A Toolchain Transforming Descriptive Domain-Specific Models into Executable Browser-Based Applications

Matthias Sedlmeier[1] and Martin Gogolla[2]

[1] matthias.sedlmeier@mailbox.org
[2] Database Systems Group, University of Bremen, Germany
gogolla@informatik.uni-bremen.de

Abstract. While model-driven software development methodologies are still connected to non-agile approaches like the waterfall model, we aim to enable developers to use model-based evolutionary prototyping within agile software development techniques facilitating fast product increments through constant refinement. To reach this goal, we take advantage of the yEd diagram editor for creating, storing and updating graphical domain-specific models. With a defined transformation process carried out by a specially developed pair of Ruby tools, we rapidly generate working implementations from these models based on Ruby on Rails, a widely adopted solution for web application design. Our toolchain at hand, we demonstrate all steps required to provide executable browser-based applications from our descriptive domain models managed in yEd and suggest an agile software development approach.

Keywords: Domain-Specific Modeling · Graphical Modeling Language · Model Management · Model Transformation · Code Generation · Evolutionary Prototyping · Agile Software Development

1 Introduction

Modeling in general and Domain-Specific Modeling (DSM) are typically carried out within an approach-specific model management environment, like Eclipse [1], Epsilon [5] or Gemoc [3]. In contrast, this contribution advocates to employ the diagram editor yEd (yworks.com) for managing DSM artifacts, in particular for model storage and update. By allowing users to develop models with a well-known tool, we can realize a light-weight modeling approach for occasional modelers as well as domain experts and in contrast to heavy-weight modeling with, e.g., Eclipse. yEd models for browser-based information systems are transformed through a well-defined transformation chain into working Ruby on Rails (short Rails) implementations. Applications of yEd for modeling by computer and domain experts have been discussed already within software development [6] and ontologies [4], whereas [7], [2] and [10] deal with model-driven development of web based respectively mobile applications.

Our transformation chain involves two specially developed Ruby tools called *Tibet* and *Tor*. Tibet parses the domain-specific yEd models and creates an in-memory representation used by Tor to render the required Rails code artifacts. This process includes several refinement steps programmatically defined in the Ruby language and finishes with an executable application. We thus use Ruby as our transformation language. We aim to support developers implementing evolutionary prototyping within agile software development approaches supported by fast, model-based product increments, which can be easily customized.

This contribution builds upon preceding work covering conceptual data models [8] and model-driven design with yEd [9]. It presents a work still in progress, yet deployable. In Sect. 2 we explain some selected aspects of our DSM language mainly based on ER and EER concepts complemented with UML class diagram features. In Sect. 3 we discuss the transformation into a working implementation and suggest a possible agile software development approach based on our toolchain. Section 4 concludes the paper.

2 Utilizing a yEd Palette as a DSM Language

Before introducing our graphical DSM language, we take a brief look at yEd, a customizable general-purpose diagram editor serving as a front end for our model management. yEd provides a comfortable user interface that developers can utilize to specify requirements in the form of graphical domain-specific models stored as XML-based GraphML (Graph Markup Language) text files. Fig. 1 shows a yEd screenshot depicting six noteworthy parts. The window in part (1) provides an overview of the model loaded allowing developers to zoom and browse its contents. The window in part (2) holds automatically generated interactive context views of selected model elements focusing on the element *Neighbourhood*, *Predecessors*, *Successors* or *Folder Contents* in case of nested nodes. The *Structure View* in part (3) provides a searchable tree view of the model including all nodes represented by their labels. The main edit window in part (4) enables users to freely draw models providing various auxiliary features like guided or automatic layout, graph transformations and element grouping. Additionally, yEd ships with diverse palettes as shown in part (5), amongst others palettes for BPMN, Entity Relationship schemas, Flowcharts and SBGN (Systems Biology Graphical Notation). Developers can mix elements from different palettes and define custom ones via the palette management feature. Detailed information about elements are obtainable via the *Properties View* as shown in part (6).

Figure 2 shows an excerpt of our custom yEd palette for our DSM language, for which we give an exemplary explanation. The upper part (1) depicts various kinds of rectangular type nodes for modeling concepts on different abstraction levels. Developers are able to represent more concrete issues using the `EntityType`, `EnumerationType` and `RelationshipType`. For describing more abstract concepts, our palette offers the `ClassType` as well as the `ModuleType` (both being different w.r.t. association and attribute inheritance).

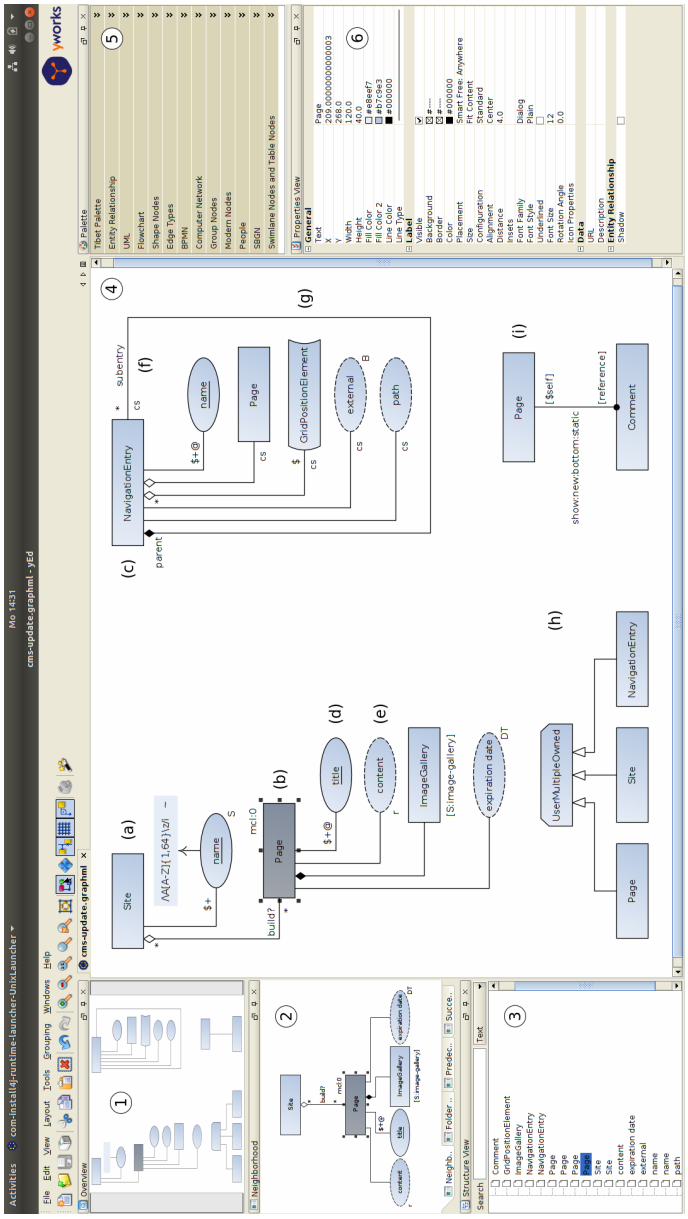

Fig. 1. Screenshot of yEd with DSM Example

Attributes are modeled utilizing oval nodes as displayed in the middle part (2) supporting different kinds of roles and data types. For example, final attributes labelled with *f* never change values after state initialization, while *browse* attribute values will be searched, if the application end user wants to link an

instance of the owning type via the user interface views. The definition of attribute validation rules can be accomplished using constraint nodes as shown in part (3). Nodes are connected via different edge types shown in the lower part (4). Amongst other concepts, our modeling language supports attribution, association, generalization and restriction edges.

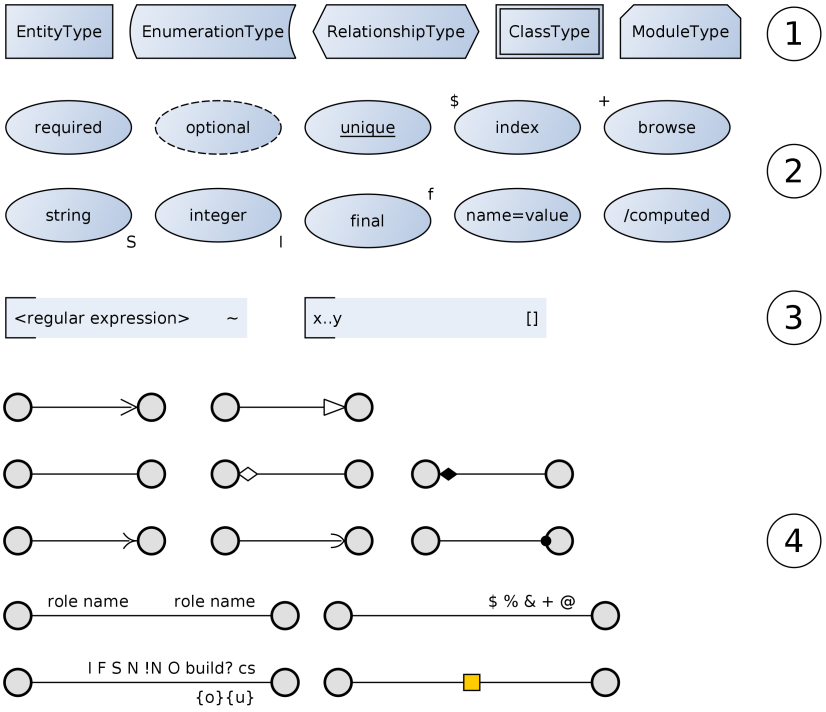

Fig. 2. Tibet Palette Excerpt

We use our language to specify an example application, a basic content management system. The following description provides a short summary of some selected aspects of the example model displayed in part (4) of Fig. 1, without claiming to be exhaustive.

For the content management system to work, we model some basic concepts like `Site` (a), `Page` (b) and `NavigationEntry` (c) as entity types with corresponding attributes and associations. In our scenario, the entity type `Site` may aggregate multiple `Page` instances and vice versa. Amongst others, the `Page` type holds a `title` (d) and a `content` (e) attribute connected via attribution edges decorated with semantic annotations (to be utilized in the transformation).

The `content` attribute is drawn with a dashed line making it optional, while the annotation `r` on the lower left side defines, that it will be represented via a *rich* text editor when rendered on the user interface views. The reflexive compos-

ite association as defined for the `NavigationEntry` (f) type allows to represent a potential hierarchy of navigation entries, which may be aligned on the user interface views by position information stored via the `GridPositionElement` (g) enumeration type. During transformation, additional attributes are augmented to store name constants and language specific translations. Our example also shows the inclusion of a module called `UserMultipleOwned` (h) by `Page`, `Site` and `NavigationEntry` inheriting required associations to the `User` entity type provided in another model file. Finally, (j) depicts a so-called supply relation between `Page` and `Comment` defining, that if a `Page` instance is shown on the user interface views, the end user is able to create a comment.

3 Transformation of the DSM language and Execution Utilizing Ruby on Rails

The complete transformation chain consists of four main steps as depicted in Fig. 3. Firstly, we use yEd to create and manage a descriptive domain-specific model based on our custom palette (1). Secondly, we utilize one of our specially

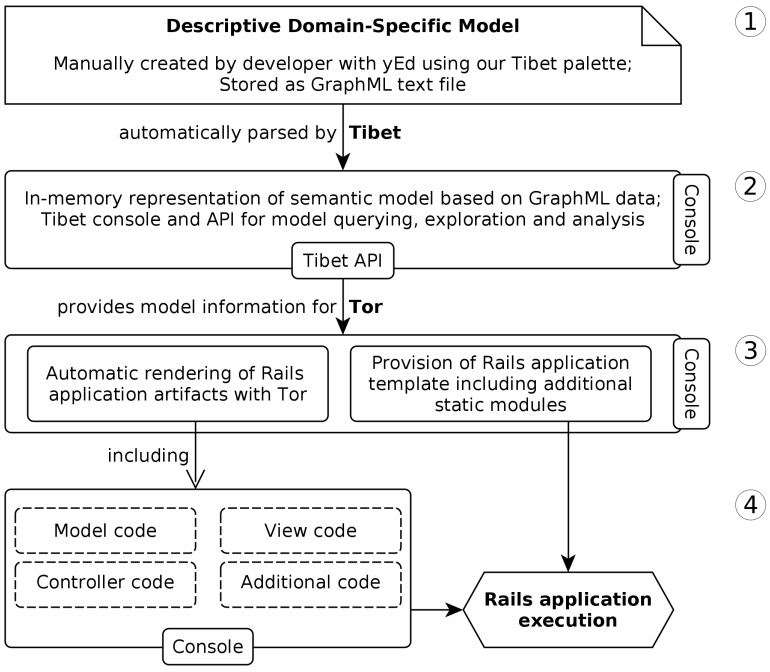

Fig. 3. Transformation Chain

developed Ruby-based tools called Tibet to instantiate an in-memory represen-

tation of the model. Tibet reads the domain-specific model, possibly spanning several files, and parses the GraphML content. Through several refinement steps it constructs a complete semantic model by performing well-defined operations expressed in the Ruby language (2). Using the additional Tibet console module, this structured process enables developers to manage model information, in particular to make meta queries for retrieving detailed information about the model down to its low-level GraphML representation.

The Tibet API component provides an interface allowing developers to access model information as done in the third step. Here we use our second specially developed Ruby tool called Tor implementing the necessary transformation logic to generate code modules by using specific templates. These modules are integrated unobtrusively into the derived Rails application instance and are complemented by additional static modules shipped with Tor (3). Tor also provides a console enabling developers to examine model details in the context of the generation process and thus helps in managing the transformation. In the fourth step, all necessary Rails modules are generated (4) and accessible via the Rails console.

Tor outputs the application data model components consisting of roughly two parts. Firstly, so-called migration files are derived containing schema information in a domain-specific language in order to initialize the database back end, in our case a Postgres SQL database. Secondly, Tor renders corresponding model classes based on the Rails ORM (Object-Relational Mapping) library implementing the active record architectural pattern. Tor also generates model validation and model access control layers as well as modules for reflection capabilities.

The application view components are also constructed from multiple generated files. On the one hand, Tor renders default HTML files, on the other hand, it builds JavaScript based front end views utilizing the React framework (reactjs.org) in conjunction with the jQuery (jquery.com) and Bootstrap (getbootstrap.com) libraries providing extended DOM (Document Object Model) manipulation features, user interface controls and layout mechanisms. As Tor builds JavaScript modules using the modern ECMAScript 2015 language specification (ecma-international.org/ecma-262/6.0), we use another open-source tool called Webpack (webpack.js.org) in conjunction with Node.js (nodejs.org) to transcompile the code and make it fully compatible with all modern browsers.

Additionally, Tor outputs application controller components connecting the model and view layers by rendering controller actions. These actions are mapped to corresponding HTTP request methods through extra generated routing definition files forming the application service endpoints. The data exchange between the browser-based front end and the Rails application back end is mainly realized with the widely-adopted JSON (JavaScript Object Notation) format.

Our toolchain at hand, we suggest a development process based on agile and rapid methods utilizing evolutionary and incremental prototyping. We are in the process of applying this approach to a larger case study covering the design and generation of a browser application for the individual creation of report templates and report instances for various documentation use cases. Dependent on the overall project size, we break our system into smaller parts individually

developed through multiple iterations. Each iteration starts with the creation of a data model in yEd and the generation of a first prototype derived by our toolchain. We then verify our requirements based on the prototype. We are able to adjust or extend the model, if required, and repeat the generation process until our expectations are met. Individual aspects, which cannot be expressed in our modeling language, are added unobtrusively with the help of various customization techniques. Our prototype evolves in each iteration until it is stable and ready for being part of our system. This structured process centers on the created model and is repeated until all increments of our system are captured.

The presented toolchain enables developers to perform simple and direct transformations from domain-specific models to executable applications. yEd as a light-weight diagram editor is freely available running on all major platforms. The latter also applies to Ruby, a widely-adopted and actively maintained open source programming language offering an easy and intuitive access harnessed by Rails, an established web application framework used by major platforms like GitLab and Airbnb. The depicted development process liberates developers from repetitive routine coding work and enables them to perform application refinement through model refinement. The domain-specific models identify central application concepts and represent a stable documentation. Our generation process facilitates fast iterations, which can be immediately validated by all relevant stakeholders. Thus, developers can take advantage of running evolving prototypes, which users can immediately interact with leading to early accepted increments and reducing the risk of project failure. Finally, if modularized properly, models and their increments are reusable in other projects.

4 Conclusion

We have demonstrated how we use yEd to create, store and update domain-specific models and how a toolchain is capable of generating browser-based client-server applications based on Ruby on Rails by applying corresponding transformations on those models. Our underlying DSM language targets web information systems and offers a rich collection of modeling features borrowed from (Extended) Entity-Relationship modeling and UML class diagram concepts. We have also suggested a feasible agile development approach we are using in our case studies.

As of now, our approach is limited to structural design. While some language features induce special application behavior, currently behavior cannot be modeled explicitly, but is projected in future tool versions. As already mentioned, we are in the process of planning and performing a larger project covering the design and generation of a browser application for the individual creation of report templates and report instances. The idea arose mainly from the need of flexible checklist based reporting for FSC (Forest Stewardship Council) audits and home inspection reports as recommended by InterNACHI (International Association of Certified Home Inspectors). Last but not least, applicability and practicabil-

ity of our approach will be further validated and improved through middle- and large-sized case studies.

References

1. Allilaire, F., Bézivin, J., Bruneliere, H., Jouault, F.: Global Model Management in Eclipse GMT/AM3. In: Eclipse Technology eXchange Workshop (eTX) - a ECOOP 2006 Satellite Event. Nantes, France (Jul 2006), https://hal.inria.fr/hal-01272277
2. Challenger, M., Erata, F., Onat, M., Gezgen, H., Kardas, G.: A Model-Driven Engineering Technique for Developing Composite Content Applications. In: Mernik, M., Leal, J.P., Oliveira, H.G. (eds.) 5th Symposium on Languages, Applications and Technologies (SLATE'16). OpenAccess Series in Informatics (OASIcs), vol. 51, pp. 11:1–11:10. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany (2016). https://doi.org/10.4230/OASIcs.SLATE.2016.11, http://drops.dagstuhl.de/opus/volltexte/2016/6016
3. Combemale, B., Barais, O., Wortmann, A.: Language Engineering with the GEMOC Studio. In: 2017 IEEE Int. Conf. Software Architecture Workshops, ICSA Workshops. pp. 189–191. IEEE Computer Society (2017)
4. Falco, R., Gangemi, A., Peroni, S., Shotton, D.M., Vitali, F.: Modelling OWL Ontologies with Graffoo. In: Presutti, V., Blomqvist, E., Troncy, R., Sack, H., Papadakis, I., Tordai, A. (eds.) Semantic Web ESWC Satellite Events. LNCS, vol. 8798, pp. 320–325. Springer (2014)
5. Kolovos, D.S., Paige, R.F., Rose, L.M., Williams, J.R.: Integrated Model Management with Epsilon. In: France, R.B., Küster, J.M., Bordbar, B., Paige, R.F. (eds.) Modelling Foundations and Applications, 7th Eur. Conf. ECMFA. LNCS, vol. 6698, pp. 391–392. Springer (2011)
6. López-Fernández, J.J., Garmendia, A., Guerra, E., de Lara, J.: An example is worth a thousand words: Creating graphical modelling environments by example. Software and Systems Modeling **18**(2), 961–993 (2019)
7. Neubauer, J., Frohme, M., Steffen, B., Margaria, T.: Prototype-Driven Development of Web Applications with DyWA. In: Margaria, T., Steffen, B. (eds.) Leveraging Applications of Formal Methods, Verification and Validation. Technologies for Mastering Change. pp. 56–72. Springer Berlin Heidelberg, Berlin, Heidelberg (2014)
8. Sedlmeier, M., Gogolla, M.: Design and Prototypical Implementation of an Integrated Graph-Based Conceptual Data Model. In: Thalheim, B., Jaakkola, H., Kiyoki, Y., Yoshida, N. (eds.) 24th Int. Conf. Information Modelling and Knowledge Bases (EJC). Frontiers in Artificial Intelligence and Applications, vol. 272, pp. 376–395. IOS Press (2014)
9. Sedlmeier, M., Gogolla, M.: Model Driven ActiveRecord with yEd. In: Welzer, T., Jaakkola, H., Thalheim, B., Kiyoki, Y., Yoshida, N. (eds.) 25th Int. Conf. Information Modelling and Knowledge Bases (EJC). Frontiers in Artificial Intelligence and Applications, vol. 280, pp. 65–76. IOS Press (2015)
10. Umuhoza, E.: Domain-Specific Modeling and Code Generation for Cross-Platform Multi-Device Mobile Apps. CoRR **abs/1509.03109** (2015), http://arxiv.org/abs/1509.03109