Improving Agility in Model-Driven Web Engineering

José Matías Rivero[1,2], Julián Grigera[1], Gustavo Rossi[1,2], Esteban Robles Luna[1], Nora Koch[3,4]

[1] LIFIA, Facultad de Informática, UNLP, La Plata, Argentina
{mrivero, julian.grigera, gustavo, esteban.robles}@lifia.info.unlp.edu.ar
[2] Also at Conicet
[3] Ludwig-Maximilians-Universität München, [4] Cirquent GmbH, Germany
kochn@pst.ifi.lmu.de

Abstract. The increasing growth of the Web field has promoted the development of a plethora of Model-Driven Web Engineering (MDWE) approaches. These methodologies share a top-down approach: they start by modeling application content, then they define a navigational schema, and finally refine the latter to obtain presentation and rich behavior specifications. Such approach makes it difficult to acquire quick feedback from customers. Conversely, agile methods follow a non-structured, implementation-centered process building software prototypes to get immediate feedback. In this work we propose an agile approach to MDWE methodologies (called Mockup-Driven Development, or MockupDD) by inverting the development process: we start from user interface mockups that facilitate the generation of software prototypes and models, then we enrich them and apply heuristics in order to obtain software specifications at different abstraction levels. As a result, we get an agile prototype-based iterative process, with advantages of a MDWE one.

Keywords: Mockups, User-Interface, Agile, Web Engineering, MDD

1 Introduction

During the last 20 years, many Model-Driven Web Engineering (MDWE) methodologies have been defined to improve the development process of web applications approaches [1-4]. All of these methodologies share a common top-down approach [5] and construct web applications by describing a set of models at different abstraction levels:

- *Content (or Domain) Model*: defining domain objects and their relationships.
- *Hypertext (or Navigation) Model*: defining navigation nodes and links that publish information specified by objects in the *Content Model*.
- *Presentation Model*: refining the *Hypertext Model* with concrete user-interface presentation features like pages, concrete widgets, layout, etc.

This process is generally top-down, delivering a final web application through a process of (sometimes automatic) model transformations which maps the previously described models into other models or a specific technology.

Agile methodologies, on the other hand, promote early and constant interaction with customers to assert that the software built complies with their requirements, by constantly delivering prototypes developed in short periods of time. Agile approaches argue that software specifications must emerge naturally, enhancing former prototypes along the development until the final application is obtained.

To summarize, while MDWE methodologies facilitate software specification portability, abstraction and productivity, they fail in providing *agile* interaction with customers because concrete results are obtained too late. On the other hand, while this feature is clearly provided by agile methodologies, they are heavily based on direct implementation and thus fail to provide abstraction, portability and productivity through automatic code-generation.

In this paper we propose an hybrid model-based agile methodology – called Mockup-Driven Development (MockupDD) – aiming to extract the best of both worlds, i.e. a process driven by the active participation of users and customers, and a classical approach following the phases of analysis, design and implementation assisted with the use of models in all stages. Our approach starts by the requirement analysis, i.e. defining mockups (ideally together with the customers) to agree upon the application's functionality, similar to Harel's behavioral programming approach [6]. Then, mockups are translated to an abstract user-interface model that can be directly derived to specific MDWE presentation models or technology-dependent UI prototypes. By tagging mockups and presentation models we add navigation features, and based on the navigation specification, we use heuristics to infer content models. Thus, we are starting the requirement specifications with objects that are perceivable by customers (UI structure elements), easing requirements gathering and traceability [7].

Therefore, since we start with presentation models obtained from mockups and then construct or obtain *upper* (i.e. abstract) models, we are inverting the traditional MDWE process, yielding to a more *agile*, yet truly model-based approach. While we exemplify with the UML-based Web Engineering (UWE) [3], MockupDD can be applied to any MDWE approach.

2 MockupDD by Example

User Interface (UI) Mockup tools like Balsamiq, Pencil or Mockingbird[1] suit well in agile methodologies [8-10], since they provide a quick and easy way of capturing interaction requirements. Usually, mockups are defined in companion with other specifications like use cases [11, 12], user stories [13] or informal annotations [14]. Also, mockups have been introduced in the context of model-driven development (MDD) approaches like ConcurTaskTrees [15]. In most cases, however, mockups themselves are not considered as models and they are usually thrown away after requirement modeling. Thus, mockups are not used as important drivers of the development process although they contain precise information about the users' needs.

MockupDD starts the development process by creating UI mockups with a mockup tool. As we have shown in a previous work [16], the resulting mockup files can be

[1] http://balsamiq.com, http://pencil.evolus.vn/en-US/Home.aspx, https://gomockingbird.com, last visited 18.3.2011

parsed and translated to an abstract UI model called *SUI model* (Structural UI Model) that can be in turn translated to presentation models of modern MDWE methodologies through a simple mapping, since most presentation metamodels (SUI included) usually share the same concepts (e.g., pages, panels, links, buttons, etc.). We propose to enrich SUI models using *tags*. Tags define simple but precise specifications that are applied over particular types of SUI elements and represent hints that can result in the derivation of particular MDWE model concepts.

In this paper we introduce *navigation tags* that enrich SUI models in order to derive navigation models. After obtaining both presentation and navigation models by the aforementioned mapping and tags semantics respectively, we apply heuristics to obtain the content model as well. We illustrate our process by showing how it works in the context of the development of a music catalogue application, deriving models for the UWE methodology. We have chosen UWE because it is representative of an important group of methods, it is based on UML and it has tool support. A schematic diagram of our process is shown in Figure 1.

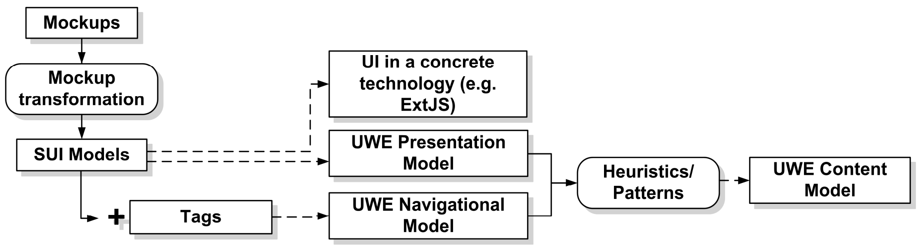
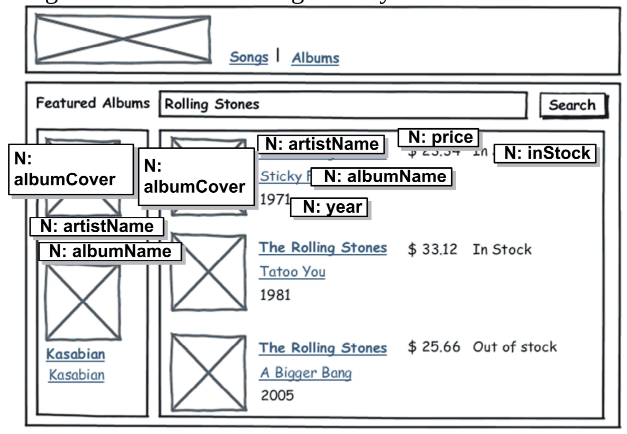


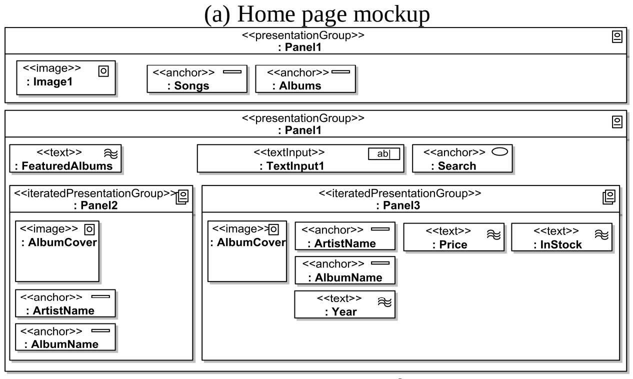
Figure 1. Mockup-Driven Development (MockupDD) process.

2.1 From Mockups to Presentation Models

The UI mockup (shown in Figure 2.a) depicts the home page of the Music Portal application containing a header, a list of featured albums, an album search box and its corresponding search result. Figure 2.b shows the corresponding UWE presentation model that can be obtained through a simple SUI-to-UWE presentation widget mapping. Some advanced features (like choosing whether to use an UWE `Presentation-Group` or an `IteratedPresentationGroup`) are inferred during the mockup transformation process through mockup analysis. The first problem that emerges is that the name of some widgets cannot be inferred; in these cases, a generic id is generated (like `Panel1`, `TextInput1` or `Image1`). Since correctly naming model elements with identifiers is important to reference them in the future and also for code or model derivation, we define a *naming tag set*, that allows redefining the name of some widgets when needed. The tagged mockup and resulting UWE presentation model are shown in Figure 2; note that naming tag starts with an `N:`. The use of naming tags implies that correct names are stored associated with SUI model elements and thus reflected in derived MDWE presentation ones. Also, when correctly applied, naming tags allow deriving mockup implementations for concrete technologies like ExtJS[2] using natural widget ids as when working directly with code.



(a) Home page mockup



(b) Generated UWE presentation model after applying naming tags

Figure 2. Deriving an UWE presentation model from a mockup.

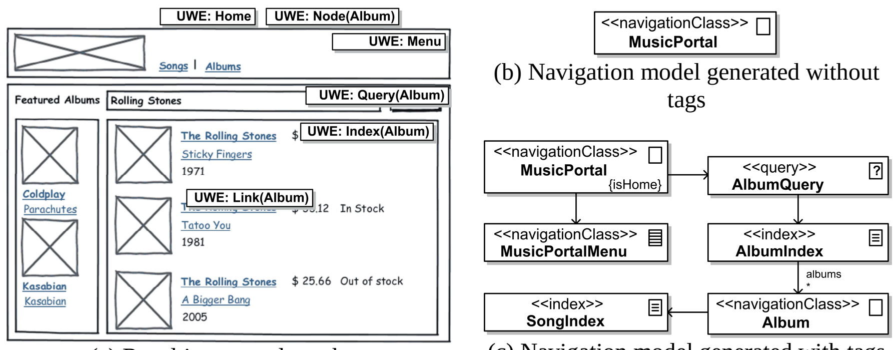
2.2 Deriving Navigational Models

After deriving presentation models, a naive approach to start generating navigation models could be defining one UWE `NavigationClass` (the UWE navigation concept for defining nodes) for each mockup. However, the UWE metamodel defines several navigation elements in addition to elements of type `NavigationClass`: `Query`, `Index` and `Menu`. While `Queryes` and `Indexes` represent information retrieval and selection of a particular element in a collection respectively, `Menus` are used to specify alternative navigation paths.

Since we cannot directly infer which UWE navigation element must be used in every mockup (this election requires design or modeling skills), we have defined a second tag set: the UWE navigation tag set. This set contains a tag for every UWE

[2] http://www.sencha.com/products/extjs/, last visited 18.3.2011

navigation element. Figure 3 shows the resulting tagged mockup and the consequences of tag application in derived UWE navigation model.



(a) Resulting tagged mockup

(b) Navigation model generated without tags

(c) Navigation model generated with tags

Figure 3. Initial mockup with UWE navigation tags applied and the resulting navigation model.

The UWE navigation tags introduced are the following:

- `Home`: defines that the `NavigationClass` related to the mockup is the home of the navigation model.
- `Node(<nodeId>)`: Assigns an id to the `NavigationClass` related to the mock-up in order to be referenced as the destination of one or more navigation (`Link`) tags.
- `Link(<nodeId>)`: Specifies a navigation link to another `NavigationClass`. A corresponding `Node` tag with the same `<nodeId>` must be specified in order correctly derive the navigation.
- `Query(<elementId>)` and `Index(<elementId>)` define a `Query` involving elements of type `<elementId>` and the `Index` in which the results of the `Query` are shown.
- `Menu` specifies that the panel over which it is applied is a set of links, a so called UWE `Menu`.

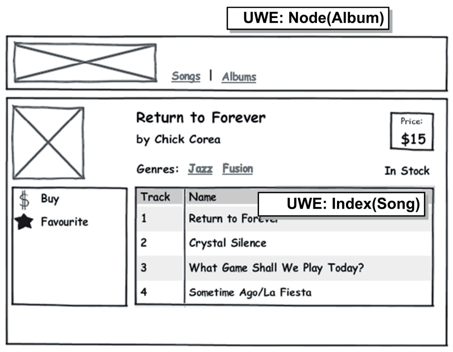


Figure 4. Album details mockup with UWE navigation tags applied.

When clicking on an album's title in the home page, an UI of the album details will be shown. A mockup of such user interface is denoted in Figure 4. The complete UWE navigation model can be observed in the already introduced Figure 3.c in which the `Album NavigationClass` is included. The navigation link is expressed through the `Link(Album)` and `Node(Album)` tags in home page and album mockups, respectively.

2.3 Towards a Content Model

Once we have obtained the UWE navigation model, a first version of the content model can be derived by applying some inference rules described in Figure 5. These rules were designed by studying many examples of UWE navigation and content models and discovering recurrent patterns in them.

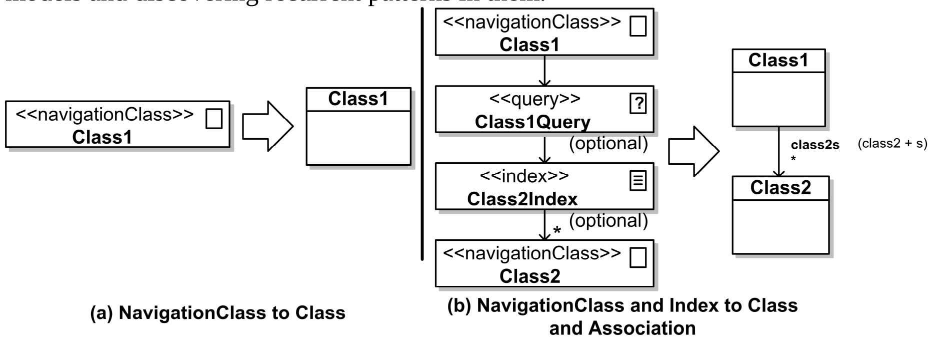


Figure 5. Two content inference rules.

UWE navigation element names (previously generated using naming and UWE navigation tags) are used to derive the names of the content elements. The resulting UWE content model after the application of the introduced rules over the UWE navigation model of Figure 3.c is shown in Figure 6 (for space reasons, only a part of the navigation model is shown).

The obtained UWE content models must be refined in order to specify class attributes. As UWE navigation models do not allow more refinement than the features already commented, this information should be taken from other models. Since in UWE every navigation concept is refined by a presentation specification (e.g., a `PresentationGroup`), and given that we have already derived these models from SUI specifications, we can use this link between models in order to obtain attributes from presentation structure. An example of this approach is denoted in Figure 7.

Automatic derivation may naturally lead to an imprecise content model, and some thoughtful design might be required from a developer in order to get to a definitive version. However, even when most design adjustments can not be fully automated, they can be still predicted. For example, an album presentation model might translate into an album class with attributes such as *artistName*, when in fact the content model should have two separate classes for `Album` and `Artist`, related to each other. We have observed that many of these inaccurate derivations usually repeat, so the required adjustments can be documented (and applied with automatic assistance when possible) just like code refactorings [17].

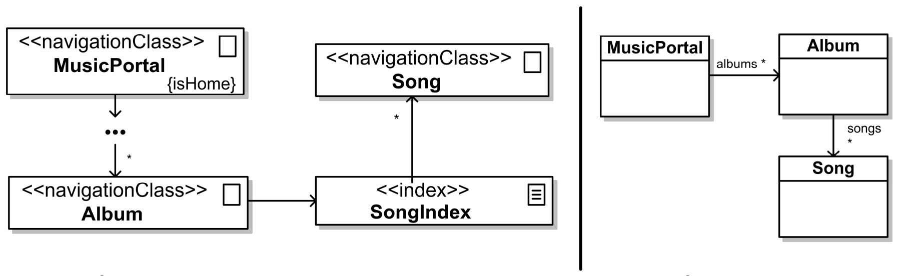


Figure 6. Inferred UWE content model derived through the application of the introduced rules.

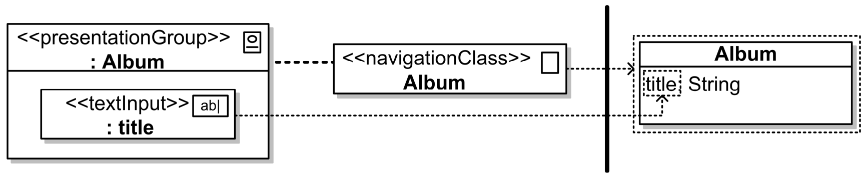


Figure 7. Attribute inference from presentation specifications.

3 Conclusion and Further Work

We have presented a mockup-based approach (MockupDD) pursuing an inversion of the traditional MDWE process. We decided to start our process with mockups because they are becoming a common tool in agile methodologies to interact and establish a shared view of requirements between customers and developers. Mockups are processed to structured UI models (called SUI) and with the help of tags they are easily derived to MDWE presentation and navigation models. Applying a set of inference rules, a first version of MDWE content models can be generated. We have shown the approach applied to a brief example using the UWE methodology. With our approach, we intend to provide an agile methodology based on UI mockups and lightweight specifications to obtain MDWE models, which offer advantages like automatic code generation.

Extending the proposed approach to other modern MDWE methodologies like WebML represents a fruitful work path. We are interested in defining a general and methodology-agnostic navigation tag set that also allow deriving navigation models for a more comprehensive set of MDWE approaches. Finally, since obtained content models likely require to be refactorized, we are interested in developing heuristics to suggest refactoring alternatives to be applied over content specifications.

4. References

1. Ceri, S., Fraternali, P., Bongio, A.: Web Modeling Language (WebML): A Modeling Language for Designing Web Sites. Computer Networks and

ISDN Systems, 33(1-6), pp. 137-157 (2000)

2. Gómez, J. and Cachero, C.: OO-H Method: Extending UML to Model Web Interfaces (2003). In: Information Modeling For internet Applications, pp. 144-173, P. van Bommel, Ed. IGI Publishing, Hershey, PA (2003)
3. Koch, N., Knapp, A.. Zhang G., Baumeister, H.: UML-Based Web Engineering, An Approach Based On Standards. In: Web Engineering, Modelling and Implementing Web Applications, pp. 157-191. Springer (2008)
4. Rossi, G., Schwabe, D.: Modeling and Implementing Web Applications using OOHDM. In: Web Engineering, Modelling and Implementing Web Applications, Springer, pp. 109-155 (2008)
5. Wimmer M., Schauerhuber, A., Schwinger, W., Kargl, H.: On the Integration of Web Modeling Languages: Preliminary Results and Future Challenges. In: Proc. of the 3nd Int. Workshop on Model-Driven Web Engineering (MDWE'07), CEUR-WS (2007)
6. Harel, D.: Some Thoughts on Behavioral Programming. In: Applications and Theory of Petri Nets. Springer Berlin Heidelberg (2010)
7. Seyff, N., Graf, F., Maiden, N.: End-user requirements blogging with iRequire. In: 32nd ACM/IEEE International Conference on Software Engineering - ICSE ’10. ACM Press, New York, New York, USA (2010)
8. Noble J., Biddle, R., & Martin, A.: The XP Customer Role in Practice: Three Studies. In: Agile Development Conference, pp. 42-54. IEEE Computer Society (2004)
9. Ferreira J., Noble J., & Biddle R.: Agile Development Iterations and UI Design. In: AGILE 2007 Conference, Washington, DC: IEEE Computer Society, pp. 50-58 (2007)
10. Ton, H.: A Strategy for Balancing Business Value and Story Size. In: Agile 2007 Conference. Washington, DC: IEEE Computer Society, pp. 279-284 (2007)
11. Kulak, D. & Guiney, E.: Use Cases: Requirements in Context. Addison-Wesley (2004)
12. Homrighausen, A., Six, H., & Winter, M.: Round-Trip Prototyping Based on Integrated Functional and User Interface Requirements Specifications. In: Requirements Engineering, 7(1), pp. 34-45 (2002)
13. Cohn, M.: User Stories Applied: for Agile Software Development. Addison-Wesley (2004)
14. Moore, J. M.: Communicating Requirements Using End-User GUI Constructions with Argumentation. In: 18th IEEE International Conference on Automated Software Engineering, pp. 360-363, IEEE Computer Society (2003)
15. Panach, J. I., España, S., Pederiva, I., & Pastor, O.: Capturing Interaction Requirements in a Model Transformation Technology Based on MDA. Journal of Universal Computer Science, 14(9), pp. 1480-1495 (2008)
16. Rivero, J. M., Rossi, G., Grigera, J., Burella, J., Robles Luna, E., Gordillo, S. E.: From Mockups to User Interface Models: An Extensible Model Driven Approach. In: 10th International Conference on Web Engineering, pp. 13-24. Springer (2010)
17. Fowler, M., Beck, K., Brant, J., Opdyke, W., Roberts, D.: Refactoring: Improving the Design of Existing Code. Addison-Wesley Professional (1999)