali, S. Ceri and J. Bzivin, "On the Use of Higher-Order Model Transformations", in 5th European Conference on Model-Driven Architecture Foundations and Applications, 2009, pp. 18–33.

F. Jouault, J. Bzivin and I. Kurtev, "TCS: a DSL for the Specification of Textual Concrete Syntaxes in Model Engineering", in 5th International Conference on Generative Programming and Component Engineering, 2006, pp. 259–254.

L. Iribarne, N. Padilla, J. Criado, C. Vicente-Chicote, "Metamodeling the Structure and Interaction Behavior of Cooperative Component-based User Interfaces", Journal of Universal Computer Science, 18(19), 2012, pp. 2669–2685.

J. Cabot and M. Gogolla, "Object Constraint Language (OCL): A Definitive Guide", in M. Bernardo, V. Cortellessa and A. Pierantonio (eds), Formal Methods for Model-Driven Engineering, LNCS, vol. 7320, Springer, Heidelberg, 2012, pp. 58–90.

D.S. Kolovos, R.F. Paige, F. Polack, "The Epsilon Transformation Language", Theory and Practice of Model Transformations, 2008, pp. 46–60.

J. Snchez-Cuadrado, J. Garca-Molina, M. Mernanguez-Tortosa, "RubyTL: A Practical, Extensible Transformation Language", Model Driven Architecture-Foundations and Applications, 2006, pp. 158–172.

A. Schrr, "Specification of graph translators with triple graph grammars", Graph-Theoretic Concepts in Computer Science, 1995, pp. 151–163.

J.E. Rivera, D. Ruiz-Gonzlez, F. Lpez-Romero, J. Bautista and A. Vallecillo, "Orchestrating ATL Model Transformations", in MtATL, 2009, pp. 34–46.

E. Guerra, J. de Lara, D. Kolovos and R. Paige , "A Visual Specification Language for Mode-to-Model Transformations", in 2010 IEEE Symposium on Visual Languages and Human-Centric Computing, 2010, pp. 119–126.

R.C. Gronback, “Eclipse Modeling Project: A Domain-Specific Language (DSL) Toolkit”, Addison-Wesley Professional, 2008.

J. Bzivin and F. Jouault, “Using ATL for Checking Models”, Electronic Notes in Theoretical Computer Science, 152, 2006, pp. 69–81.

M. Gogolla, F. Bttner and M. Rochters, “USE: A UML-based specification environment for validating UML and OCL”, Science of Computer Programming, 69(1–3), 2007, pp. 27–34.

A. Toval, V. Requena and J. Fernndez, “Emerging OCL tools”, Software and Systems Modeling, 2(4), 2003, pp. 248–261.

B. Moring, O. Barais, J.M. Jzquel, F. Fleurey and A. Solberg, “Models@run.time to support dynamic adaptation”, Computer, 42(10), 2009, pp. 44–51.