Towards Agile Model-Driven Web Engineering*

José Matías Rivero[1,2], Julián Grigera[1], Gustavo Rossi[1,2], Esteban Robles Luna[1,3], and Nora Koch[4,5]

[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] Also at CIC
[4] Ludwig-Maximilians-Universität München
[5] 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]. These methodologies share a common top-down approach [5] and construct web applications by describing a set of models at different levels of abstraction:

- *Content (or Domain) Model*: defining domain objects and their relationships.
- *Hypertext (or Navigation) Model*: defining navigation nodes and links that publish and manipulate information specified by objects in the *Content Model*.

* This work is an extended version of the paper "Improving Agility in Model-Driven Web Engineering", published in CEUR, Vol. 734.

S. Nurcan (Ed.): CAiSE Forum 2011, LNBIP 107, pp. 142–155, 2012.

- *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; application prototypes are then used as some kind of *common language* between developers and final users to assert captured requirements and to discover new ones. 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 Related Work

User Interface (UI) Mockup tools like Balsamiq[1], Pencil[2] or Mockingbird[3] suit well in agile methodologies [8-10], since they provide a quick and easy way of capturing

[1] Balsamiq - http://balsamiq.com/, last visited 3/9/2011.
[2] Pencil Project - http://pencil.evolus.vn/en-US/Home.aspx, last visited 3/9/2011.
[3] Websites wireframing: Mockingbird - https://gomockingbird.com, last visited 3/9/2011.

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 as can be appreciated in the use of UI sketches in the context of the ConcurTaskTrees [15] to express interaction requirements and in the definition of a language that introduces storyboards over user interfaces composed through a specific user interface widget set [16]. Finally, the result of statistical studies [17] conducted over inexperienced software engineers asserted that mockups effectively increases and eases software comprehension.

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. In our previous work [18], we introduced the idea of translating mockups constructed with prototyping tools like Balsamiq to a common presentation language in order to preserve and reuse them as truly software specifications. While this approach facilitates both a "*quick and dirty"* way of user interface construction with intense customer participation and a fast method to generate high quality UI from models (something that is not currently supported by well known MDWE approaches), it lacks the capacity of defining non-presentational features. As a consequence, mockups (and final UIs generated from them) can be used as a common language to gather further non-presentational requirements but these requirements must be coded by hand, missing the agility provided by automatic code generation in the context of a model-driven process. Here we go a step further and propose not only mockup reusing but the specification of advanced features like navigation and content through the application of a set of lightweight enrichments directly over them. This makes our approach easily understandable for all stakeholders, in particular customers and end users with the goal to involve them in all steps of the development process.

3 MockupDD Process

In this section we will introduce the MockupDD Process technically. First, we will show how mockups are refined and then translated into presentation models.

Then we will describe how similar refinements are applied in order to obtain navigation models. Finally, we introduce the heuristics applied to derive content models. In all the phases of our process we have chosen to use the UWE methodology because it is representative of an important group of methods, it is based on UML and it has tool support. An overview of the whole process can be observed in Figure 1.

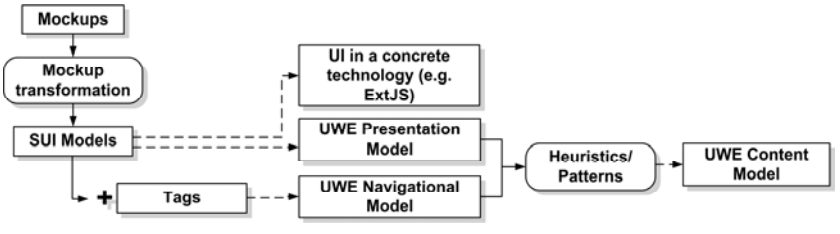
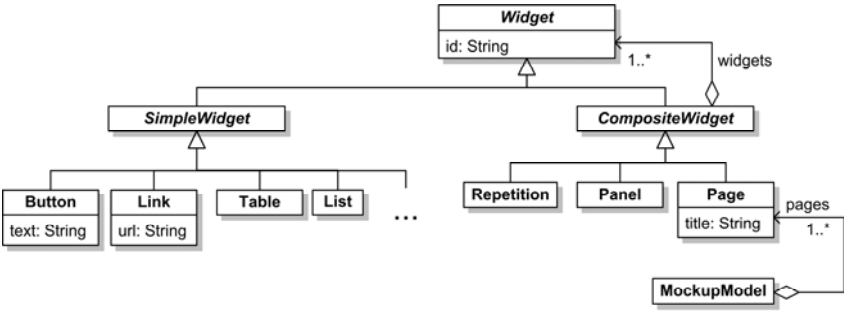


Fig. 1. Mockup-Driven Development (MockupDD) process

3.1 From Mockups to Presentation Models

MockupDD starts the development process by creating UI mockups with a mockup tool. As we have shown in a previous work [18], the resulting mockup files can be 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. In Figure 2, SUI metamodel is introduced and the mapping of its elements to UWE Presentation model is shown.



MockupDD Structural UI	UWE Presentation
Panel	PresentationGroup
Panel	Form
Image	Image
Button	Button
Link	Anchor
Label	Text
TextBox	TextInput
Repetition	IteratedPresentationGroup
CheckBox / RadioButton	Selection
Page	Page

Fig. 2. SUI metamodel and UWE mapping

Since mockup tools represent a user interface prototype as a set of unsorted widgets [18], we apply a sequence of different processors that analyzes mockup source structure and outputs the corresponding SUI model. Mockup source processing is done in a pipeline workflow. First, a set of widgets is obtained and validated from a mockup source file using a *Mockup Parser* and a *Validator* respectively. Then, widget composition and repetition is detected with the help of *Hierarchies* and *Repetition Detectors* analyzing the widget set obtained in the previous step. Finally, optimum layout for `CompositeWidgets` is inferred using a *Layout Inferer*. A graphical representation of the whole process can be observed in Figure 3.

3.2 The Tagging Approach

Structural UI models obtained from mockup source through the aforementioned processing represent only the structural view of a web application. In order to add different software features over the existing user interface specification we define the

concept of a *tag*. A tag defines a simple but precise specification that is applied over a concrete SUI element and is formed by a name and zero or more textual parameters. Every tag can be applied only over a particular subclass of `Widget` and represents a hint that can result in the derivation of particular MDWE model concepts. Moreover, tags are grouped into *tag sets* that can be combined to construct more complex specification. With the tagging approach we propose a simple, incremental and agile method to model features over previously defined user interface structure (SUI models).

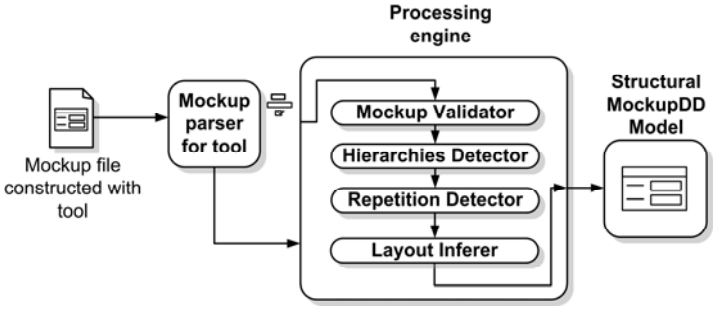


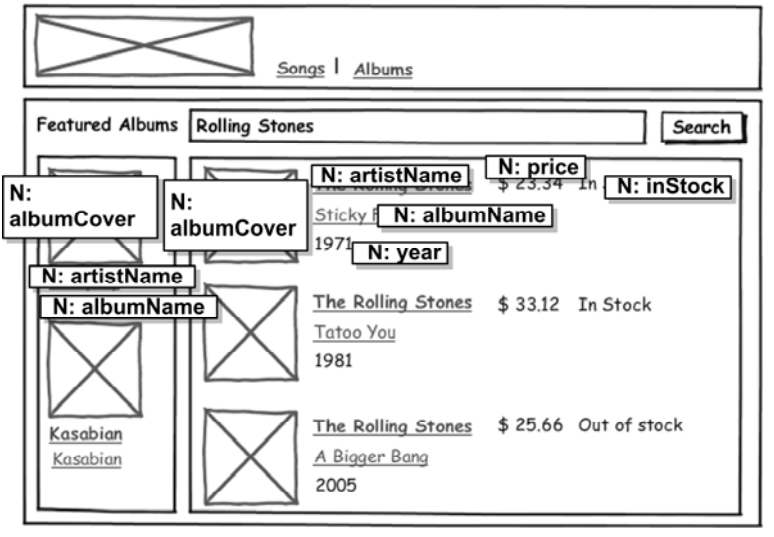
Fig. 3. Mockup processing workflow from original mockup source file until reaching final SUI models

In this paper we introduce *navigation tags* that enrich SUI models in order to derive navigation models. The UI mockup (shown in Figure 4.a) depicts the home page of a music catalogue application (we will call it *Music Portal*) containing a header, a list of featured albums, an album search box and its corresponding search result. Figure 4.b shows the corresponding UWE presentation model that can be obtained applying the previously introduced SUI-to-UWE presentation widget mapping. The *Repetition Detector* processor discovers similar widgets at equal positions and then generates a `Repetition` containing both album lists in the mockup, which are further translated into UWE's `IteratedPresentationGroup`s. By default, generic ids for controls are generated (like `Panel1`, `TextInput1` or `Image1`), but they can be refined using the *Name* tag (denoted with `N:`); these ids are important since they are used to name further MDWE elements. The tagged mockup and resulting UWE presentation model are shown in Figure 4.

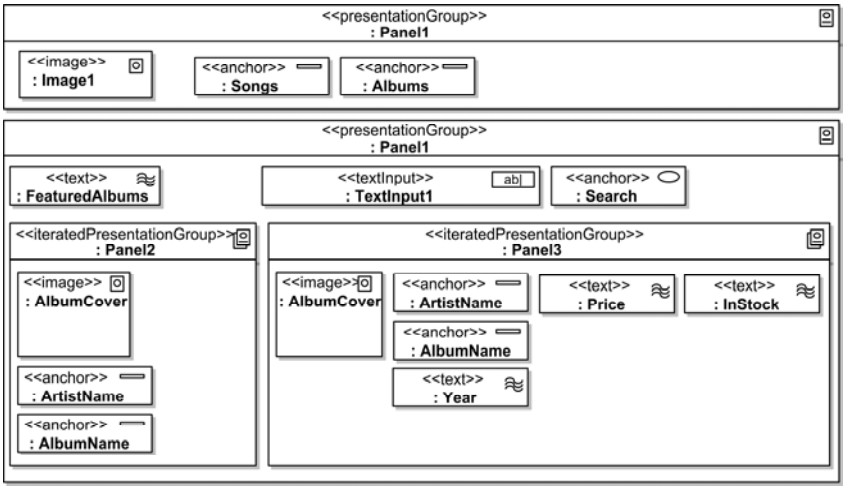
3.3 Deriving Navigational Models

After deriving presentation models, a naive approach to start generating navigation models could be defining one UWE `NavigationNode` (the UWE navigation concept for defining nodes) for each mockup. However, the UWE metamodel defines several navigation elements:

- `NavigationClass`, represents a generic navigable element in the hypertext structure,
- `Menu`, that is used to handle alternative navigation paths,
- `Query`, that is used to retrieve content from a data source, and
- `Index`, that allows selecting one content class instance from a set of instances that have been compiled during previous navigation.



(a) Home page mockup

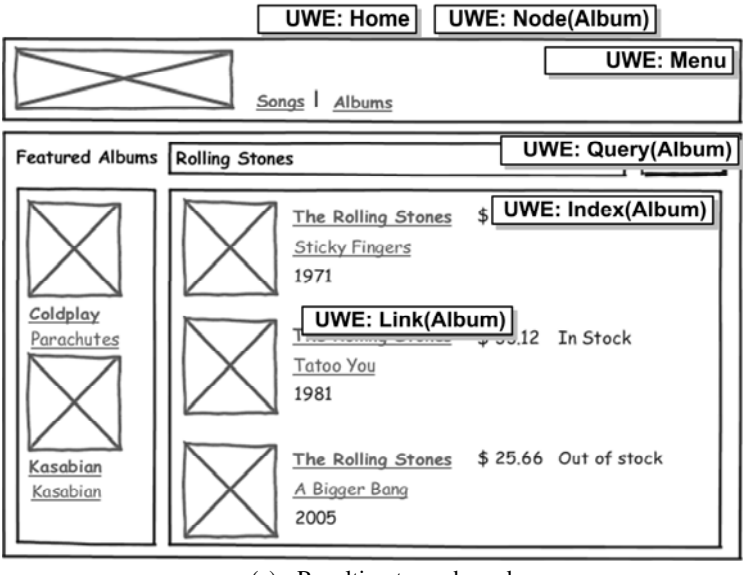


(b) Generated UWE presentation model after applying naming tags

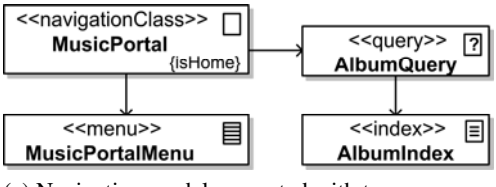
Fig. 4. Deriving an UWE presentation model from a mockup

Additionally, UWE links between navigation elements are expressed through `NavigationLink` instances.

Since we cannot directly infer which UWE navigation element must be used in a mockup as some alternatives are possible (for example, the content of a single mockup may include an UWE `NavigationClass`, a `Menu` and a `Query`), we have defined a second tag set: the UWE navigation tag set. This set contains a tag for every UWE navigation element. Figure 5 shows the resulting tagged mockup and the consequences of tag application in derived UWE navigation model.



(a) Resulting tagged mockup



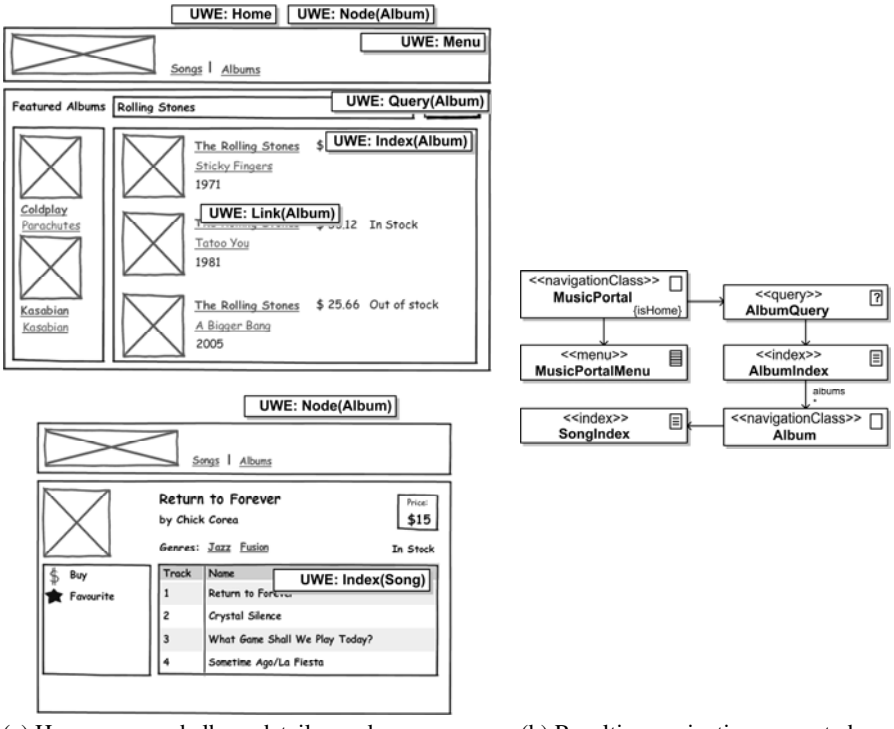
(c) Navigation model generated with tags

(b) Navigation model generated without tags

Fig. 5. 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 mockup 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 to 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`.



(a) Home page and album details mockups, property tagged with UWE navigation tags

(b) Resulting navigation generated from mockups in (a)

Fig. 6. Final version of tagged mockups and generated UWE models

When clicking on an album's title in the home page, an UI of the album details will be shown. After being defined, the mockup implementing the added functionality can be joined to the existing model through the aforementioned processing and further tagging, maybe in a new iteration. The *big picture* of the application being modeled can be observed in Figure 6 in which the complete tagged mockups and UWE model generated are depicted. The navigation link between the two existing mockups (SUI models in fact) is expressed through the `Link(Album)` and `Node(Album)` tags in home page and album mockups, respectively.

Some of the transformation rules that we defined (and implicitly applied in the previous example) are schematized in Figure 7.

3.4 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 graphically described in Figure 8. These rules were designed by studying many examples of UWE navigation and content models and discovering recurrent patterns in them.

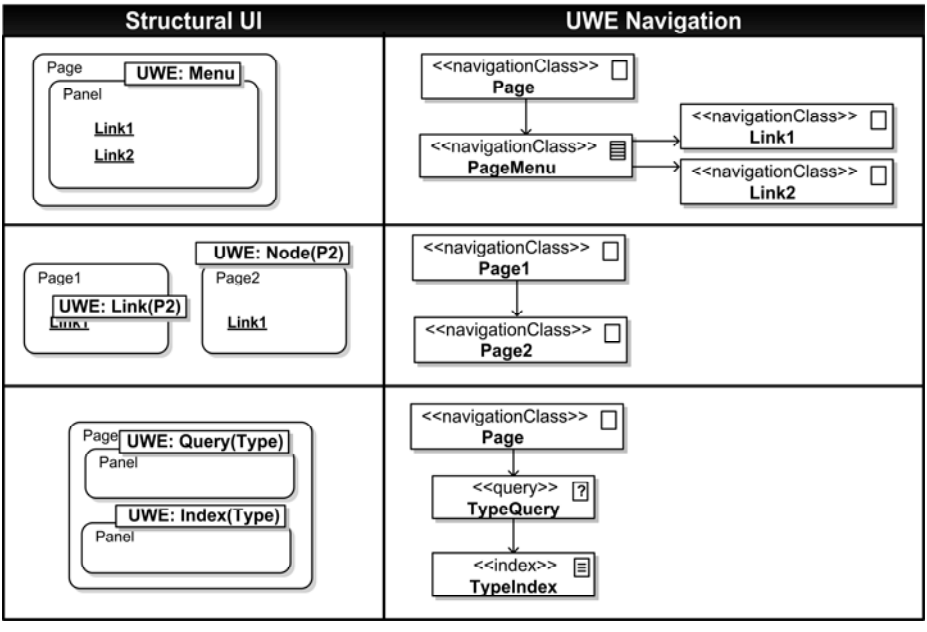


Fig. 7. Transformation rules applied over tagged SUI models to derive UWE features

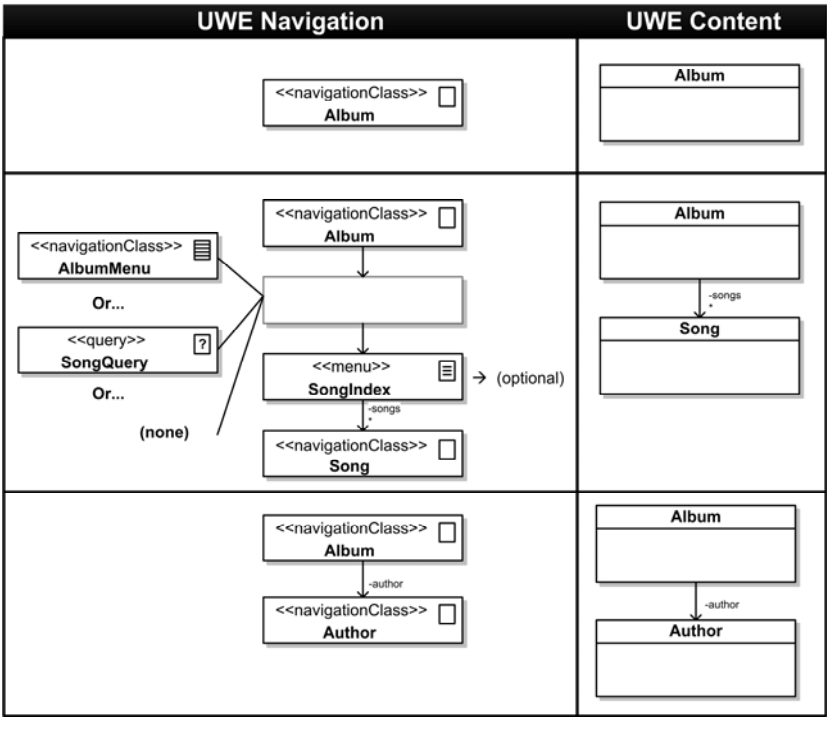


Fig. 8. Some content inference rules to generate UWE Content models from Navigation models

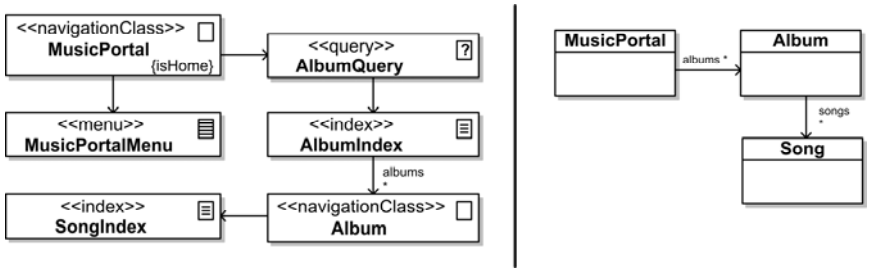


Fig. 9. Inferred UWE content model derived through the application of the introduced rules

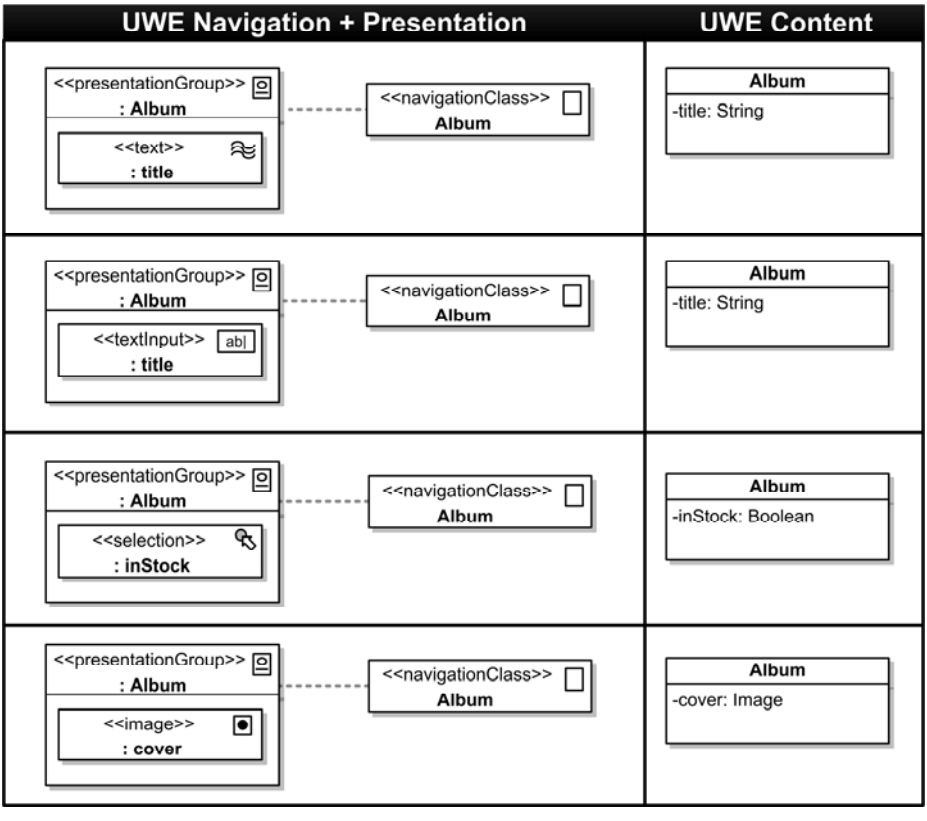


Fig. 10. Attribute inference combining existent navigation and presentation specifications

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 6.b is shown in Figure 9).

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. Rules using this link to infer attributes are graphically described in Figure 10.

4 Discussion

In this paper we presented an approach that adds agility to existing MDWE methods and we show how it can be applied in the context of UWE. The main intent of our approach is to enable early and constant communication and interaction with end users, a key requirement in agile methodologies. This interaction is facilitated in the early stages of development through the usage of UI mockups as a common language to start the process and discuss requirements; later, during further steps we provide an automatic and fast prototype generation through mockup enrichment and processing with help of our model-driven tooling. Thus, the MockupDD approach changes the traditional MDWE workflow, using presentation models (initially UI mockups) as the starting artifact in the process, and facilitating the incremental and iterative introduction of features through atomic tags over existing models. Since user interface are elements perceived by final customers users, they can be involved early and during every iteration.

Currently, since MockupDD is intended to provide an agile process to existing MDWE methodologies like UWE, WebML or OOHDM, it only allows specifying features which are present in these approaches, "inheriting" their applicability and limitations in different contexts. According to the current state of our research, many aspects of these methodologies can be generalized in a unified tag set while others must be refined with concrete tag sets, as explained in the following section. Additionally, the automatic derivation process commented in this paper may naturally lead to an imprecise content model, and some thoughtful design changes might be required in order to get to a definitive version. However, even when most design adjustments cannot be fully automated, they can be still predicted. For example, observing the examples in the previous section, an album class in the 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 are usually recurrent, so the required adjustments can be documented (and applied with automatic assistance when possible) just like code refactorings [19].

5 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 the iterative introduction of simple and precise refinements through the so-called *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

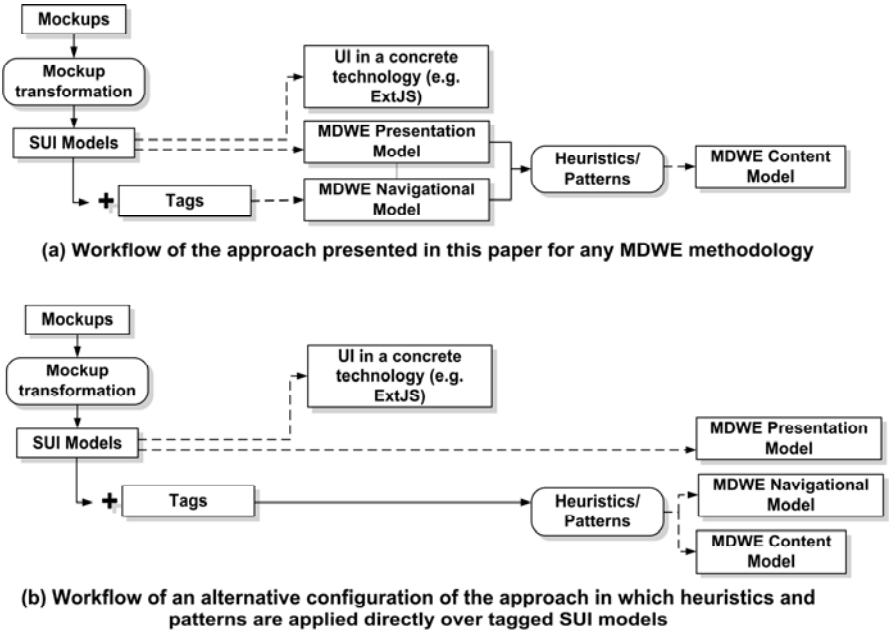


Fig. 11. Comparison of the approach presented in this paper (a) and an alternative approach that is being evaluated (b)

approach applied to an 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 (increasing software specification productivity) and a high level of abstraction (improving application portability) among others.

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 will allow us to derive navigation models for a more comprehensive set of MDWE approaches. We are experimenting with the application of heuristics not only at model level as is proposed in this paper, but also directly at the SUIT (SUI plus Tags) level; a comparison of both strategies is depicted in Figure 11. In addition, we are currently working in discovering and cataloging more heuristics and researching about how to define minimum tag sets that provide the highest expressive power and flexibility while preserving the simplicity of the approach.

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. Currently, experiments are being conducted to measure the differences found between MDWE models constructed entirely by hand from mockups and those generated automatically with the proposed tool. The delta found between those models will determine the definition of a catalogue of suggested refactorings and the heuristics implied in the detection of potential *bad smells* in automatically generated models in order to assist the improvement of their quality.

Finally, other approaches that propose *enriching* user interfaces in some way are Portlets and Mashups. While the former represent pluggable user interface elements that can be added to a *portal* page, the second propose to include and combine external services injecting one or more (in this case) UI components into a page. MockupDD propose to discover MDWE elements stereotyping existing user interface elements. However, an interesting branch of our research includes easing *Mashup* building through specific tag sets oriented to include user interface components of external services (e.g., Facebook buttons or comments). Also, we are considering the modularization and further reuse of common elements between mockups (something similar to the *Portlet* approach).

Acknowledgments. This work has been partially sponsored by the EU project ASCENS FP7 257414, and the DFG project MAEWA II, WI 841/7-2.

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), 137–157 (2000)
2. Gómez, J., Cachero, C.: OO-H Method: Extending UML to Model Web Interfaces. In: van Bommel, P. (ed.) Information Modeling for Internet Applications, pp. 144–173. IGI Publishing, Hershey (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, pp. 109–155. Springer (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 3rd Int. Workshop on Model-Driven Web Engineering (MDWE 2007). CEUR-WS (2007)
6. Harel, D.: Some Thoughts on Behavioral Programming. In: Lilius, J., Penczek, W. (eds.) PETRI NETS 2010. LNCS, vol. 6128, p. 18. Springer, 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 2010. ACM Press, New York (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, pp. 50–58. IEEE Computer Society, Washington, DC (2007)
10. Ton, H.: A Strategy for Balancing Business Value and Story Size. In: Agile 2007 Conference, pp. 279–284. IEEE Computer Society, Washington, DC (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. Requirements Engineering 7(1), 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), 1480–1495 (2008)
16. Mukasa, K.S., Kaindl, H.: An Integration of Requirements and User Interface Specifications. In: 6th IEEE International Requirements Engineering Conference, pp. 327–328. IEEE Computer Society (2008)
17. Ricca, F., Scanniello, G., Torchiano, M., Reggio, G., Astesiano, E.: On the effectiveness of screen mockups in requirements engineering. In: 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement. ACM Press, New York (2010)
18. Rivero, J.M., Rossi, G., Grigera, J., Burella, J., Luna, E.R., Gordillo, S.: From Mockups to User Interface Models: An Extensible Model Driven Approach. In: Daniel, F., Facca, F.M. (eds.) ICWE 2010. LNCS, vol. 6385, pp. 13–24. Springer, Heidelberg (2010)
19. Fowler, M., Beck, K., Brant, J., Opdyke, W., Roberts, D.: Refactoring: Improving the Design of Existing Code. Addison-Wesley Professional (1999)