The Journal of Systems and Software 117 (2016) 612–637

Contents lists available at ScienceDirect

The Journal of Systems and Software

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

Automated design of multi-layered web information systems

Fábio Paulo Basso[a,*], Raquel Mainardi Pillat[a], Toacy Cavalcante Oliveira[a], Fabricia Roos-Frantz[b], Rafael Z. Frantz[b]

[a] Systems Engineering and Computer Science Department, Federal University of Rio de Janeiro, Rio de Janeiro, Brazil
[b] Department of Exact Sciences and Engineering, UNIJUÍ University, Ijuí, RS, Brazil

ARTICLE INFO

Article history:
Received 17 February 2015
Revised 10 March 2016
Accepted 25 April 2016
Available online 27 April 2016

Keywords:
Model-driven web engineering
Rapid application prototype
Domain-specific language
Prototyping
Automated design
Mockup
Experience report

ABSTRACT

In the development of web information systems, design tasks are commonly used in approaches for Model-Driven Web Engineering (MDWE) to represent models. To generate fully implemented prototypes, these models require a rich representation of the semantics for actions (e.g., database persistence operations). In the development of some use case scenarios for the multi-layered development of web information systems, these design tasks may consume weeks of work even for experienced designers. The literature pointed out that the impossibility for executing a software project with short iterations hampers the adoption of some approaches for design in some contexts, such as start-up companies. A possible solution to introduce design tasks in short iterations is the use of automated design techniques, which assist the production of models by means of transformation tasks and refinements. This paper details our methodology for MDWE, which is supported by automated design techniques strictly associated with use case patterns of type CRUD. The novelty relies on iterations that are possible for execution with short time-scales. This is a benefit from automated design techniques not observed in MDWE approaches based on manual design tasks. We also report on previous experiences and address open questions relevant for the theory and practice of MDWE.

1. Introduction

Model-Driven Engineering (MDE) (Kent, 2002) is a paradigm for model-based software development implemented by several techniques and used in several industrial contexts. In typical MDE-based processes, model transformations should receive a highly detailed model to generate working pieces of applications (Schmidt, 2006). To generate full source code, several parts of an application design are detailed in Domain-Specific Languages (DSLs) (Voelter, 2009) and/or decorated with annotations added to model elements represented with the Unified Modeling Language (UML) (Booch et al., 2005), a general-purpose modeling language commonly used. In any case, this makes the software construction dependent of design tasks.

In the development of web information systems, web front ends such as layout composed of Graphic User Interface (GUI) components (Vanderdonckt, 2005) and behavioral diagrams (Nunes and Schwabe, 2006) are usually represented. To allow the generation of full source code with an approach for Model-Driven Web Engineering (MDWE) (Rossi, 2013), these models are manually decorated with semantics for the actions of users, screen flows and business logic. It is possible to abstract implementation details using a design language, focusing on the specification of semantics in models that formalize the knowledge about software requirements (France and Bieman, 2001). Before the source code generation, these models can be further refined by designers, enabling clients to experiment an executable prototype in the end. This approach is known as *multi-view* (France and Bieman, 2001), and the model is created and enriched taking as input high-level abstractions of other models that map implementation details through model transformations.

The execution of a multi-view approach for MDWE may use design tasks that require months of work (Kulkarni et al., 2011; Zhang and Patel, 2011). Depending on the size of the software project and the adopted schedule in software process iterations, the effort invested in detailing models is seen as a reason to avoid the adoption of some of MDWE approaches (Whittle et al., 2013). Therefore, the ability to execute these tasks in short time-scales is a desirable feature in some contexts, such as in start-up companies (Rivero et al., 2014; Giardino et al., 2014).

* Corresponding author.
E-mail addresses: fabiopbasso@cos.ufrj.br, fabiopbasso@gmail.com (F.P. Basso), rmpillat@cos.ufrj.br (R.M. Pillat), toacy@cos.ufrj.br (T.C. Oliveira), frfrantz@unijui.edu.br (F. Roos-Frantz), rzfrantz@unijui.edu.br (R.Z. Frantz).

http://dx.doi.org/10.1016/j.jss.2016.04.060
0164-1212/

A possible solution to speed-up the modelling phase, thus helping in the execution of iterations in short time-scales, is the use of techniques for automated design (Linington, 2005; Batory et al., 2013). In this paper, we suggest the use of three different phases for constructing models for MDWE, namely: evolutionary, architectural, and functional. Models are based on the Model-View-Controller (MVC) architectural pattern (Evans, 2004). Although each prototyping phase is handled by some DSL and tools found in the literature, their integrated use is still a challenge in MDWE.

We present a methodology for MDWE named MockupToME Method, which includes tasks supported by (semi-)automated design techniques for some use case patterns (Molina et al., 2002). We extend previous contributions (Basso et al., 2014b), by detailing tasks and artefacts that include many DSLs, developed to support the design of many layers of MVC-based application models, and the tools associated with these tasks for automated design. We also summarized data collected from two software projects, the first considering mostly manual design tasks and the second considering the use of tasks based on automated design techniques.

A partially assisted design through Wizards was used in the first software project, with iterations planned for one month or more. In the second project, we used MockupToME Method with iterations planned and executed with one to two weeks. Both approaches are based on use case patterns of type CRUD (Souza et al., 2007), and use the same DSLs for representation of MVC-based application models, which are used in the end of a lifecycle for model transformations by the same source code generators. Differently, MockupToME Method includes DSLs and tools for designers to work in high-level of abstraction than in MVC-based application models.

The use of short iterations is a benefit observed in MockupToME Method, but not in our previous approach, i.e., in manual design of these models. The reasons why short time-scales are feasible in MockupToME Method has to do with the automated design techniques discussed in this paper. Thus, we also derived interesting research questions as a result from these two software projects.

The rest of the paper is organized as follows: Section 2, conceptualizes this work and Section 3 motivates this research; Section 4 exemplifies the representation of preliminary requirements, which are the input for the automated design approach introduced in Section 5; Section 6, describes the methodology, which is complemented in Section 7 with implementation details and in Section 8 with activities performed after the source code generation; Section 9, summarizes the two software projects, with lessons and insights for future research; Section 10, points out limitations; Section 11 presents the related work; and, finally, Section 12, reports on our main conclusions and possible future work.

2. Concepts

In the context of the development of web information systems, the following concepts are important for the understanding of this paper (Evans, 2004; Souza et al., 2007; Allier et al., 2015):

- **Model-View-Controller (MVC).** Is an architectural pattern (Parnas, 1994) frequently used in the construction of web information systems (Burke and Monson-Haefel, 2006). This pattern is important to modularize and structure the source code in three layers, thus facilitating the maintenance (Bosch, 2013) and avoiding the erosion of architectures as they evolves over time.
- **Conceptual model.** A class diagram composed of analysis classes, also named entities, which represents the *Model* layer of the MVC (Evans, 2004).
- **GUI Templates.** Facilitate the development of standardized structures for GUI (Han and Liu, 2010) allowing developers to focus on the logic layer, while layout details and actions are managed by a template engine. By means of templates, developers focus on the content that is placed inside a template structure.
- **CRUD.** A type of GUI template and an acronym for *create, read, update, and delete* (Souza et al., 2007) characterizing frequent set of use cases developed in information systems that allow to persist, retrieve and remove objects to/from a database. Different structures for CRUD can be used, and may include a specific GUI template.
- **Domain-Driven Design (DDD).** The *Model* layer is used to represent all the other application layers using a DDD approach (Evans, 2004). In MDWE, DDD drives the generation of a detailed MVC-based model, guiding the refinement of multiple layers associated with a particular use case scenario and a paper prototype.
- **Master/Detail.** A well-known concept among software developers, which allows the classification of use cases for *use case patterns* (Molina et al., 2002). These concepts of Master and Detail are well discussed in approaches for DDD (Evans, 2004) and the object oriented method (Molina et al., 2002).

The following concepts are important to contextualize our work:

- **Use case scenario.** Is one of possible flows from a use case (Sommerville, 2010) or user story (Landre et al., 2007). Use case scenarios are important both for design and for tests with clients (Sommerville, 2010), which evaluate models, prototypes and also the final version of an application piece with acceptance tests.
- **Paper prototype.** A hand drawing on a paper showing user interfaces with user interactions that represents use case scenarios (Sommerville, 2010). It is a software artefact represented in a high-level of abstraction than a mockup. A paper prototype is not a model, but a document usually associated with user stories specified in initial brainstorming meetings for the requirements elicitation. It is also called as pre-prototype (Davis and Venkatesh, 2004) and, sometimes, as throwaway prototype (Sommerville, 2010).
- **Mockup.** A model for a GUI, which is not possible to be fully implemented in functional prototypes (Blankenhorn, 2004; Rivero et al., 2014; Forward et al., 2012). In our understanding, mockups are abstractions in a high-level than the business logic needed in the development of web information systems, focusing on GUI components specification. Mockups may also be called sketches (Balsamic Mockups Company, 2015).
- **Round-trip engineering.** A set of activities aiming at synchronize generated source code with manually developed code (Mussbacher et al., 2014). It is performed automatically with the support of tools or, sometimes, manually, when it is required to update the model based on changes from source code.
- **Full source code generation.** Is the ability to generate 100% of what is designed, not 100% of all the application (Kelly and Tolvanen, 2008). Kelly and Tolvanen (2008) claim that full source code generation is a possible solution that mitigates the execution of changes in generated artefacts.

The Java platform is important for the implementation of web information systems and is divided in J2EE, J2SE, and J2ME editions. Burke and Monson-Haefel (2006) state that:

1. For the development of forms to desktop platforms, developers adopt J2SE and APIs such as AWT and Java Swing.