Information and Software Technology 56 (2014) 670–687

Contents lists available at ScienceDirect

Information and Software Technology

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

Mockup-Driven Development: Providing agile support for Model-Driven Web Engineering

José Matías Rivero [a,b,*], Julián Grigera [a], Gustavo Rossi [a,b], Esteban Robles Luna [a,c], Francisco Montero [d], Martin Gaedke [e]

[a] LIFIA, Facultad de Informática, UNLP, La Plata, Argentina
[b] Research institute, Conicet, Argentina
[c] Research institute, CIC, Buenos Aires, Argentina
[d] LoUISE Research Group, UCLM, Albacete, Spain
[e] Chemnitz University of Technology, Germany

ARTICLE INFO

Article history:
Received 12 December 2012
Received in revised form 21 January 2014
Accepted 22 January 2014
Available online 29 January 2014

Keywords:
Mockups
User-Interface
Agile
Web Engineering
MDD

ABSTRACT

Context: Agile software development approaches are currently becoming the industry standard for Web Application development. On the other hand, Model-Driven Web Engineering (MDWE) methodologies are known to improve productivity when building this kind of applications. However, current MDWE methodologies tend to ignore important aspects of Web Applications development supported by agile processes, such as constant customer feedback or early design of user interfaces.
Objective: In this paper we analyze the difficulties of supporting agile features in MDWE methodologies. Then, we propose an approach that eases the incorporation of well-known agile practices to MDWE.
Method: We propose using User Interface prototypes (usually known as *mockups*) as a way to start the modeling process in the context of a mixed agile-MDWE process. To assist this process, we defined a lightweight metamodel that allows modeling features over mockups, interacting with end-users and generating MDWE models. Then, we conducted a statistical evaluation of both approaches (traditional vs. mockup-based modeling).
Results: First we comment on how agile features can be added to MDWE processes using mockups. Then, we show by means of a quantitative study that the proposed approach is faster, less error-prone and still as complete as traditional MDWE processes.
Conclusion: The use of mockups to guide the MDWE process helps in the reduction of the development cycle as well as in the incorporation of agile practices in the model-driven workflow. Complete MDWE models can be built and generated by using lightweight modeling over User Interface mockups, and this process suggests being more efficient, in terms of errors and effort, than traditional modeling in MDWE.
© 2014 Elsevier B.V. All rights reserved.

1. Introduction

In the last years, Model-Driven Web Engineering (MDWE) approaches like WebML [1], UWE [2] or OOHDM [3] have become mature solutions for developing Web Applications. These methodologies apply Model-Driven Development (MDD) concepts to capture high-level Web Applications concepts into models and use these models to derive applications automatically. The classic MDWE development process consists of three steps [4]: (1) building a domain model, (2) defining a hypertext model and (3) defining the application's look and feel. The result of the process is a set of models that can generate the final Web Application through code generation.

While standard MDWE processes improve productivity, they tend to leave User Interface aspects to the end of the development cycle [5]. As a consequence, customers only see their Web Application in action after a complete iteration and, at this point, they may encounter new requirements which may entail a full application rework. Changes in the UI may not only affect the application's presentation, interaction and usability aspects, but also the business logic [6,7], and in the context of MDWE, this may entail potential changes in the models at all three levels. In addition, while using high-level languages facilitates the requirements transformation to working software, the hard dependencies between each modeling step usually slows down the process.

* Corresponding author at: LIFIA, Facultad de Informática, UNLP, Calle 50 y 120, La Plata, Buenos Aires, Argentina. Tel.: +54 92215082703.
E-mail addresses: mrivero@lifia.info.unlp.edu.ar (J.M. Rivero), julian.grigera@lifia.info.unlp.edu.ar (J. Grigera), gustavo@lifia.info.unlp.edu.ar (G. Rossi), erobles@lifia.info.unlp.edu.ar (E. Robles Luna), fmontero@dsi.uclm.es (F. Montero), martin.gaedke@informatik.tu-chemnitz.de (M. Gaedke).

http://dx.doi.org/10.1016/j.infsof.2014.01.011
0950-5849/© 2014 Elsevier B.V. All rights reserved.

In order to solve this problem we have researched many of the emerging issues in agile development approaches, especially in the requirements engineering field [8]. In this context, user interface prototypes (usually known as *mockups or wireframes*), have proven to boost efficiency when capturing requirements of Web Applications [6]. One of their advantages is that they are technically valuable for developers and, at the same time, fully understandable by end-users [9]. In the approach we present detailed mockups (initially originated from User Stories [10]) are used to capture most requirements. We propose using mockups to specify software concepts using high-level models, breaking the linear MDWE workflow into small non-linear cycles in which end-users can actively participate. Then, instead of discarding mockups (as most agile approaches do [11,12]), we transform them into platform-independent UI specifications, and incrementally enrich them to later obtain a set of domain, navigation and presentation models – which can be also iteratively enriched. In this way, mockups serve both as an early requirements gathering tool and as a dedicated starting point for the model derivation process.

The resulting process is primarily based on models, but avoids the *waterfall-like* structure of classic MDWE methodologies. Building the *initial* mockups is the only mandatory step to start modeling, but since anyway mockups are used as requirement artifacts, this does not represent an additional overhead on the process. Once defined, requirements can be quickly modeled over mockups, reducing the requirement-to-software-artifact translation effort. As we show in the paper, this process fits well in agile methodologies like Scrum [13] and, at the same time, makes use of the well-known MDWE infrastructure to avoid *reinventing the wheel*. In face of the broad spectrum of Web Applications types, we decided to focus our approach on data-intensive ones (i.e., *applications whose main purpose is presenting large amount of data to their users* [1], as WebML [14] does), tackling mainly functional requirements. Nevertheless, some non-functional aspects like usability and presentation quality can be evaluated directly using mockups throughout the modeling cycle.

The main contributions of this work are: (1) showing how the traditional MDWE workflow can be improved to allow a less structured specification of concerns, adding agility to the whole process and (2) proposing a metamodel to specify data-intensive Web Applications concepts over user interface prototypes, and derive fully functional MDWE models. We also present a validation to show how the proposed process reduces the development effort in the context of a real-world example.

The paper is structured as follows: in Section 2 we present some background to the approach, in Section 3 we briefly describe an example application that we refer to throughout the paper. In Section 4 we describe our approach in detail and then in Section 5 we detail how it is architected and implemented. In Section 6 we share the results of our field experiences, and in Section 7 we comment related work. Finally, in Section 8 we draw some conclusions and present our future work on this field.

2. Background

Digital UI mockup tools are becoming a very popular way of rapidly drafting user interfaces. Their main purpose is to help in discussing UI specifications with end-users, and also to discover and define non-UI requirements in a language that is more familiar to them, as opposed to plain textual specifications [7,9]. In addition, UI mockups work not only as requirements artifacts, but also as general requirements elicitation helpers [15]. Ricca et al. present statistical studies that show how mockups improve requirements gathering in comparison to traditional textual methods, without implying an additional effort in the process [6]. Also, mockups have been proposed as a successful tool to capture and register *fluid* requirements [16] – those that are usually expressed orally or informally and are an implicit (and usually lost) part of the elicitation process. The importance of capturing additional information or requirements specifications associated to user interface prototypes (which is addressed in this work) is also commented on the work of Ravid and Berry [7].

User Interface prototyping and modeling is an extensively studied field. A plethora of UI modeling methodologies and environments exist [17,18]. UI prototyping tools have quickly emerged in the last years for both desktop (like Balsamiq[1] or Pencil[2]) and Web platforms (like Mockingbird[3] or MockFlow[4]). In this context, the DENIM tool [19] shows an interesting variation of the common *plain* mockup tools, offering interface sketching in different *conceptual* levels and widget drawing by hand. However, these tools are rather focused on building UI sketches bound to be disposed after requirements gathering, or on defining user interfaces in a top-down fashion to generate running applications. In the context of our work, we propose to use mid to high-fidelity UI mockups [20,21], built either with the mentioned tools or by designers in plain HTML. Then we propose to keep them, using their structure as a foundation to specify features like content, navigation or business logic.

Model-Driven Web Engineering (MDWE) approaches like WebML, UWE or OOHDM have a long track proposing improvements in the Web Development field. The main motivation of MDWE is to define the essential aspects of the Web Application using a high level language and then generating the running Web Application automatically, thus promoting more productivity. Similarly, in this approach developers spend their time specifying semantically relevant aspects instead of coding, avoiding coding errors.

We already proposed a technique and tool [22] to include digital User Interface mockups in a Model-Driven process, transforming them into valuable UI specifications that can be used to generate code for several platforms and technologies. By using this background, we also introduced the idea of annotating the obtained models to specify MDWE concepts over the original mockups and then generate final models both for UWE and WebML methodologies [5,23]. In this work, we propose to unify all the methods and tools proposed, describing a detailed UI Mockup-Driven process to develop Web Applications.

3. Photo Stock: an example application

In order to explain all the stages in the approach, we will use a Photo Stock website as a sample Web Application. The Photo Stock Web Application was designed analyzing several similar websites, which typically enable users to upload original pictures, sort them in categories/folders and optionally sell their publication permission. Besides these basic functionalities, the application provides the user with a personal blog to post contents related to their photography portfolio or production and it also offers forums where it can participate in discussions.

In the first meeting with our customers, we got some User Stories for a first iteration:

- **User Story 1**. As a User, I want to create, delete and change the name of folders so that I can store and manage my photos.

[1] Balsamiq Mockups – http://www.balsamiq.com/products/mockups, accessed: 07-Sep-2012.
[2] Pencil Project – http://pencil.evolus.vn, accessed: 07-Sep-2012.
[3] Website Wireframes: Mockingbird – https://gomockingbird.com, accessed: 07-Sep-2013.
[4] MockFlow – Online Wireframe Tool – http://www.mockflow.com, accessed: 07-Sep-2013.

- **User Story 2**. As a User, I want to upload new photos to an existing folder.
- **User Story 3**. As a User, I want to publish a new post in my personal blog.

In the next sections we will show how the different steps in the methodology are executed in order to satisfy some of these User Stories.

4. The MockupDD process

As we previously mentioned, our approach starts by building UI mockups; thus, we decided to call it *MockupDD* (standing for Mockup-Driven Development). Although our approach supports different kinds of mockups, like those created with digital tools (Fig. 1a), throughout our example we will use HTML mockups (Fig. 1b), since they are closer to what end-users see in the final application, and also because they are natively supported in our tooling. In this section we present a general description of the approach (Section 4.1), the different ways to get the UI mockups and their underlying model representation (Section 4.2), how these mockups can be interactively enriched through atomic specifications (Section 4.3), how these atomic specifications are disambiguated in a semi-automatically way to derive MDWE models (Sections 4.4 and 4.5) and the underlying process that guides the modeling (Section 4.6).

4.1. The process in a nutshell

The MockupDD process starts with a quick requirements gathering stage, which results in a set of User Stories to be fulfilled. Customers or end-users then build the mockups to represent User Stories graphically (Fig. 2, step 1). Then, the development team captures the main concepts in these mockups in a *Structural User Interface* (*SUI*) model, preserving the mapping between both.

Once all the User Stories are represented with mockups and then mapped to SUI models (Fig. 2, step 2), the requirements are specified in the form of *tags*. To do this, the development team works together with the stakeholders to interpret the semantics in the User Stories and mockups, and *tag* the mockups with annotations that represent these semantics. Since the SUI model preserves the mapping from the original mockup source, tags can be *seen* as applied over the initial mockups, but they are in fact linked to SUI elements. This permits keeping a tool-independent enrichment strategy (since SUI models do not depend on any platform, tool or UI technology) and, at the same time, allows application of the specifications intuitively over the initial mockups in their original graphical representation, facilitating requirements traceability and understanding by end-users.

We will show in Section 4.3 how the tags are used in an unstructured way to help specifying concepts and features iteratively. The result of this step (Fig. 2, step 3) is a completely enriched (tagged) SUI Model. At this point, stakeholders can immediately run a demo to test that requirements were correctly captured and met (Fig. 2, step 4a). All steps, from mockup building to the running demo can be applied in an incremental way, until all the requirements for the iteration are met or larger requirements (entailing longer implementation times) are found. If the new functionalities involve altering the mockups, the process should start from step 1, creating new mockups or refining existing ones, enriching them and then generating a new running demo of the application. On the other hand, if newly discovered requirements do not require changes in the UI, the process can start from step 3, enriching existing mockups until the requirements are fulfilled. Both cases involve crucial interaction and communication between the development team and customers or end-users to assert that the resulting product is working as expected.

The whole iteration finishes with the generation of models from the tagged SUI and the further derivation of the final application (Fig. 2, step 4b). Specifications can be finally refined with tool support to make design and modeling decisions unnoticed during the interaction with customers. Fig. 3 summarizes the actors involved in each step of the MockupDD process and their responsibilities.

4.2. Step 1 and Step 2: mockup construction and processing

The MockupDD process starts by building UI mockups. The number of mockups needed in each iteration depends on how the interaction steps are distributed amongst the Web Application's navigation nodes. Also, several User Stories can share one or more mockups since they can involve crosscutting functionalities.

To formalize the user interface structure, we use the SUI metamodel [5], whose core structure can be observed in Fig. 4a. The SUI metamodel in MockupDD is very similar to popular UI description languages and standards like UsiXML [24], XAML[5] or XUL[6]. It defines a `Widget` abstract class; widgets may be `SimpleWidget`s (atomic) or `CompositeWidget`s (container widgets), and they are grouped in *navigational* units (`Page`s). However, instead of only defining a UI structure, the SUI metamodel is designed to support the specification of an extensible and customizable set of features using UI elements through *tags,* which will be explained in the following section.

In Fig. 5 we show some mockups built for the User Story 1 of the Photo Stock website example. A SUI model instance has been extracted from those mockups after their creation, referencing and mapping the main parts of the application's UI – i.e., the widgets that are important from the point of view of the Story. In the mockups shown in Fig. 5, the highlighting is not part of the mockup, but a visual decoration added by the interactive tool to denote the widgets already mapped to underlying SUI elements.

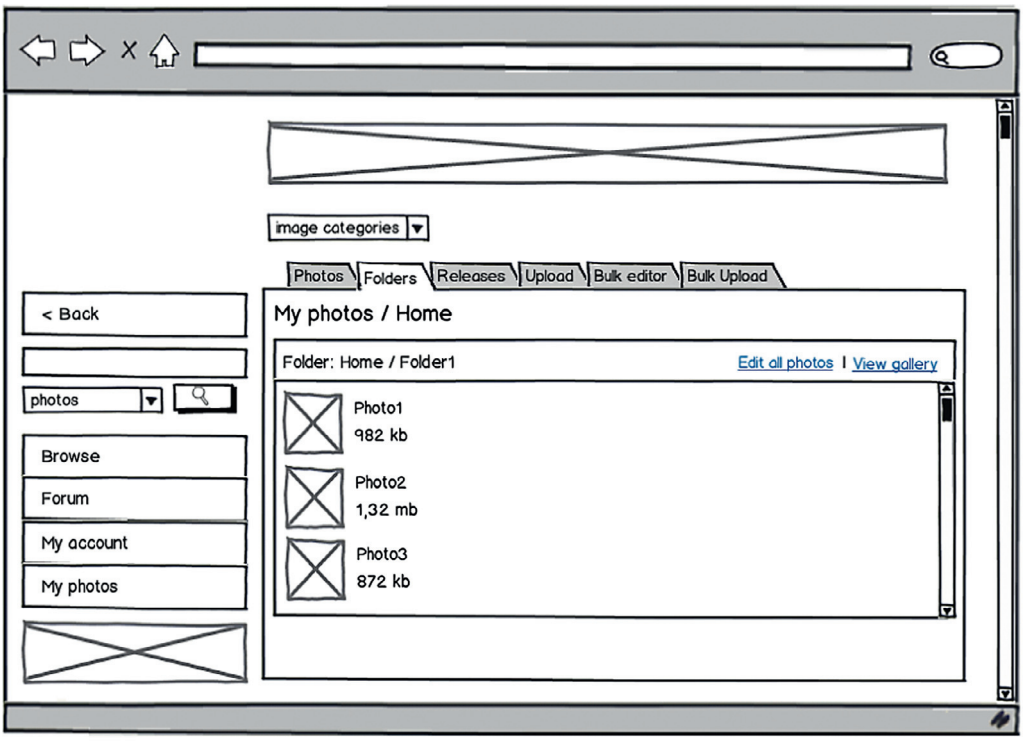
Once we have highlighted and mapped all the relevant widgets according to the User Story concepts, we move onto the specification of features on these widgets.

4.3. Step 3: Features specification

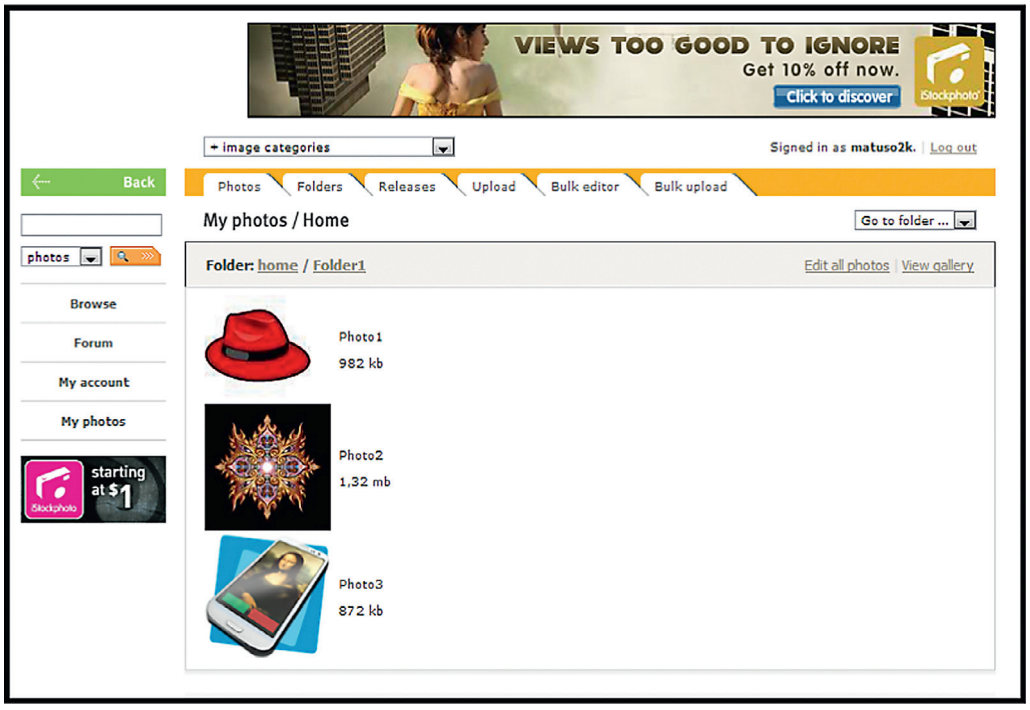
In order to enrich mockups (in this step, SUI models) with different kinds of specifications, we introduce the concept of *tag*. A tag is an atomic enrichment composed by a name and zero or more textual parameters (e.g. `Tag(Param1, Param2, ... ParamN)`. Every tag type can be applied only over a subset of SUI widgets. Also, every tag can define a custom syntax for each parameter, extending its semantics as much as needed. Tags are grouped in *tag sets* in order to isolate specific concerns like navigation or data-manipulation that can be tackled separately in most cases. Through this separation, we avoid the sequential dependencies between concerns specified during traditional MDWE modeling that delay the direct product testing by customers or end-users – e.g., navigation features that cannot be specified without a previous domain model. Fig. 6 describes all the tags included in MockupDD specification, along with their semantics, applicability and corresponding tag set. Tags semantics can be used to discover and map requirements from end-users when discussing the UI structure, and also to specify how the final application must behave. For instance, a `Data(Folder)` and `Save(Folder)` (see Fig. 6) can be applied when an end-user or customer expresses

[5] XAML Overview (WPF) – http://msdn.microsoft.com/en-us/library/ms752059.aspx, accessed 07-Sep-2013.

[6] XUL | Mozilla Developer Network – https://developer.mozilla.org/en/docs/XUL, accessed 07-Sep-2013.



(a) Mockup drawn using the Balsamiq tool



(b) Mockup built using HTML

Fig. 1. Sample mockup drawn using different approaches (tool vs. HTML).

that a particular mockup should create an instance of a `Folder` using the input fields contained in it. A *Folder* is a domain object that was identified by end-users and, at the same time, mapped to a software specification by developers using tags (a `Folder` entity or class). In the same way, if another end-user states that right after instantiating a Folder the application must show the updated lists of folders, a `Link(folders)` tag can be added – where `folders` is the name of the destination mockup. These two requirements (the former related to data model and manipulation, and the latter defining a navigation behavior) can be defined independently, in any order and even as the result of requirements specified and discovered over the mockups by different stakeholders.

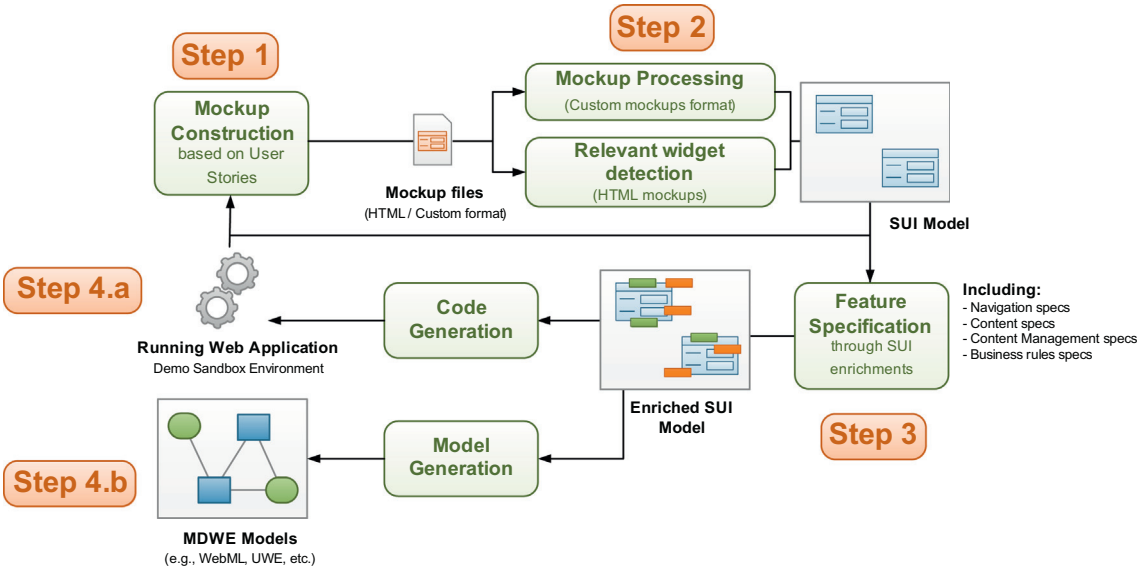


Fig. 2. The workflow of a MockupDD iteration, including technical steps.

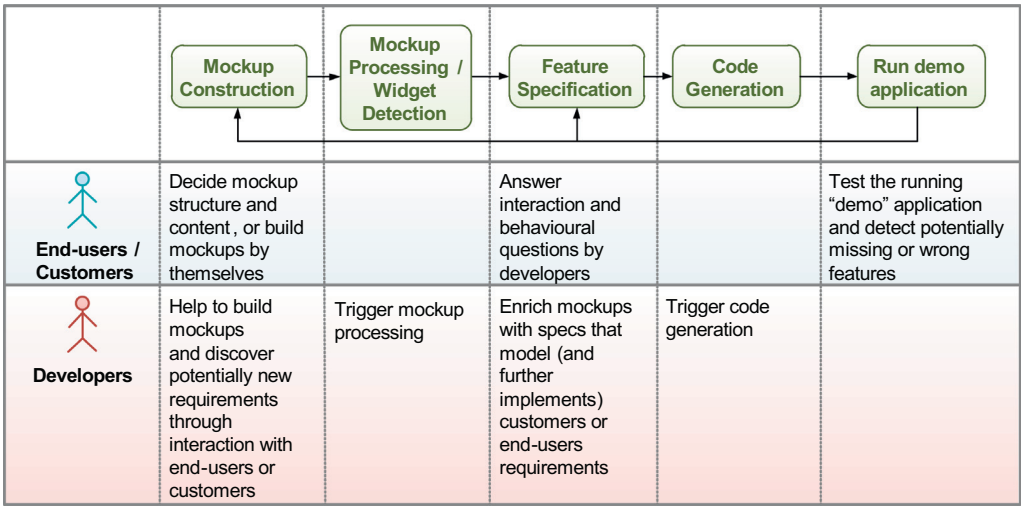
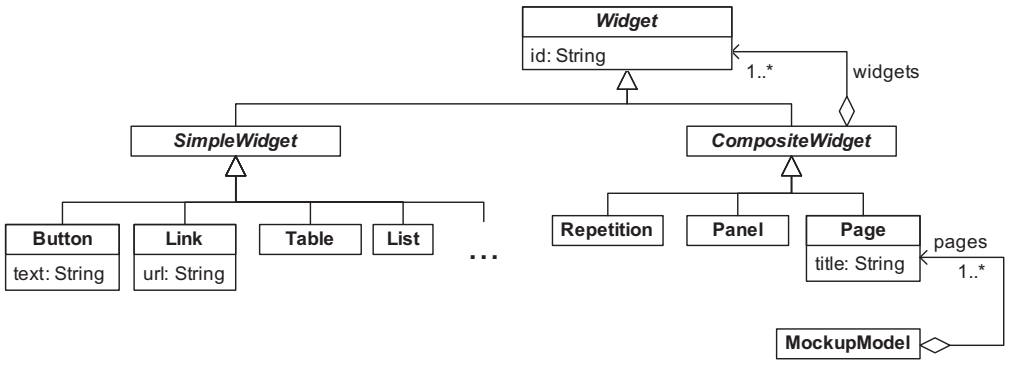


Fig. 3. Actors involved in every MockupDD process stage.

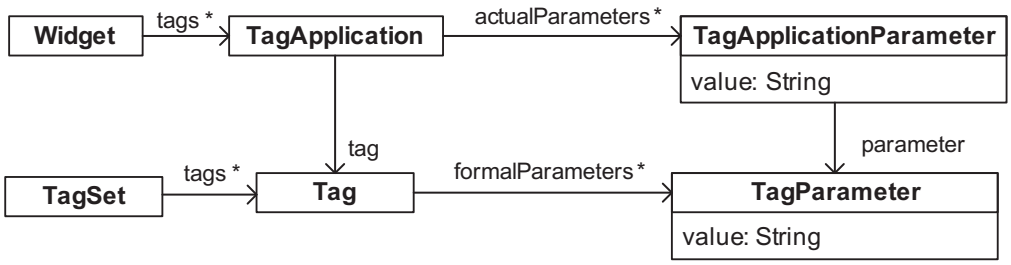
In Fig. 4b we depict how tags are formally structured, grouped and applied *over* SUI Widgets from the MockupDD metamodel perspective. As the figure shows, tag sets (`TagSet` class) are conformed by a list of tags (`Tag` class), which in turn can have parameters (`TagParameter` class). The `TagApplication` class represents the application of a `Tag` over a particular `Widget`. Since one or more tags can be applied over a single `Widget`, `Widget`s have a list of `TagApplication`s. A `TagApplication` simply references the Tag applied and the corresponding actual values for the parameters required for it (a list of `TagApplicationParameter`).

The detailed requirements expressed in the User Story 1 of the example of the Photo Stock website can be implemented with the application of the following tags (depicted in Fig. 7):

- The `Data(Folder)` tag applied over the `Panel` in Mockup2 indicates that it will edit or create instances of `Folder`.
- The `Data(Folder.name)` and `Data(Folder.description)` tags, applied over different `TextBox`es in Mockup2 (see Fig. 7), imply that these textboxes will enable setting and obtaining the values of the `name` and `description` attributes of a `Folder` instance.
- The `Save(Folder)` tag, applied over the *Submit* button in Mockup2, indicates that the default action for clicking it will be storing (creating or updating) an instance of the `Folder` being edited in the container `Panel`.
- The `Data(Folder)`, `Data(Folder.name)` and `Save(Folder)` tags in the lower `Panel` in Mockup1 have the same semantics as the previously commented tags, implementing a *quick folder creation*.
- The `Data(Folder)` in the upper `Repetition` (Mockup1) is used to specify that it will show a list of `Folder`. In addition, the description of the `Folder` will be shown in the internal `Label` tagged with `Data(Folder.description)`.
- The `Delete(Folder)` tag over the `Link` labeled *Del* indicates that clicking it will trigger the deletion of the `Folder` associated to the containment context (in this case, the row in the `Folder` table).
- The `Link(EditFolder)` – visually covered by the `Delete()` tag – and `Transfer(Folder)` tags applied over the *Edit* `Link` specify that a click on it will produce a navigation to the `EditFolder` page (Mockup2), transferring the `Folder` in the containment context as a parameter. Since Mockup2 already

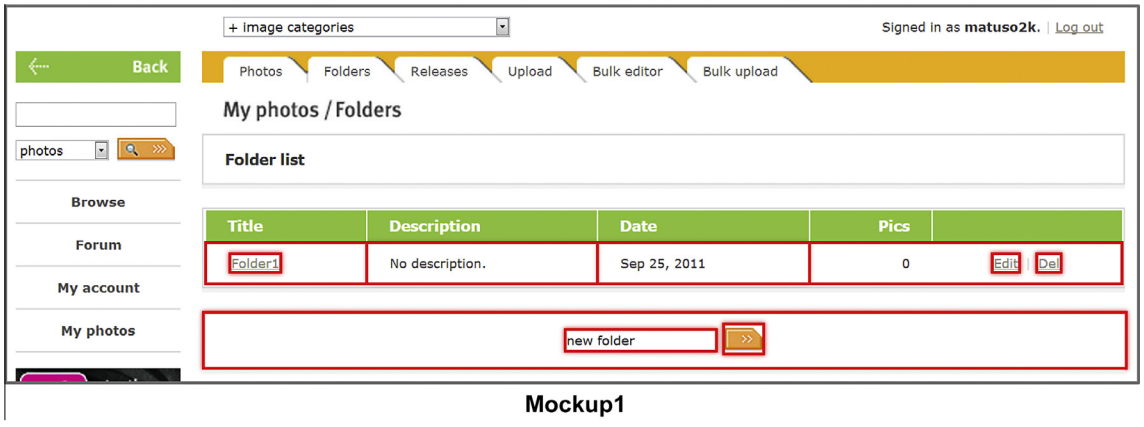


(a) Core SUI metamodel

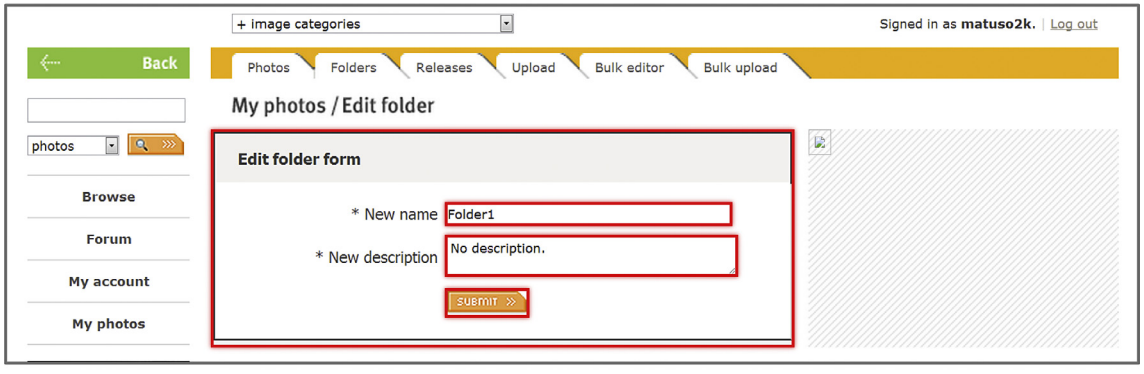


(b) Tag metamodel

Fig. 4. The SUI Metamodel.



Mockup1



Mockup2

Fig. 5. Mockups built for User Story 1.

Tag set	Tag structure and semantics	Applicable over
Navigation/Interaction	Node(<nodeId>) "This page will be indentified with the id <nodeId>"	Page
Navigation/Interaction	Link(<nodeId>) "Clicking this Link/Button will trigger a navigation to previously identified <nodeId> Page"	Button/Link
Content	Data(<typeName>) "This widget must show or allow editing a(n) <typeName>"	Widget
Navigation/Interaction	Select() "This widget will be used to mark objects in order to perform some action"	SimpleWidget
Navigation/Interaction	Transfer(<type1, …, typeN) "When navigating, <type1>, … and <typeN> will be transferred to the destination page"	Button/Link
Content	Save([<type1, …, typeN>]) "Clicking that Link/Button will save <type1>, … <typeN>"	Button/Link
Content	Delete([<type1>, …, <typeN>]) "Clicking that Link/Button will delete <type1>, … <typeN>"	Button/Link
Content	Associate(<type1>, <type2>[, <associationName>]) "Clicking that Link/Button will specify that <type1>'s <associationName> will be <type2>"	Button/Link
Content	Dissociate(<type1>, <type2>[, <associationName>]) "Clicking that Link/Button will specify that <type1>'s <associationName> will not be <type2> anymore"	Button/Link
Content	Query(<description>, <type>) "Widgets content in this panel will be used to get a list of <type> through the query expression <description>"	Panel
Content	Action(<action> [, <type1>, …, <typeN>]) "Clicking in that Link/Button will trigger a(n) <action> action, involving <type1>, …, <typeN>"	Button/Link

Fig. 6. MockupDD's main tag set description, semantics and applicability.

allows creating and storing instances of `Folder`, the `Folder` passed as a parameter can be edited using the same UI. With this last addition, the form in Mockup2 will allow the creation of new `Folder`s (when no `Folder` is passed as a parameter) and also the edition of existing ones – when a `Folder` is passed as a parameter.

While tags (and their implied concerns) can be applied in any order, one tag can specify more than one concern; for instance, the `Data(Folder)` tag associates not only a `Panel` to an instance of `Folder` to show its data in the UI, but also specifies that a `Folder` class exists in the domain model. In the same fashion, `Data(Folder.description)` and `Data(Folder.name)` tags bind individual UI widgets to instance variables values, and at the same time specify the properties `name` and `description` for the class `Folder` in the domain model.

4.4. Tags refinements

Tags are intended to be as conceptual as possible, following the *just barely good enough* agile modeling practice [25]. They enable specifying concepts in an incremental way, by refining their parameters or combining their semantics when there are many of them applied to the same widget, or related to the same mockup. We define an *ambiguous tag* as a tag not semantically complete by itself, and that may have more than one valid semantic interpretation. While quicker to apply, these tags do not have concrete expressive value until they are *reified*. In some cases, this reification can be automatically done using heuristics, for instance:

- The `Data()` tag (see Fig. 6), provides a basic `Data(<typeName>)` syntax. Also, this tag allows the form `Data(<widgetId>:<typeName>)` to specify the source widget `(<widgetId>)` which will provide the data instance of type `<typeName>` explicitly. If a list (named `listl`) and a panel (named `panell`) both tagged with `Data(Folder)` are found in a common `Page`, then it is assumed that the latter shows the data selected from the former and thus, the tag in it will be changed from `Data(Folder)` to `Data(listl:Folder)`.
- If two different `Page`s have the same `Data(<typeName>)` tag applied over one of their internal widgets and a navigation is expressed from one to another using the `Link()` tag, then a `Transfer(<typeName>)` tag can be automatically added since this navigation will potentially involve an object transfer as a side effect in most cases. This may have been applied in Mockup1 (Fig. 7), if the `Transfer(Folder)` tag had been missing.

4.5. Step 4: code and model generation

Up to this point, we have presented tags as isolated and atomic specifications with specific semantics. In this section we show how they can be translated to MDWE elements and combined to specify more complex design features.

Every iteration of the MockupDD process involves adding, changing or removing tags. After tags have been completely defined, we use model generators to obtain the corresponding MDWE models. So far, we have defined and implemented generators for UWE [23] and WebML [5], showing that (1) the tag metamodel can be semantically rich as modern MDWE metamodels and (2)

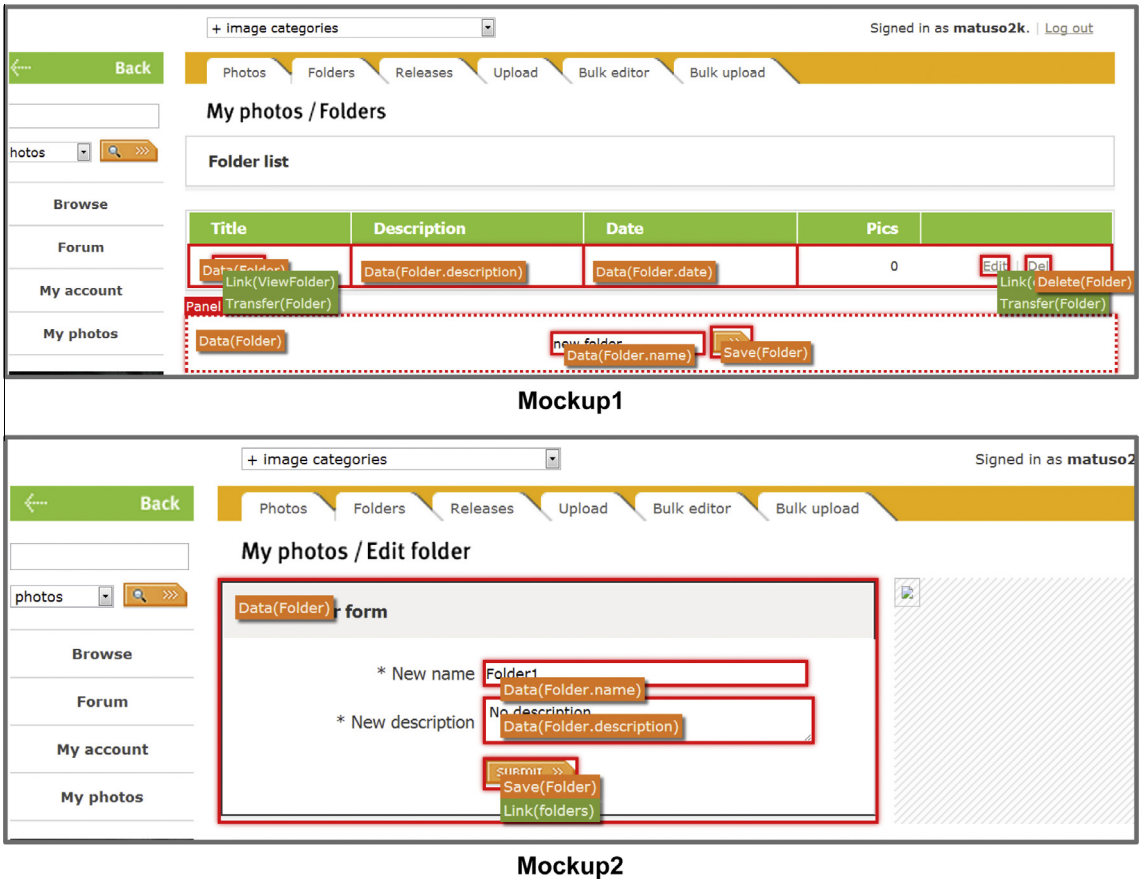


Fig. 7. Mockups of Fig. 5 with MockupDD tags applied, describing the expecting behavior.

code generation capabilities for existing methodologies can be used, gaining the inherent productivity of code generation in Model-Driven methodologies.

In Fig. 8 we show how isolated tags can be derived into atomic MDWE concepts for WebML and UWE – we only show a small subset of MockupDD-to-MDWE transformations for space reasons. Sometimes tags enable discovering and specifying elements at different design levels and designing models at the same time (for example, a `Class` in content model and a WebML's `Index` or `Query` at the hypertext level).

When the tagging task is finished (and the ambiguous tags reified), we obtain a complete tagged SUI model. With this model we are able to run an instant demo of the application, and generate MDWE models. In both cases, the first step in the generation process is to derive a *Spec Model*. This model is a more precise one that helps reducing the translation effort to code or MDWE models. In the Spec Model, the inner textual representation of the tags (e.g., class names for `Data()` tags, destination page names for `Link()` tags, etc.) are represented through a set of linked Spec Objects. Then, instead of processing, parsing and merging all the textual information in tags once for every MDWE model generator (for instance, UWE and WebML), they only have to iterate over the graph of Spec Objects and generate the destination MDWE concepts.

In Fig. 9 we show a WebML model generated from the tagged mockups of Fig. 7, using the MockupDD model generation features. A WebML model like the one depicted in the figure is composed by Pages and Units that allow showing and manipulating information within Pages. More details about WebML concepts can be found in [1].

4.6. MockupDD development process

So far, we have introduced the procedural and theoretical aspects of MockupDD. In this sub-section, we will briefly introduce MockupDD's development process, which is an adaptation of *Scrum* [13]. Hence, for a better understanding of our process, we will briefly describe the Scrum development process first.

The Scrum process (see Fig. 10a) starts with the construction of a Product Backlog, which is a list of all the features that the software product must have, prioritized by value delivered to the customer. Then, the product is built iteratively in so-called *Sprints*. Every Sprint starts with a planning meeting to develop a detailed plan for the iteration in which the most important features remaining in the Product Backlog are broken down into tasks, forming the Sprint Backlog. Once this backlog is carefully defined to take no more than a month, the development team starts working on it. A short Daily Scrum Meeting is run every workday to share the Sprint work progress and new problems found during it. Finally, at the end of a Sprint, a potentially shippable application is demonstrated to the Product Owner, then the Product Backlog is reprioritized if needed and the goal for the next Sprint is defined.

With MockupDD (Fig. 10b), the first step of a Scrum Sprint follows the same strategy, except that the Sprint Backlog must be expressed as a list of User Stories whose detail level should be enough to be described in detail using mockups. Instead of using direct coding, mockups are constructed to concretize the User Stories as detailed in Section 4.1. These mockups are then translated into a SUI representation and tagged with the tool support. After

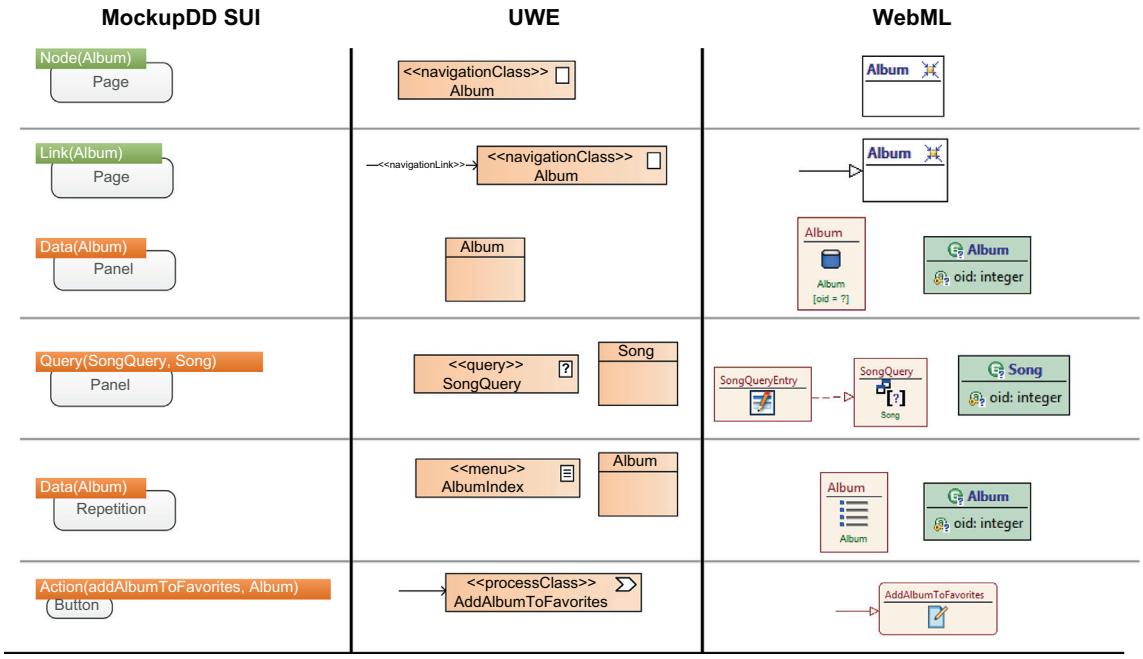


Fig. 8. Mapping from tagged SUIs elements to MDWE concepts.

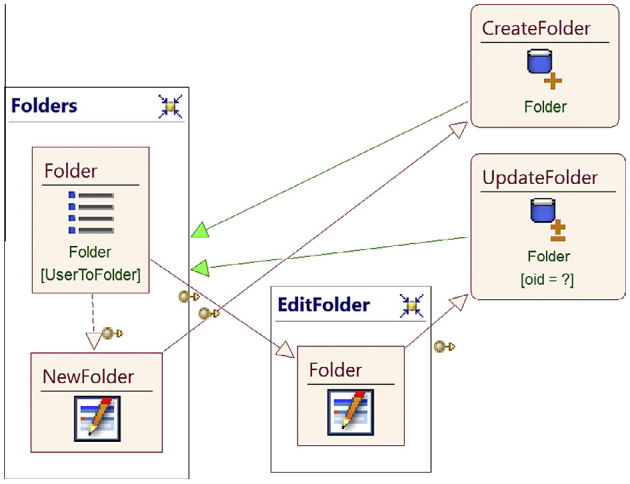


Fig. 9. WebML model generated from the tagged mockups of Fig. 7.

mockups have been completely tagged, a demo of the running application can be run to end-users or to the Product Owner to show how the application is behaving with the modeling done so far. Thus, instead of waiting for the end of the Sprint, stakeholders can see a working prototype to assess whether the application is behaving as expected.

4.7. Conclusions

The MockupDD approach, introduced throughout this section, proposes a lightweight modeling process of Web Applications. Using atomic tags and applying heuristics to infer new model elements (or refine existing ones), MockupDD pursues an agile-as-possible strategy. Modeling can be applied directly over the user interface mockups as soon as new requirements are discovered. Heuristics help to relate and refine all these model concepts (e.g., inferring data relationships or object transference between pages) in order to generate models as complete as possible in a semi-automatic fashion. To summarize, the main advantages and motivation of using the MockupDD in comparison to traditional MDWE are the following:

1. It removes the negative, static dependencies that frequently exist between models of different concerns – e.g. content, hypertext and presentation.
2. Requirements captured using mockups can be modeled directly over them, avoiding the necessity of intermediate requirements artifacts – such as Use Cases.
3. After capturing and modeling requirements on-the-fly, a demo of the modeled application can be run to test whether it was correctly specified.

These advantages help shortening the MDWE development cycles, fulfilling important agile principles like *[...] satisfy the customer through early and continuous delivery of valuable software or Deliver working software frequently [...] with a preference to the shorter timescale.*

5. Tooling and implementation

5.1. Tooling support

Almost every step of the MockupDD methodology (depicted in Fig. 2) is assisted with specific tooling. Below we list different tools supporting the MockupDD methodology:

- *Mockup-to-HTML tool*: Introduced in [5], this tool enables to process mockup files built with custom tools like Balsamiq or Pencil, and generate both the underlying HTML representation and the SUI associated with it (*Mockup Processing* step in Fig. 2). Thus, it avoids writing the HTML from scratch from the mockups, making Step 1 sensibly quicker if tool-based mockups are used.
- *Interactive Tagging Tool*: On the one hand, this tool is used for creating a SUI model from an HTML structure (*Relevant widget*

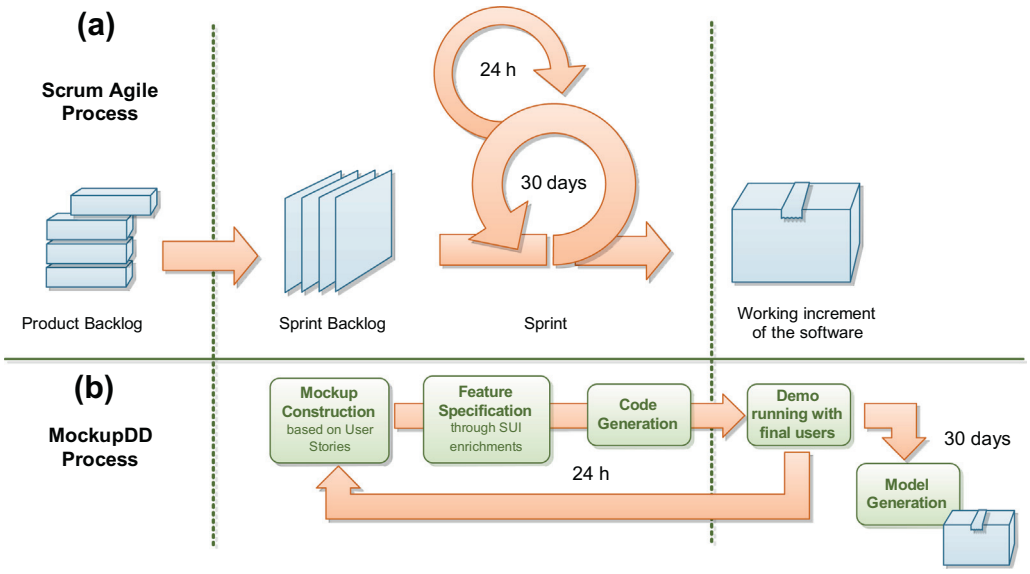


Fig. 10. Workflow of the Scrum process vs. the MockupDD one.

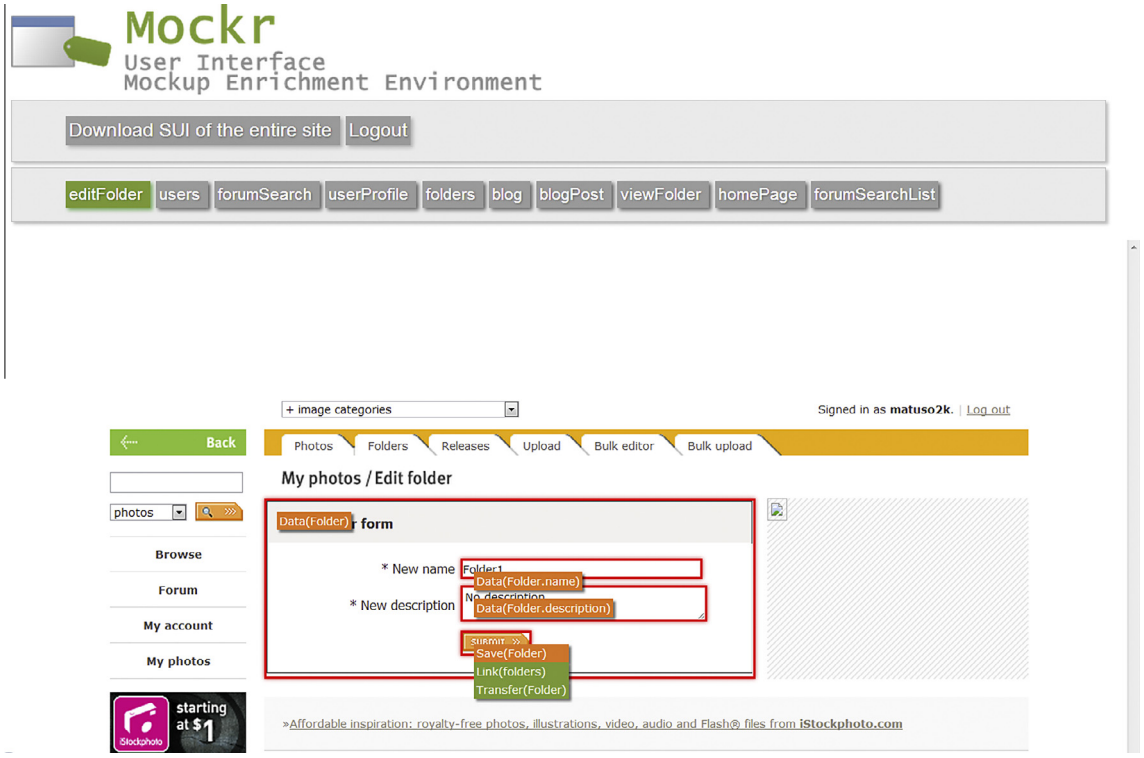


Fig. 11. Screenshot of the Interactive Tagging Tool.

detection step in Fig. 2). As we show in Fig. 11, the tool allows importing the HTML source and interactively marking the elements that will form the SUI model in an iterative and interactive way. After the HTML and SUI model are completed and correctly mapped, this tool also enables applying behavior enrichments (*Feature Specification* step in Fig. 2) with tags, using the HTML as a modeling frontend. For this purpose, the tool can work in two different modes: *widget discovering* (to build the SUI) and *widget tagging* (to apply tags over already mapped widgets).

- *Demo Sandbox Environment*: Once the enrichments are enough to satisfy the requirements tackled on a given iteration, this tool enables to quickly generate a version that can be tested by end-users. This tool does not depend on the underlying MDWE methodology; neither does it require any application deployment to emulate the final running application, using a *sandbox* to enrich the static HTML files with the behavior expressed by the enrichments.

The Interactive Tagging Tool and the Demo Sandbox Environment perform the most important tasks. The Interactive Tagging Tool also provides the tag disambiguation and refinements suggestions and can trigger the application demo running or the MDWE model generation as well.

5.2. Implementation aspects

SUI models must be built and mapped over the original elements in the mockup source. From a technical point of view, if HTML mockups are used, this mapping is preserved with an XPath expression obtained from the source element and attached to the corresponding SUI object. If a supported mockup tool is used instead (like Balsamiq or Pencil), an *id* is computed from the mockup source code and associated to the SUI object; this *id* is generated in a way that can unequivocally identify the mapped element – for instance, in the Balsamiq case is obtained concatenating the mockup name and the element's unique name.

The Interactive Tagging Tool consists of a Web Application that enriches the *static* HTML mockup with the behavior required to highlight and mark HTML (DOM) elements. It provides an API on the server-side that allows creating new SUI elements to be mapped over the DOM, or to apply tags over already mapped elements. Since the user explicitly relates concrete HTML elements to SUI model concepts, the mapping between them can be easily done by linking the HTML's XPath to its corresponding SUI element.

When working with mockups built using supported mockup tools, the use of the *Mockup Processing Engine* (MPE) automatically generates an instance of the SUI. The MPE is embedded into the Mockup-to-HTML tool and is composed by a set of chained individual processing components; the most important processing components that are traversed starting form a plain mockup source file input and finishing with a full-fledged SUI model are the following:

1. *Widget Parser*: This processor takes the mockup source files as input and identifies all the individual widgets in it. The output is an unsorted but uniform list of widgets, grouped by mockup. Since the mockup source format is specific for each tool, there is a different parser for every tool supported.
2. *Containment Detector*: This processor takes the unsorted widget list from the *Widget Parser* and discovers the containment relationships between them. Since most digital mockup tools represent mockups as a simple list of widgets with specific coordinates, width and height, this processor allows discovering the mockup's hierarchical structure.
3. *Layout Inferer*: This processor takes the hierarchized widget list from the *Containment Detector* and analyzes the internal graphical distribution of every `CompositeWidget` to infer a specific layout type and the widgets distribution according to it. The layout inference processing is done by applying heuristics over the widgets' graphical disposition in two steps: (1) analyzing the widgets distribution to establish the optimal layout and (2) determining the disposition of each widget according to the chosen layout. While this processing is not relevant from the feature modeling approach discussed in this work – hence, not included in the SUI metamodel description – having a detailed layout configuration is mandatory for deriving a proper HTML visual structure. Extra details about layout modeling and inference can be found in [5].

Regarding the SUI metamodel representation (see Fig. 4), while UML Stereotypes can be used to introduce the concept of *tagging* `Widgets` in equivalent way, we decided to model tags and their application as separate classes in order to map exactly the implementation of the set of tools developed for MockupDD. At the same time, through this decision we are preserving the same expressive level than using stereotypes but omitting UML features that in practice are not directly supported by common Object-Oriented languages like Java or C#. In addition, the used metamodel representation maps closer different alternative representations of the SUI metamodel, as an XSD schema[7] already introduced to serialize SUI instances.

Regarding the tag processing and refining, an *Inferer* is used to analyze the ambiguous tags and propose potential solutions to the modelers. If more than one solution of improvement is found for a tag, then a popup menu is shown in *the Interactive Tagging Tool* over the tag to be improved showing all its variants. On the other hand, if only one solution or improvement is detected for a tag, it is done automatically.

Regarding the Demo Sandbox Environment, it presents the HTML mockups and executes a set of client-side enrichments per tag (according to their semantics) to let the end-users interact with a prototype of the final application. The Sandbox emulates some aspects of the application (for instance, client–server communication) in order to avoid the time-consuming building and deployment stages, thus quickly providing testable versions of the application on-the-fly with the tags introduced so far.

6. Evaluation

In order to evaluate the MockupDD approach and its tool support we carried out an experiment. We used the Goal–Question–Metric (GQM) method [26] to drive our evaluation, which represents a systematic approach for tailoring and integrating goals to models of the software processes, products and quality perspectives of interest, based upon the specific needs of the project and the organization. GQM defines a certain goal, refines this goal into questions, and specifies metrics that should provide the information to answers these questions. Next, we describe information related to our evaluation, from planning to interpretation stage.

6.1. Planning stage: goal

The GQM model starts top-down with the definition of an explicit measurement goal. The primary purpose of the evaluation was to make a quantitative comparison of the MockupDD method vs. the traditional *pure* MDWE modeling process to evaluate the improvements that the approach can provide in terms of effectiveness and efficiency. Table 1 shows the main goal of our evaluation.

Achieving these measurements goals would yield a better understanding of effectiveness and efficiency of the MockupDD approach.

6.2. Definition stage: questions and metrics

The experiment does not include an evaluation of the advantages of using mockups through the development process, since this has been already assessed in detail by Ricca et al. [6].

The following research questions were considered in our evaluation:

- **Question 1**. Do MockupDD tags provide the semantics required to represent and implement data-intensive applications with effectiveness – i.e., in an accurate and complete way?
- **Question 2**. Does MockupDD generate MDWE models with efficiency, i.e. less effort and less error incidence, instead of building them from scratch using a MDWE language and its accompanying design method?

In order to provide evidence related to these questions, different metrics were selected, collected and interpreted. The metrics defined are based in WebML models we used for checking and

[7] MockupDD SUI XSD Schema – http://agilemdd.lifia.info.unlp.edu.ar/mockupdd/sui.xsd, accessed 07-May-2013.

Table 1
Main goal of our experiment.

Analyze the:	Delivered web product and development process
For the purpose of:	Understanding the MockupDD approach
With respect to:	Effectiveness and efficiency of the MockupDD approach
From the viewpoint of:	The project team
In the context of:	Web development environment with simulated clients and users (a Photo Stock website application was developed)

comparing whether the MockupDD approach has the same expressive level and allows generating the same kind of models, but with more effectiveness and efficiency. The experiment participants worked with WebML and MockupDD in order to gather quantitative information of both approaches. The metrics defined were the following:

- **Metric 1. Coverage Ratio (CR)**. Percentage of WebML model elements that can be generated from tags to specify the desired semantics. It provides a measure of modeling effectiveness.
- **Metric 2. Time Spent on Correcting Errors (TSCE)**. Errors are particular situations where experiment participants did not complete the modeling successfully and a new specification must be considered. TSCE metric is computed by analyzing MDWE models generated through a MockupDD tagging against models built manually and checking failures (such as missing elements and semantic errors). Both approaches (MockupDD vs. MDWE) are compared to define which one performed better and with fewer errors. This provides a measure of modeling efficiency.
- **Metric 3. Completion Rate/Mean Time-on-task (CRMT)**. This metric summarizes the relationship between the time taken for modeling the same application in both MockupDD and *manual* MDWE. This gives us a better understanding on how much faster developers can build MDWE models with MockupDD in comparison to manual modeling. Like TSCE, it provides a measure of modeling efficiency.

6.3. Data collection stage: method

6.3.1. Participants

In order to collect the statistical information for the evaluation, we organized 10 academic development groups, each one composed of three modelers; five groups of participants worked using MockupDD (Experiment groups) and the remaining five groups used WebML (Control groups) with its supporting tool, WebRatio.[8] The participants were advanced undergraduate students of Computer Science and were familiar with Web Engineering methods and WebML. Both groups worked in different laboratory sessions and were able to work practically day-by-day with potential users and clients within both processes. Clients and users were simulated in the evaluation process by assistant professors of Computer Science from University of La Plata. In this context, they used natural language, User Stories and mockups to refine requirements specifications. They also provided technical recommendations or suggestions by using only User Stories; that is, expressed requirements derived from interaction patterns [27].

While clients were available the same amount of time for both groups (Control and Experiment), we noted that participants belonging to the Control groups interacted in a more *spaced* manner in comparison to participants of the Experiment groups. We empirically attribute this to the use of the traditional, waterfall-like MDWE workflow.

6.3.2. Apparatus

MockupDD and WebML approaches were considered in this test. We chose the WebRatio solution because it is a mature, fully model-driven setting and it has been used in several industrial-scale projects.[9] In general terms, our validation approach consisted in using MockupDD to generate WebML models and then, check whether MockupDD improves the modeling process instead of building WebML models directly.

Experiment groups applied the previously commented MockupDD Scrum adaptation, dividing the whole development in three Sprints of 1 week of duration each one. In each Sprint, a subset of the presented mockups were refined and tagged. On the other hand, Control groups worked in a more traditional MDWE way, without a structured process behind.

6.3.3. Procedure

The validation was divided in two different stages. In both stages, the participants were asked to build models to implement the previously introduced Photo Stock website application. In both cases, a set of detailed mockups and User Stories of the application to be modeled was given to all the participants. The set of mockups and User Stories was the same for all the participants within the experiment.

The first stage of the validation, aimed at answering Question 1, involved the manual building of WebML models and checking if MockupDD was able to express the same concepts that were present in them, thus assessing its semantic feasibility. We gave the set of mockups and Stories describing the application to every participant of Control groups and asked them to build a WebML model that implements the functionality described.

On the other hand, all subjects of Experiment groups received an initial MockupDD training session in which an instructor explained the approach (prototyping, tagging, and related tools).

The summarized tasks selected for the evaluation were:

- To generate unique credentials to access to the MockupDD tooling for every participant.
- To instruct participants of the Experiment groups in the MockupDD process (introducing tool usage and concepts such tags, SUI, etc.)
- To ask all participants to develop a simple data-driven application (we chose the mentioned Photo Stock website) using the assigned methodology (WebML or MockupDD depending on the group).
- To ask all participants to keep track of the identified shortcomings, time spent and type of errors in modeling tasks in detail.

To increase the motivation on the subjects, the instructor explained to them that the web development in the experiment would be similar to their final projects at the end of the term.

6.4. Interpretation stage: results and discussion

The first objective of the experiment was to assess the effectiveness of MockupDD by calculating a value for Metric 1 (Coverage Ratio, CR). In Fig. 12, a graphical summary of the CR in one of that WebML models can be observed. In that figure, every Data Unit and Link (WebML concepts) is graphically covered by its originating tag (i.e., the tag that causes its creation during the model generation

[8] WebRatio – http://webratio.com/, accessed 07-May-2013.

[9] WebRatio Success Stories – http://www.webratio.com/portal/content/en/success-stories, accessed 07-May-2013.

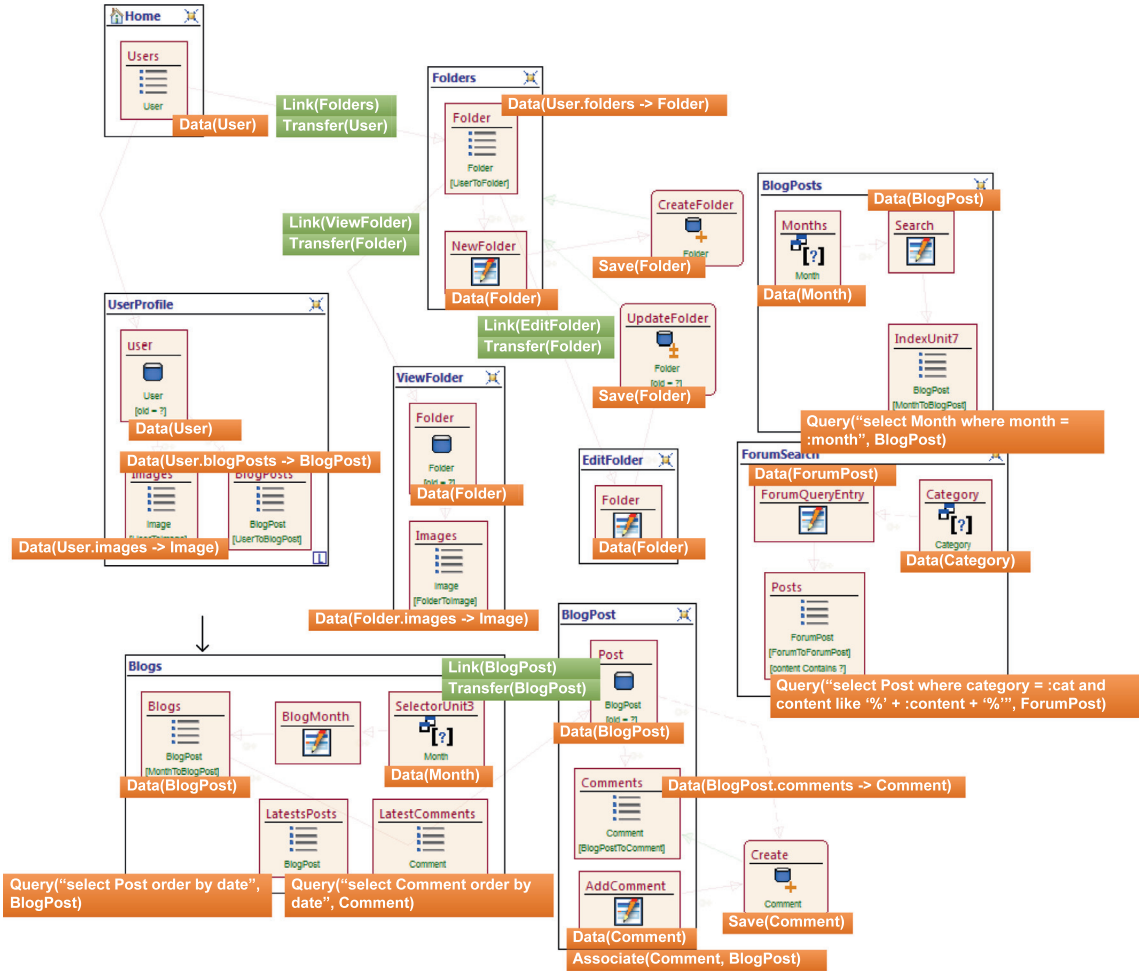


Fig. 12. Tag coverage of a WebML model.

process). As the figure shows, tags cover almost all of the elements. In addition, in some case one tag can generate more than one WebML element; for instance, a `Data()` tag including data navigation generates both the underlying Data Unit and its internal selector in order to navigate to the related instance. Also, some elements like Entry Units are generated from underlying mockup elements (e.g., input forms) and since MockupDD does not use connections to express simple business logic, CRUD-related Units and their links (usually between 2 and 3 in average) are generated automatically from only one tag.

The results of the detailed analysis showed that the only feature/model elements that cannot be automatically generated were the Selectors of the Units where the `Query()` tag was used. For every WebML model built by a participant, the CR was calculated and then averaged to obtain a single value for Metric 1. The results showed that the average CR for WebML models was between 5.44% and 14.14% with a confidence of 98%. In addition, as a result of the analysis, we concluded that the only elements that cannot be covered by MockupDD were WebML Query Units and Units with advanced functionalities which semantics are part of our future work – like AJAX behavior and improvements in data listing through `Data()` tags. Because of this result and the fact that executable queries can be freely expressed in MockupDD through a `Query()` tag (for instance in plain SQL language) but not exactly derived to WebML concepts, we can conclude that the MockupDD approach is effective in terms of the tagging coverage, thus answering positively to Question 1.

To study the efficiency criteria we established a set of error scale points that are listed in Table 2. The table assigns an error score to every error type according to the estimated time that have to be spent onto correct it. Thus, the score also represents the *semantic importance* of the error, i.e., how much it causes the model

Table 2
Estimated factor used to evaluate the time spent on correcting errors.

Error type	Criticality factor	Description
Extra Unit	1.25	The modeler used a Unit that can be omitted or grouped using a single, more semantic one, thus representing more modeling effort
Missing Unit	1.5	A Unit is missing, causing a missing feature in the application
More complex Unit	1.1	A simpler Unit (one that can be instantiated with less effort and configuration) can be used
Missing/wrong attribute in Unit	1.1	The modeler missed or provided an inadequate configuration attribute in a Unit
Missing link/transfer	1.1	The modeler missed a link or object transfer to another Page

to differ semantically from the same concept expressed in an ideal WebML model or how much extra effort it requires adapting it to its ideal representation. For wrong or missing element errors, the score assigned is based on the amount of time saved by omitting the instantiation of the Unit (*Missing Unit* error), something related to it (*Missing link/transfer* error) or a part of it (*Missing/wrong attribute in Unit*). The experiment organization team calculated and agreed these values proportionally considering the modeling time required in each case, starting from the least costly elements. The calculations were made observing the usual modeling behavior of the participants and by the personal experience of the team. Finally, while the introduction of additional WebML Units (*Extra Unit* error) does not imply that some functional requirement is not met in the application, it adds development time that is not directly valuable from the end-user point of view, thus delaying the implementation without providing any perceivable added value. In these cases (*Extra Unit* error), we decided to use a similar score that for the *Missing Unit* case but, since the end-user is getting a complete final application anyway, we only computed a half of the original value. Using this scoring approach, a perfectly matching model (comparing to an ideal WebML model) will have a error score of zero, since no time has to be spent correcting it. Any other model that differs from the ideal one will have a positive score, denoting the amount of proportional time required to correct this model or adding WebML components that do not meet an explicit end-user requirement.

We used the mentioned score to analyze the performance of both groups in terms of the quality of the WebML models obtained. To accomplish this task, we required an *ideal* WebML model that we could use as a reference to compare models obtained in both cases and analyze their quality. For this purpose, a *control WebML model* was carefully built step-by-step by all the members of the experiment organization team, checking that each concept used in it was the fastest-to-build amongst all the possible alternatives. For instance, when showing a list of data present in multiple interrelated entities, both a WebML Query and an Index can be used. However, the former takes longer to be built since it requires a full SQL-like query to be written while the latter only needs to specify the root entity and the relationships that have to be crossed to navigate to the interrelated classes. Since the experiment organization team members have more than 5 years of experience in WebML modeling and even more in the MDWE field, we assumed that the obtained model is the one that can be built the quickest to meet the requirements of the Photo Stock website. Moreover, since mockups restrict the type and structure of the Pages in the WebML model, practical experience demonstrated that most of the design doubts when building such model were related to choosing which specific type of Unit to use (e.g. Query vs. Index, Multidata Unit vs. Index, etc.).

Using the control WebML model, we computed Metric 2 (Time Spent on Correcting Errors, TSCE) for the manually built WebML models and for models generated from the tagged mockups. We used Cliff's Delta [28] metric to measure the effect size of applying MockupDD in the development process. We chose that measure because it allows computing the amount of difference between two groups in a non-parametric way – in this case, without referring to concrete time spent in modeling. The result of the Cliff's Delta computation returned a value of -0.07 which implies a slight improvement regarding time in correcting errors in favor of MockupDD. In addition to this metric, we averaged the TSCE per page/mockup to analyze where MockupDD performed better than WebML and vice versa. The partial results are depicted in Fig. 13a. In this figure we also show box plots with the error score distribution for WebML (Fig. 13b) and MockupDD (Fig. 13c). Because of this, we can conclude that the MockupDD approach did not introduce new errors in the modeling in comparison to WebML, and also showed a slight efficiency improvement in terms of defining correct models from scratch. In addition, we observed that participants who built models directly with WebML were more prone to commit errors with a higher score, while those that used MockupDD tended to commit less valuable errors.

For Metric 3 (Completion Rate/Mean Time-on-task, CRMT), we assessed the time spent by all the individuals during the modeling tasks; that is, initial instruction and tagging in the MockupDD case and direct WebML model building in the context of WebML. When participants worked, we asked them to carefully take account of

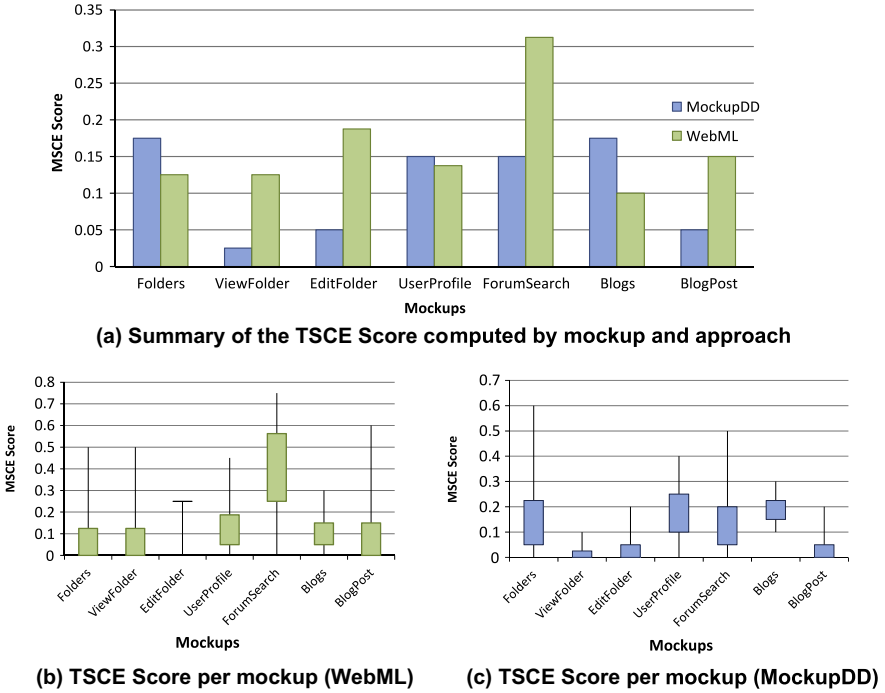


Fig. 13. WebML vs. MockupDD average error score and error difference per mockup.

the time spent in the modeling as a key issue in their work. As in the previous case, we computed the Cliff's Delta to measure the effect size of applying MockupDD in comparison to WebML, this time in relationship to the time required to obtain models. The results of the Cliff's Delta computation were −0.89, which implies that MockupDD time measures were significantly less in average than the ones taken in the WebML case. We also included a graphical comparison of the averaged time spent in every step in Fig. 14.

Having positive results for Metric 2 (TSCE) and Metric 3 (CRMT), we can conclude that Question 2 can be also answered positively. Having answered positively to the defined questions through the detailed metrics we can conclude that in general terms the MockupDD approach offers a faster and less error-prone model building strategy in comparison to classic MDWE workflow.

6.5. Threats to validity and limitations

We comment here on some aspects that can represent threats to the results obtained, and were taken into account in our work path. In addition, we describe how we intended to mitigate their impact on the results. The threats considered were:

1. **Small subset of test applications**. In our validation we tested MockupDD only with one common data-intensive application. Although data-intensive applications are characterized by similarities regarding the complexity of their structure and behavior, we are making experiments with different data-intensive applications to have a bigger statistical data set. On the other hand, we analyzed several classic and well-known examples of data-intensive websites (for instance, the ones bundled with WebRatio tool) and chose one similar and also less known for modelers, thus obtaining results of a typical and representative application in the field.
2. **Participants' industrial experience**. While the participants in the experiment had a clear and deep understanding of Web development and modeling, they had poor industrial experience in general. Including participants with industrial experience can lead to a more mature understanding of the real benefits of MockupDD vs. traditional Web modeling. While it is not fully comparable with industrial experience, to mitigate the impact of this threat, we selected students that participated in the development of at least two Web Applications in the academy already.
3. **Domain familiarity**. Several data-intensive applications examples are widespread and well known (e.g., E-Commerce sites). Experiment participants can have an initial knowledge of the domain and business logic that can influence the application modeling under both approaches compared in our validation. In order to avoid that issue, we asked questions about the Photo Stock application domain to check whether it was completely new for all the participants involved in the experiment.
4. **Errors in the control WebML model**. The WebML control model is a key artifact to compute some of the metrics in the evaluation. If a simpler control model can be defined, then the error score computed for the models built by hand vs. those generated by MockupDD in comparison to the WebML control model can result unfavorable for MockupDD. As we explained before, to reduce this potential threat, the control WebML model was carefully built by a team of professionals with at least 5 years of experience in the WebML language and even more in the MDWE field.
5. **Error score**. The error score computed to compare WebML models built by participants to the control WebML model was defined and agreed by the experiment organization team according to the estimated proportional time required to deal with every error (e.g., adding new WebML concepts, changing their attributes, etc.). However, errors in this score may lead to failures in model evaluation, for instance, considering models built by hand more costly than those generated through the MockupDD tooling. To mitigate this threat, the experiment team verified that correcting errors made in the models built by hand would take longer to be corrected than those generated using MockupDD after the execution of the experiment sessions.

6.6. Lessons learned

In our experiment we quantified and validated the benefits of MockupDD regarding Web Application modeling and generation. However, we have not mentioned yet how the participants behaved in the context of this agile and MDD combined process. In this subsection we will comment different situations observed in practice during the MockupDD experiment modeling sessions. We compare them with modeling behaviors witnessed during the traditional MDWE working sessions in the same context. The main motivation for this section is to capture the part of the experience that could not be specifically quantified in relation to agile features and practices observed in the process – something that we commented in the previous section.

1. **Learn the language through modeling**. While participants learned and practiced WebML modeling with WebRatio through specific courses that took from 2 to 3 months long, MockupDD was introduced with a small instruction session lasting less than an hour. Then, modelers learned the remaining aspects of MockupDD during the modeling, using the user interface metaphors present in mockups to understand the semantics and rules of the language constructs (tags). Thus, we assessed that MockupDD was easier to learn *on-the-fly* while modeling.
2. **Frequency of modeler/developer feedback**. In MockupDD, generating a running application was at the click of a button. As a result, running prototypes were generated more often when using it than in the WebRatio case. This implied a less spaced interaction between end-users and modelers in the MockupDD case. It is important to note that WebRatio already has a *generate* feature that enables to derive, build, deploy and then run the application being modeled. In this case, running prototypes were also used but, since starting the updated application required more effort (e.g. restarting a web server,

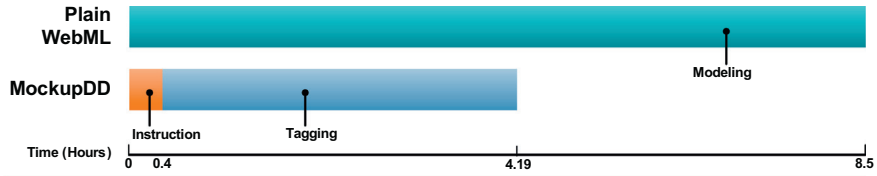


Fig. 14. Experiment time distribution.

updating database structure, etc.), it was less frequent. In many cases, the members of the experiment organization team were able to correct misinterpretations in requirements gathered as soon as noticed in the running demo. Thus, the experience showed that instead of discussing requirements orally or textually, the application itself was used more often to discuss the current development version in comparison to the WebML case. At the same time, the fluent interaction clearly showed who the more active developers were in every group in the MockupDD case.

3. **Explicit doubts from developers**. We noted that direct questions made to end-users when modeling over the visual metaphors in mockups (MockupDD) were more frequent than when modeling just observing them (WebML/WebRatio).
4. **Modeling the big picture**. We noted that MockupDD modelers tended to model in a more *iterative* or *shallow* way, i.e. in rounds or stages in which they enrich every mockup partially with some features. On the other hand, WebML modelers were more prone to model pages *in depth*, building every page completely before moving onto the next one.

From the observed summary, points (2) and (3) helped us to empirically assess that MockupDD is closer to satisfy some of the agile principles in comparison to classic MDWE. In general, we found that some of the principles in the Agile Manifesto[10] were easier to accomplish when using MockupDD because of the advantages provided by the more frequent interaction between developers and end-users. Since we evaluated only one application in detail and used (through automatic model generation) only one technological stack, it was not possible to assess agile features related to team work practice visible in long periods of time like *Continuous attention to technical excellence and good design enhances agility*, *The best architectures, requirements, and designs emerge from self-organizing teams* or *at regular intervals, the team reflects on how to become more effective [...]*.

Because of these initial behaviors observed in the MockupDD case, in the future we plan to take metrics related to modeling practice in situ to measure the practices noted with more detail and more accurately.

7. Related work

In the model-driven development field, prototype or structural user interface models have been linked to other metamodels within different approaches. All modern MDWE methodologies consider detailed user interface definition in the last stages of their lineal processes: WebML provides templates to render its units, UWE specifies a custom presentation UML profile for the same purpose and OOHDM uses *Abstract Data Views*. In addition, there have been defined MDD processes non-strictly related to the Web domain in which presentation specifications have been tackled as a final step, sometimes using mockups. An example of such process is detailed in [29], in which business and data artifact models represent the starting requirements specifications, enabling to generate user interface mockups from them afterwards. MockupDD differs from all these approaches promoting the use of UI specifications from the first stages of the modeling process, thus using them as initial, detailed requirement artifacts to discuss and assess how the application should look and behave in a language that is understandable by end-users.

Several model-driven approaches started to consider using UI specifications in the initial steps of their processes. In the context of the OO-Method, Panach et al. propose gathering interaction requirements from UI sketches [30], creating structural task models from them, using the ConcurTaskTrees [31] formalism. Efforts were also made to integrate UI specifications into requirement languages, such as RSL (Requirements Specification Language) [9]. In this work, a general UI model was used to specify the kind of domain objects the User Interface should manage and how they are manipulated using a non-linear storyboard metamodel – i.e. a metamodel that expresses the type of interaction done within the UI but not the specific order in which the steps are executed, as in [30]. Instead of focusing on the generation of task models like in [30], the methodology proposed here is oriented to generate runnable data-intensive Web Applications as WebML does. WebSpec [32], a visual domain-specific language designed to capture interaction requirements, uses formal descriptions like preconditions and logical assertions to specify how the user interface must behave during every User Story. Thus, it is more oriented to detailed interaction requirements (i.e. how the application should behave) instead of defining the kind of domain objects that should be manipulated and how. MockupDD is oriented and restricted to these last features in order to generate a running application that can be tested by end-users as quick as possible.

Different forms of user interface specifications were used in the context of agile and model-driven approaches. On one hand, mockups have been successfully used in agile processes; in [11], different experiences of using UI mockups in the context of agile processes are commented. The results of the survey conducted in this work showed that mockups are essential to assure usability and also represent a valuable addition to User Stories to describe requirements. In this context, they also help to estimate the effective development cost of each Story [12]. The same conclusions were drawn in [33], where developers admit giving more relevance to mockups than to User Stories associated to them. However, in common agile settings mockups are used only as a requirement artifact and developers have to code all the application by hand just observing them. In this work we have shown how digital mockups can be reused throughout the development process in the context of a user-centered MDWE development methodology.

Recently, the idea of starting the development of web applications from HTML content and then wiring the different aspects of it through languages (like CSS already does for presentation) was proposed [34]. In comparison to MockupDD, this approach is more related on the technical aspects of building Web Applications using HTML mockups than using mockups themselves as a requirement artifact. However, it shares with MockupDD the focus on tackling the application behavior first in the development process.

Another common practice is to link mockups to more traditional software artifacts like UML Use Cases [35], where the authors obtain mixed models and use them to derive prototypes that even become updated whenever requirements change, in a round-trip way. In this way, Kulak et al. have also used UI requirement tools combined with Use Cases [36]. In addition, Cohn [10] proposes incorporating UI mockups with User Stories in an Agile Development context. Annotating *informal* mockups is another approach frequently used in the context of requirements gathering [16,37]. In [38], mockups with different fidelity levels are linked to users, tasks and interaction models. This approach also proposes to evaluate UI mockups using annotations. The idea of discovering content models from form-based user interface prototypes has also been discussed [39]. However, all the approaches commented previously are developer-oriented and focused on requirements refining or generating only a part of an application – for instance, data models. MockupDD proposes to generate the whole final application (as in common MDWE approaches) placing annotations over mockups that are technically valuable for developers and also simply enough for end users to understand.

[10] Principles Behind the Agile Manifesto – http://agilemanifesto.org/principles.html, last accessed 23-Aug-2013.

Several Model-Driven approaches for defining and generating User Interfaces exist. One of the most remarkable approaches in the field is UsiXML [24]. The UsiXML approach supports the definition of User Interfaces independently of their device, platform and modality, defining 4 different levels of abstraction from tasks models to Concrete User Interface (CUI) specifications. In this work, we use a unique UI language and abstraction level (the SUI) to represent UIs and we use it as a foundation to define different concerns and aspects. Our approach is closer to the idea of generating UI prototypes from high-level languages that have also been discussed in [40]. In this work, the authors present a domain specific language (DSL) to specify the content of user interfaces that are structured following a global menu pattern. Concrete implementations of UI mockups are generated and the approach follows an iterative process between developers and customers until the final prototype application is obtained. While MockupDD is also oriented to generate the Web Application too, instead of starting with an abstract or general view of the it (like a global menu), it starts the development process with mockups, which represent the concrete and expected view of what the end-users require.

Within the existing methods for software design, User-Centered Design (UCD) [41] is intended to incorporate user's perspective into the software development process in order to achieve a usable system. A key principle of UCD is, among others, the prototyping activities [42]. The prototyping approach has been promoted as a primary solution to manage and even spur design changes during system development. Prototyping methods move from low to medium fidelity [43]: paper sketches, storyboards, Pictive, scripted simulations and so on, each getting slightly more sophisticated. In this context, MockupDD proposes to use graphical mid-fidelity prototypes to start and guide the modeling of Web Applications, as we already described.

Regarding our previous work, we already introduced the idea of generating MDWE models from user interface mockups both for the UWE [2] and for WebML [1,5] approaches. Using both approaches as a basis, here we have outlined a unified process to iteratively discover features and build models for MDWE methodologies in general through an UI prototype enrichment strategy that can be used in the context of agile processes. While in [5] we briefly introduce the presented approach, in this paper we describe it in detail,

1. introducing a more exhaustive set of high-level web specifications,
2. showing its general applicability in the MDWE field using both UWE and WebML methodologies,
3. detailing how it can facilitate the introduction of agile features in MDWE and also how can be directly used within well-known agile processes,
4. introducing a *Demo Sandbox Environment* that enables end-users to test how the application will behave in production in any moment during the modeling and
5. including a validation experiment that showed the quantitative and qualitative improvements that the approach can provide in the context of the development of a concrete Web Application.

8. Conclusions and future work

In this paper we presented MockupDD, a model-driven approach that combines traditional model-driven practices with some features of agile software development approaches. Since a great deal of work has already been done in the MDWE area, we have oriented our approach not as *yet another model driven approach*, but as an *agilizer* of existing MDWE proposals, and also as a requirements gathering approach oriented to data-intensive Web Applications.

We have shown through an evaluation that, in comparison to traditional MDWE approaches, MockupDD reduces the errors and effort in model building, reusing the already defined MDWE architecture and tooling. In addition to the concrete validation results, we observed that MockupDD allows introducing agile features to the modeling process thanks to: (1) the use or mockups that improves the requirements gathering in comparison with textual methods [6] and also facilitates communication between customers and developers, and (2) the atomic modeling strategy provided by tags and the demo generation features to show the application modeled so-far to end-users at any particular moment during the development process.

Despite the fact that part of the process involves features discovering from user interfaces, MockupDD is mostly a forward approach rather than a reverse engineering one. Except for the heuristics used in tagging assistance and automatic model importing or generation, the rest of the stages involve designer skills and irreplaceable human intervention. User interface mockups are used as a technique to discuss and formally capture requirements that can be expressed in modern MDWE methodologies.

We have shown that mockups are completely reused in the approach in the following way:

1. If detailed HTML mockups representing the definitive presentation for the application are built, they are completely reused, importing them into the Interactive Tagging Tool.
2. If digital tool-based mockups are used, they can be derived to an HTML representation and then imported into the tool. In this case, the automatically generated frontend can be refined until obtaining the final presentation structure.

Regarding the current and future work in the approach, we plan to continue testing and validating it through the development of applications like the one commented in this paper. In addition, we are continuously analyzing and improving the proposed tag sets in the context of real Web Application developments in order to improve them by making them clearer, more consistent and complete. The addition of several features through pluggable tag sets like RIA behavior, social, geographical and security-oriented are also being studied.

Supporting many MDWE approaches is also an important task we are taking into account, since it assures that the methodology is applicable to the model-driven Web Engineering field in general, and not only over a subset of methodologies. Furthermore, we are building a catalogue of well-known user interface patterns using the current tag set in order to prove its applicability and generality.

Outside the MDWE field, mockup tagging can be used to generate different types of specifications that are not directly related to UI concepts. We are investigating different features that can be specified using the graphical metaphors expressed in mockups that are related to non-visible behaviors like transactionality, APIs composition or concurrency. When mockups are used to elicit interaction requirements of an application, this kind of tags can aid to discover and specify (and in some cases generate automatically) parts of applications that are not necessary visible to customers or end-users.

Related to this work, the WebSpec approach [32] allows describing how user interfaces must behave using a detailed language. We are also working in a MockupDD-to-WebSpec translator that eases the task of defining common WebSpec constructions using tags that are easy to apply. Finally, we are extending our tool to provide code generation capabilities for common Web technologies and architectures.

Acknowledgments

This project is partially supported by the PROALAR DAAD – MINCYT project DA/11/11.

References

[1] S. Ceri, P. Fraternali, M. Matera, Conceptual modeling of data-intensive Web applications, IEEE Internet Comput. 6 (2002).
[2] N. Koch, A. Knapp, G. Zhang, H. Baumeister, UML-Based Web Engineering, Springer, London, 2008.
[3] G. Rossi, O. Pastor, D. Schwabe, L. Olsina, Modeling and implementing web applications using OOHDM, in: G. Rossi, O. Pastor, D. Schwabe, L. Olsina (Eds.), Web Eng. Model. Implement. Web Appl., Springer London, London, 2008, pp. 109–155.
[4] M. Wimmer, A. Schauerhuber, H. Kargl, On the integration of web modeling languages: preliminary results and future challenges, in: Proc. 3rd Int. Work. Model. Web Eng., 2007.
[5] J.M. Rivero, G. Rossi, J. Grigera, E.R. Luna, A. Navarro, From interface mockups to web application models, in: 12th Int. Conf. Web Inf. Syst. Eng., Sydney, New South Wales, Australia, 2011, pp. 257–264.
[6] F. Ricca, G. Scanniello, M. Torchiano, G. Reggio, E. Astesiano, On the effectiveness of screen mockups in requirements engineering, in: 2010 ACM-IEEE Int. Symp. Empir. Softw. Eng. Meas., ACM Press, New York, NY, USA, 2010.
[7] A. Ravid, D.M. Berry, A method for extracting and stating software requirements that a user interface prototype contains, Requir. Eng. 5 (2000) 225–241.
[8] L. Cao, B. Ramesh, Agile requirements engineering practices: an empirical study, IEEE Softw. 25 (2008) 60–67.
[9] K.S. Mukasa, H. Kaindl, An integration of requirements and user interface specifications, in: 6th IEEE Int. Requir. Eng. Conf., IEEE Computer Society, Barcelona, Catalunya, Spain, 2008, pp. 327–328.
[10] M. Cohn, User Stories Applied: For Agile Software Development, Addison-Wesley, 2004.
[11] J. Ferreira, J. Noble, R. Biddle, Agile development iterations and UI design, in: Agil. 2007 Conf., IEEE Computer Society, Washington, DC, 2007, pp. 50–58.
[12] A. Martin, R. Biddle, J. Noble, The XP customer role in practice. three studies, in: Agil. Dev. Conf., IEEE Computer Society, Salt Lake City, Utah, USA, 2004, pp. 42–54.
[13] J. Sutherland, K. Schwaber, The Scrum Papers: Nuts, Bolts, and Origins of an Agile Process (n.d.).
[14] S. Ceri, P. Fraternali, A. Bongio, Web Modeling Language (WebML): a modeling language for designing Web sites, Comput. Networks. 33 (2000) 137–157.
[15] A. Rashid, D. Meder, J. Wiesenberger, A. Behm, Visual requirement specification in end-user participation, in: First Int. Work. Multimed. Requir. Eng., IEEE Computer Society, Minnneapolis, MN, USA, 2006, pp. 6–6.
[16] K. Schneider, Generating fast feedback in requirements elicitation, in: Proc. 13th Int. Work. Conf. Requir. Eng. Found. Softw. Qual., 2007, pp. 160–174.
[17] P.P. da Silva, User interface declarative models and development environments: a survey, Lect. Notes Comput. Sci. 2001 (1946) 207–226.
[18] J. Guerrero-Garcia, J.M. Gonzalez-Calleros, J. Vanderdonckt, J. Munoz-Arteaga, A theoretical survey of user interface description languages: preliminary results, in: Proc. 2009 Lat. Am. Web Congr., IEEE, 2009, pp. 36–43.
[19] J. Lin, M.W. Newman, J.I. Hong, J.A. Landay, DENIM: finding a tighter fit between tools and practice for Web site design, 2000, pp. 510–517.
[20] M. McCurdy, C. Connors, G. Pyrzak, B. Kanefsky, A. Vera, Breaking the fidelity barrier, in: SIGCHI Conf. Hum. Factors Comput. Syst. – CHI '06, ACM Press, New York, NY, USA, 2006, p. 1233.
[21] D. Engelberg, A. Seffa, A Framework for rapid mid-fidelity prototyping of web sites, in: IFIP 17th World Comput. Congr. – TC13 Stream Usability Gaining a Compet. Edge, Montreal, Quebec, Canada, 2002, pp. 203–215.
[22] J.M. Rivero, G. Rossi, J. Grigera, J. Burella, E. Robles Luna, S. Gordillo, From mockups to user interface models: an extensible model driven approach, in: ICWE'10 Proc. 10th Int. Conf. Curr. Trends Web Eng., Springer-Verlag, Berlin, Heidelberg, 2010, pp. 13–24.
[23] J.M. Rivero, J. Grigera, G. Rossi, E.R. Luna, N. Koch, Towards agile model-driven web engineering, Lect. Notes Bus. Inf. Process. 107 (2012) 142–155.
[24] Q. Limbourg, J. Vanderdonckt, B. Michotte, L. Bouillon, V. López-Jaquero, USIXML: A language supporting multi-path development of user interfaces, in: R. Bastide, P. Palanque, J. Roth (Eds.), Eng. Hum. Comput. Interact. Interact. Syst. Jt. Work. Conf. EHCI-DSVIS 2004, Springer, Berlin Heidelberg, Hamburg, Germany, 2005, pp. 200–220.
[25] S.W. Ambler, Agile model driven development is good enough, IEEE Softw. 20 (2003) 71–73.
[26] V. Basili, G. Caldiera, D. Rombach, The Goal Question Metric approach, 1994.
[27] M. van Welie, G. van der Veer, Pattern languages in interaction design: structure and organization, in: Proc. Interact '03, 2003, pp. 527–534.
[28] G. Macbeth, E. Razumiejczyk, R.D. Ledesma, Cliff's delta calculator: a non-parametric effect size program for two groups of observations, Univ. Psychol. 10 (2011) 545–555.
[29] N. Sukaviriya, V. Sinha, T. Ramachandra, S. Mani, Model-driven approach for managing human interface design life cycle, in: Proc. 10th Int. Conf. Model Driven Eng. Lang. Syst. Model. 2007, Nashville, USA, 2007, pp. 226–240.
[30] J.I. Panach, S. España, I. Pederiva, O. Pastor, Capturing interaction requirements in a model transformation technology based on MDA, J. UCS. 14 (2008) 1480–1495.
[31] F. Paternò, ConcurTaskTrees: an engineered notation for task models, in: D. Diaper, N. Stanton (Eds.), Handb. Task Anal. Human-Computer Interact, Lawrence Erlbaum Associates, Manwah, 2003, pp. 483–503.
[32] E. Robles Luna, G. Rossi, I. Garrigós, WebSpec: a visual language for specifying interaction and navigation requirements in web applications, Requir. Eng. 16 (2011) 297–321.
[33] H. Ton, A strategy for balancing business value and story size, in: Agil. 2007 Conf., IEEE Computer Society, Washington, DC, USA, 2007, pp. 279–284.
[34] E. Benson, Mockup driven web development, in: Proc. 22nd Int. Conf. World Wide Web Companion, International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, Switzerland, Rio de Janeiro, Brazil, 2013, pp. 337–341.
[35] A. Homrighausen, H.-W. Six, M. Winter, Round-trip prototyping based on integrated functional and user interface requirements specifications, Requir. Eng. 7 (2002) 34–45.
[36] D. Kulak, E. Guiney, Use Cases: Requirements in Context, Addison-Wesley, 2004.
[37] J.M. Moore, Communicating requirements using end-user GUI constructions with argumentation, in: 18th IEEE Int. Conf. Autom. Softw. Eng., IEEE Computer Society, Montreal, Quebec, Canada, 2003, pp. 360–363.
[38] T. Memmel, H. Reiterer, Model-based and prototyping-driven user interface specification to support collaboration and creativity, J. Univers. Comput. Sci. 14 (2008) 3217–3235.
[39] R. Ramdoyal, A. Cleve, J.-L. Hainaut, Reverse engineering user interfaces for interactive database conceptual analysis, in: B. Pernici (Ed.), 22th Int. Conf. Adv. Inf. Syst. Eng., Springer, Berlin, Heidelberg, Hammamet, Tunisia, 2010, pp. 332–347.
[40] J. Zhang, K. Läufer, Z. Gong, Mockup-supported web requirements engineering, in: 2003 Int. Conf. Internet Comput., Las Vegas, Nevada, USA, 2003, pp. 684–687.
[41] Ergonomics of human-system interaction – Part 210: Human-centred design for interactive systems (n.d.).
[42] M. Maguire, Methods to support human-centred design, Int. J. Hum. Comput. Stud. 55 (2001) 587–634.
[43] T.Z. Warfel, Prototyping: a practitioner's guide, Rosenfeld Media (2009).