Combining Textual and Web-based Modeling

Martin Haeusler, Matthias Farwick, and Thomas Trojer

University of Innsbruck, Technikerstr. 21a, 6020 Innsbruck, Austria
firstname.lastname@uibk.ac.at,

Abstract. Documenting large scale IT-architectures is a laborious task that is executed by many different stakeholder types. We argue that a major obstacle that keeps stakeholders from keeping models up-to-date is inadequate user interfaces for specific stakeholders. In this demo paper we present a novel modeling tool that provides adequate stakeholder views and describe the implementational challenges. The tool motivates users to contribute documentation by allowing textual modeling for technical users and web-based modeling via forms for business users at the same time. The tool demonstration video is available at: https://www.youtube.com/watch?v=PaP2Sppiv7g&feature=youtu.be

1 Introduction

In the context of Enterprise Architecture Management (EAM) and systems operation management, specialized tools are often used to model the dependencies between the IT-infrastructure, deployed applications and the business functions they support [5]. These models are used to analyze the current architecture, assess risks and plan changes to the architecture.

In our previous empirical research [3] we showed that a major problem in EA documentation is that the EA models become quickly outdated. Therefore we focused on increasing automation in EA documentation and enhancing semi-automated data collection processes [3]. However, we realized that automation is not always applicable, especially in the case of application modeling where human abstraction is needed to hide distracting detail.

Therefore, we argue that in order to motivate stakeholders to continuously contribute their knowledge to the model, the data input methods need to be adapted to the preferences of the different user types. In our previous work [4] we presented *Txture*, an EA documentation tool that uses a textual domain-specific language (DSL) as a collaborative model input method. In *Txture* visualizations of the model can then be created online via a corresponding web-application. Experience that we gained with the tool in practice showed that this textual input method is fast and intuitive for technically skilled users and increases their motivation to contribute to the documentation, in particular when they are already working in a textual environment. However, for less technically skilled users, no alternative input method was available.

In this demo paper we therefore present a proof–of–concept prototype as part of the *Txture* development process that combines textual and form-based

modeling in the web. It thereby allows distinct user groups to work on the same model in their preferred input style. Both the textual and the web-based clients are kept in-sync.

There exists some related work in this area. For example the projectional modeling approach of the Meta Programming System[1] or the recent work of Atkinson et al. [1] that combines graphical and textual modeling. The two major differences to our approach are that the presented systems (a) work on desktop clients and (b) are applying a projectional editing approach in which the text can only be edited with a specific tool and not any text editor. Other approaches, like the work of Engelen et al. [2] only allow one-way synchronization between the clients. For example, the authors propose a textual language for describing *UML Activity* instances which are embedded in an *XMI* file. This file is processed by a compiler which transforms the textual descriptions into real Activities. The opposite direction – from object representation to textual syntax – is left as an open issue. In our demonstration video that accompanies this paper, we explain our use–case in more detail and demonstrate both transformation directions. Since the combination of the two modeling paradigms is a particularly complex task, we detail some of the key technical challenges in this demo paper that we faced when implementing this novel approach to model management and EA documentation.

2 Combining Textual and Web-based Modeling

The prototype consists of three main components. First, an Eclipse-based plugin that allows textual modeling according to a pre-defined syntax and is based on the *Xtext* framework[2]. Second, a web-application that allows to enter model data in a form-based manner. Third, a model repository that centrally stores the model for all clients. The difficulty of integrating the two types of modeling lies in the two different model organization styles. Model element organization on the textual side is governed by folders and files that contain structured text which is parsed to build the model in memory. On the other side a multi-user web application is employed that stores the model centrally in a model repository. As shown in Figure 1 we tackle this problem with our tools and both, the changes to the text model and the ones applied to the web-client synchronized via the model repository.

2.1 Challenges in Synchronizing Textual and Web-based Models

This section gives an overview of the most significant problems and our solutions. Figure 1 shows the simplified communication between attached clients in a distributed modeling context. Every number in the figure corresponds to a paragraph number which explains the synchronization challenges that occur.

[1] JetBrains MPS Homepage: `http://jetbrains.com/mps`
[2] Xtext Homepage: `http://www.xtext.org`

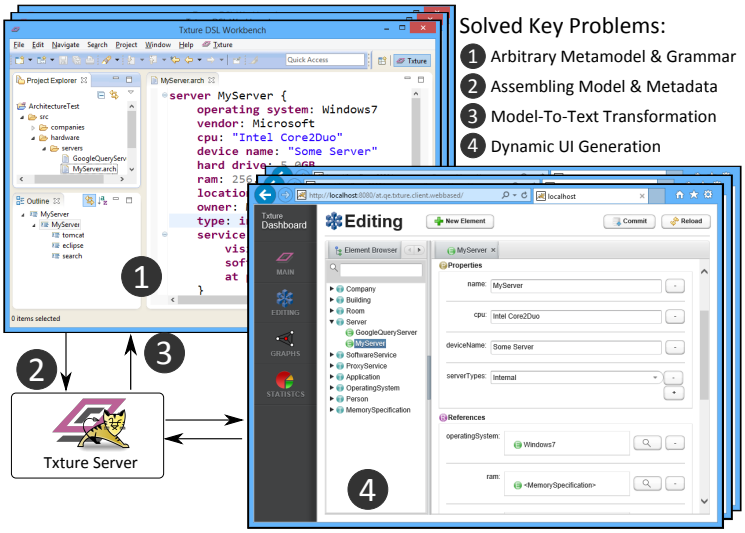

Fig. 1. Synchronizing Model Data across multiple Clients

Problem 1: Arbitrarity of Grammars & Metamodels
The textual client uses *Xtext*, a parser-based framework for DSLs. In order to suit the requirements of a given use case, our tool needs to be capable to deal with any Xtext grammar and metamodel.
Solution: We employ a reflective approach to metamodel access by utilizing the capabilities of *EMF*[3], avoiding a direct compile-time dependency to any given metamodel. In order to deal with arbitrary grammars, we offer interfaces for *Language Extensions*; i.e. *Eclipse Plugins* that contain Xtext grammars and Model–to–Text generators.

Problem 2: Assembling the Model & Gathering Metadata
Since the textual client relies on files containing concrete DSL syntax for storing the model, they must be parsed and merged into a single model. Still, we later require the source file location of any merged model element. This is important to later re-assemble model data into the correct positions in the text files and folders.
Solution: Xtext itself is based on EMF *Ecore* for representing the *Abstract Syntax Trees* (AST) that result from each parser run. We assemble the AST of each DSL file into a common model, resolving all encountered cross-references between model elements. The result is the model that we need to commit to the server. Xtext also offers access to element-based metadata (e.g. the element file location) which we need to send to the server as well. Please refer to Section 3 for details.

[3] Eclipse Modeling Framework: `http://www.eclipse.org/modeling/emf/`

Problem 3: Model–to–Text Transformation & File Locations
During a checkout process, the textual client receives a serialized version of the model. Each element in the model must be converted to concrete DSL syntax and furthermore must be placed in the correct file at the correct position.
Solution: The Model–to–Text Transformation depends directly on the grammar of the model at hand and therefore cannot be processed directly by the generic part of the application. Consequently, we defer this task to a specialized Eclipse Plugin which contains the Model–to–Text generator for the particular grammar. After generating the concrete syntax for a model element, the tool decides upon the file location of each resulting piece of text by taking the element file location meta-information into account. If none is given (e.g. if the element was newly created by the web client), it is put into a special folder for *pending elements*. The user can then manually reorganize these elements into appropriate locations.

Problem 4: Dynamic UI Generation
The *Vaadin*[4]-based web clients need to process instances that adhere to a metamodel which is unknown at compile time. Similar to the textual client, the web client needs to treat the metamodel as arbitrary. This raises the question of how to assemble a proper user interface in this scenario.
Solution: We employ a dynamic GUI generation approach which infers a UI widget for every element and property in the metamodel. For example, the inference mechanism will produce a Textfield widget for every attribute of type *String* with multiplicity one. For an attribute with an enumerated type, a Combo Box will be generated that contains the literals of the enumeration as choices. In a second step, the inferred widget receives its value directly from a given model element, effectively creating a databinding between GUI and model. When the model changes are committed, this binding is used in the inverse direction to apply changes made by the user directly to the model.

3 Problem Discussion: Element-based Metadata

As explained by Atkinson et al. [1], maintaining element-based metadata in an environment based on direct editing and parser technology (such as our text-based client) is a difficult problem. Metadata, such as unique identifiers and file locations, is usually not contained in the textual syntax for usability reasons. Therefore, it must be processed and maintained by the system in the background. Every time a file is changed by the user, a new AST is built by the parser. In our scenario, that AST (after minor modifications) is effectively the same as the resulting model. However, we only have element-based metadata for our current AST, not for the new one produced by the parser. Furthermore, there are no unique identifiers in the new AST that we could utilize to match two elements. For that reason, we have to rely on content-based matching. We use the *EMFCompare*[5] framework for this purpose, which identifies pairs of elements

[4] Vaadin Homepage: `https://vaadin.com/home`
[5] EMFCompare Homepage: `http://www.eclipse.org/emf/compare/`

(one from each parse tree) that refer to the same semantic object. Due to the undecidability of the model matching problem, resulting matches are only best-effort attempts. We have to make this trade-off of potentially losing element metadata in order to preserve the user experience of true textual editing, as opposed to indirect ("projectional") editing employed for example by Atkinson et al. It is important to note that the maintenance of element-based metadata across parse processes is not a strict requirement for the current tool implementation (the textual client is currently the sole producer and consumer of the metadata), but will be more important once other editors are added to the tool as well, which in turn may add more metadata to each element.

4 Conclusion & Outlook

In this paper we presented a modeling tool prototype that allows for the synchronization between textual and web-based modeling via a central model-repository. The motivation for this tool is the combination of these modeling paradigms to allow different stakeholder types to enter data in their desired format in the context of IT-architecture modeling. We highlighted the technical and usability challenges we faced in the implementation. We have shown that the different modeling paradigms like form-based and textual modeling pose a challenge in the implementation as well, in particular when it comes to element–based metadata in text–based environments that allow for direct editing.

The insights we gained from the here-described prototypical implementation are currently used to extend our EA tool *Txture*[6] with form-based modeling capabilities. We are also extending our approach of using multiple types of modeling editors. An editor e.g. using *Excel spreadsheets* is part of our current efforts.

References

1. Atkinson, C., Gerbig, R.: Harmonizing Textual and Graphical Visualizations of Domain Specific Models Categories and Subject Descriptors. In: Proceedings of the Second Workshop on Graphical Modeling Language Development. pp. 32–41. ACM, New York, NY, USA (2013)
2. Engelen, L., van den Brand, M.: Integrating textual and graphical modelling languages. Electronic Notes in Theoretical Computer Science 253(7), 105–120 (2010)
3. Farwick, M., Schweda, C.M., Breu, R., Hanschke, I.: A situational method for semi-automated Enterprise Architecture Documentation. Software & Systems Modeling (2014)
4. Farwick, M., Trojer, T., Breu, M., Ginther, S., Kleinlercher, J., Doblander, A.: A Case Study on Textual Enterprise Architecture Modeling. In: Enterprise Distributed Object Computing Conference Workshops (EDOCW), 2013 17th IEEE International. pp. 305 – 309. IEEE, Vancouver, BC (2013)
5. Matthes, F., Buckl, S., Leitel, J., Schweda, C.M.: Enterprise Architecture Management Tool Survey 2008. Tech. rep., Technische Universität München,Chair for Informatics 19 (sebis) (2008)

[6] Txture Homepage: `http://www.txture.org`