Capturing and Validating Personalization Requirements in Web Applications

Esteban Robles Luna
LIFIA, UNLP, Argentina
La Plata, Argentina
erobles@lifia.info.unlp.edu.ar

Irene Garrigós
Lucentia Research Group, DLSI,
University of Alicante, Spain
igarrigos@dlsi.ua.es

Gustavo Rossi
LIFIA, UNLP, Argentina
Also at CONICET
La Plata, Argentina
gustavo@lifia.info.unlp.edu.ar

***Abstract* — Personalization is a key feature to improve user experience in Web applications and therefore many Web engineering approaches allow the specification of some type of personalization when modelling a website. However, these approaches usually neglect the process of capturing and representing personalization requirements, thus not considering them when the application evolves; maintenance of these requirements is then a very complex task. In this paper, we present WebSpec, a requirement artefact used to capture navigation, interaction and interface aspects of Web applications. Concretely, we focus on how to specify personalization requirements, and on how to automatically generate the personalization model from their specification. Furthermore, from the requirements specification we derive a set of interaction tests to assess the personalization functionality. We illustrate our ideas with an E-commerce application example and describe a prototype tool which implements the described functionality.**

Keywords: Personalization, Web requirements, Requirements Validation

I. INTRODUCTION

The World Wide Web has changed the way we communicate and exchange information. Web applications have become more complex and the information they provide is continuously growing. Web engineering approaches [2], [4], [7], [9], [12], [20] appeared to provide a systematic way to develop complex Web applications. In this area, personalization [11] has been proposed as a solution to improve the user experience by analyzing his context, characteristics and browsing history and changing different aspects of the application according to his needs.

Due to the different needs and goals of the large and heterogeneous audience that a Web application serves, user expectations need to be considered from the beginning of software projects. However as indicated in [5], most Web engineering approaches do not seriously consider the requirements analysis phase, and as a consequence these requirements are barely taken into account when the application evolves. Therefore, the resulting Web applications usually have outdated requirements which makes impossible to test the actual customer's requirements, and there are difficulties to handle fast evolution, which is usually essential in the Web field.

Personalization is also a missing aspect in the requirement elicitation phase; there are few approaches that allow modelling personalization requirements (see Sect. VI for details). Moreover, usually (personalization) requirements are described informally, thus becoming a problem when we dive into the implementation and validation phases, particularly to assess if (personalization) requirements have been correctly implemented.

To tackle these problems we use an agile approach called WebTDD [18] which has a TDD (Test Driven Development) style of development; however and differently from "conventional" TDD [1], instead of relying on an extreme coding approach, we use models to generate the application. Using models we raise the level of abstraction as the application is automatically derived from them [18]. Our approach incrementally adds requirements to the existing application, following a short development cycle. WebTDD uses a DSL (Domain Specific Language) called WebSpec [17] to specify these requirements.

In this paper, we focus on how to specify personalization requirements and how to use this specification to improve the development process by automating some time-consuming and error-prone tasks. Summarizing, as the contributions of this paper, we show how to:

- Specify personalization requirements using a model-driven style.
- Automatically generate the conceptual models for the personalization functionality of the Web application, thus avoiding manual errors and the mismatch between the requirements and the implementation.
- Automatically generate tests from the requirements specification to validate the personalization functionality in the WebTDD cycle.

The rest of the paper is structured as follows: in Section II we briefly present the WebTDD approach. In Section III we show how personalization is specified in WebSpec and how we automatically derive interaction tests from the requirements specification. Section IV shows how the personalization model is automatically derived from the personalization requirements. Section V describes the implementation of our ideas. Section VI presents some related work and finally Section VII concludes and presents some further work we are pursuing.

II. WEBTDD IN A NUTSHELL

WebTDD is an agile approach which follows a TDD style of development, using models to generate the Web

application. Like most agile approaches, it is based on short development cycles; in each cycle new requirements are added and the application is upgraded incrementally.

The cycle starts by capturing requirements with mockups (stub HTML pages) to agree on the look and feel of the application, and WebSpec diagrams (Step 1 of Fig. 1) to represent navigation and interaction behaviours. WebSpec is a DSL which allows specifying navigation, interaction and user interface aspects in a more formal way (e.g. in comparison with use cases [10]).

Next we automatically derive (Step 2) a set of meaningful tests that the application must pass to satisfy the captured requirements. As in "conventional" TDD, we run them prior to the implementation (Step 3) in order to check that the application does not satisfy the requirements yet. Afterwards, the modelling activities begin (Step 4): we create or enhance a set of models and derive a running application (Step 5). We check whether each requirement has been successfully implemented by running the previous tests (Step 6). If one test fails, we have to go back, tweak the models and derive the application again until all tests pass. The approach continues with the next requirement until the sprint is over. We must notice that WebTDD is independent of the model driven Web engineering approach used for the modeling activities as the core of the process does not depend on the specific modelling artefacts or mechanics [18].

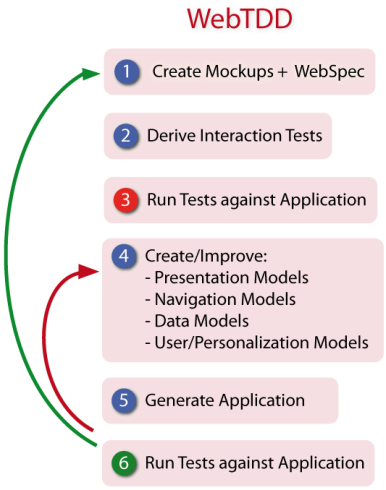

Figure 1. Approach overview

A WebSpec diagram has two key elements: interactions and navigations. An interaction (the counterpart of a Web page in the requirements stage) represents a point where the user can interact with the application by using its interface objects. Interactions may have widgets such us: labels, list boxes, buttons, radio buttons, check boxes and panels. Labels define the content (information) shown by an interaction. A diagram has a starting interaction which is represented with dashed lines. Some actions (clicking a button, adding some text in a text field, etc) might activate a navigation from one interaction to another. These actions are written in WebSpec's DSL which conforms to the syntax: var := expr | actionName(arg1,… argn).

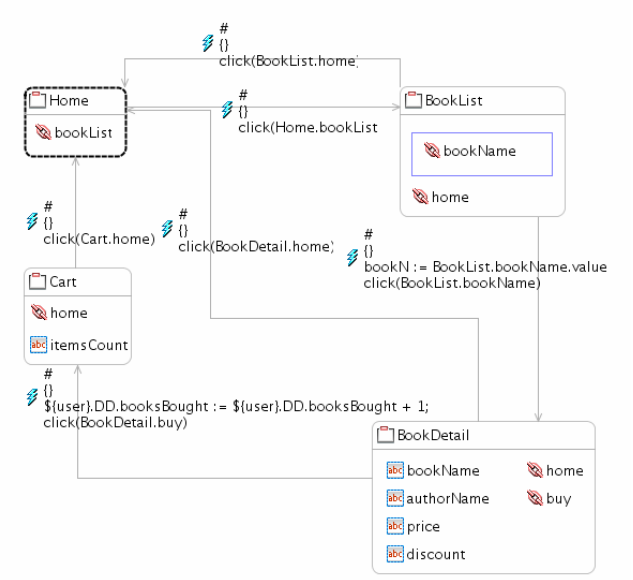

Figure 2. A WebSpec diagram

Fig. 2 shows a WebSpec diagram where navigation in a simplified E-commerce application is specified. The diagram shows how the user can move from one interaction to another thus allowing him to explore books, go back to the home page, buy a book and so on. To express what properties the diagram (and thus the application) must hold, we add invariants to the interactions (invariants are not shown in Fig. 2 for the sake of readability). For instance, the BookDetail interaction must satisfy the invariant BookDetail.bookName = ${bookN} which states that the value shown in the bookName label should be equal to the bookN variable (see in Fig. 2 the navigation from BookList to BookDetail where the variable is updated).

After we have specified the scenario in a diagram we can automatically derive a set of tests that the application must satisfy. This is an important feature of WebSpec, because, as in TDD, we use tests as software artefacts that decide whether the requirement is satisfied by the final application. However, instead of the typical unit tests of code-based TDD, we rely on interaction tests which fit better with Web applications. For this scenario, our support tool, the WebSpec Eclipse plug-in (Sect. V), generates a set of tests for the different paths that we can follow from the starting interaction. We next explain how to specify and validate personalization requirements in our approach.

III. Specifying and Validating Personalization Requirements

In this section we show how to specify personalization requirements using WebSpec diagrams. Moreover, these requirements can be validated by deriving a set of interaction tests, allowing to check if they are satisfied by the generated application, as explained in Sect. III. B.

A. *Specification of Personalization Requirements*

A personalization requirement describes some functionality that a Web application has to fulfil to (dynamically) adapt itself, depending on the user or environment profile [11]. In our approach we specify personalization requirements using WebSpec allowing their automatic validation. A WebSpec diagram specifies a personalization scenario that must be satisfied by the final application.

The conditions on which personalization requirements are defined usually refer to user-related information, which is traditionally specified in a so-called user model (UM). This user-related information can be classified in different types:

- User-specific characteristics (independent of the application domain) like age or country.
- Information related to the domain, for instance, from the user browsing behaviour we can derive the preferences or interests on different elements of the domain.
- Information related to the user context (e.g. device, network, actual location, etc).

In WebSpec we use a special variable named ${user} to denote the different elements associated with the UM. Since, during the requirement elicitation phase the UM does not exist, we assume that the ${user} variable is a prototype [14] on which we can add properties simply by accessing it and assigning it a value (e.g. ${user}.age:= 32). To refer to user-specific characteristics or user-context information, we directly access the property of the user variable, e.g. ${user}.age. In the case of domain dependent information we add the DD prefix, e.g. ${user}.DD.booksBought.

The personalization actions can be specified over the content, the navigation or the presentation of the Web application. Though personalization of the presentation is out of the scope of this paper, we can specify this kind of requirements by associating mockups to interactions (which is usual in WebSpec). Concretely, the personalization actions that can be specified in WebSpec are the following:

- Updating user information: In WebSpec we can specify updates on attributes of the UM by adding actions to the *navigations* of a diagram. The syntax is ***${user}****.attribute := value* where the value can be a literal or a formula.
- Filtering contents of the site: In WebSpec the labels of the different *interactions* can be filtered according to a condition. This is specified by means of invariants associated to the *interactions* of a diagram. To indicate if a label is shown or not, we use the "visible" property. The syntax is as follows: *label.****visible*** *<--> (Boolean expression).* The Boolean expression can also contain a loop, depending on the condition we want to express.
- Filtering the navigation: The links to be shown can also be selected by means of the "visible" property, by specifying invariants over the *interactions* of a diagram. The syntax is as follows: link.***visible*** *<--> (Boolean expression).*

In order to illustrate the described concepts, let's consider a simple E-commerce application in which our stakeholders want to personalize the discounts offered to customers, depending on how many books they have already bought. In particular, we would like to offer discounts in the book detail page when the user has already bought 2 or more books.

Following the approach (Fig. 1), we start capturing the requirements using WebSpec diagrams. This personalization requirement implies that the application must perform at least two actions. First, it must record how many books the user has already bought, and then it has to show the discount information in the book detail page, depending on how many books he has already bought. The first action is performed when the user navigates from the BookDetail to the Cart interaction (Fig. 2). The navigation has the side effect of adding the book to the shopping cart and thus incrementing the books that the user has already bought. We express it in the action of the navigation as follows:

```
${user}.DD.booksBought:=${user}.DD.booksBought + 1
```

This information is domain dependent, so the prefix DD is added to the attribute to update it as explained before. The second action is expressed in the invariant of the BookDetail interaction. The invariant relates the visible attribute of the label and a condition that must hold to let it be visible:

```
BookDetail.discount.visible <-->
(${user}.DD.booksBought >= 2)
```

Concretely, the discount label is visible if the user has already bought 2 or more books.

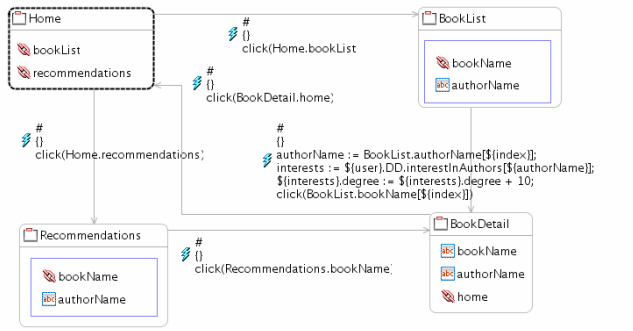

Figure 3. Recommendations personalization scenario

Another example of personalization is a recommendations feature (see Fig. 3); we would like to recommend books of those authors that the user is interested in, using his browsing history. For example, if the user has visited many books of Jose Luis Borges, we could guess that he is one of his favourite authors. This requirement needs first to decide how the users' interest is captured. We decide to increase the degree of interest when the user navigates to the book details:

```
authorName := BookList.authorName[${index}];
interests :=
${user}.DD.interestInAuthors[${authorName}];
${interests}.degree := ${interests}.degree + 10;
```

```
click(BookList.bookName[${index}])
```

The first action gets the author name. Then we retrieve the information of the interest of the user in the author (interestInAuthors) from the domain dependent information and increase it in 10. Finally, we click on the book's name to move from the BookList to the BookDetail interactions. These 4 actions store the activity of the user that can be later used to show / hide its favourite authors.

Additionally this requirement requires hiding the link that points to the recommendations node when we do not have enough information about the user's interests. So, we specify its visibility in the Home's invariant in this way:

```
Home.recommendations.visible <--> (Exists a in
${user}.DD.interestInAuthors / a.degree >= 100).
```

The above specification states that if there is an author that the user is interested in (degree > 100) then we should show the recommendations link.

B. *Derivation of Interaction tests for Requirements Validation*

After a requirement has been specified by means of WebSpec diagrams, we are able to automatically derive meaningful interaction tests to assess whether the requirement has been successfully implemented (see Fig 1, step 2). An interaction test opens a Web browser and executes a set of actions in the same way a user would do it. Interaction tests allow making assertions on HTML elements based on XPath expressions so we can check the values of the different widgets.

For each diagram, we create a test suite. Each path depicted in the diagram will be translated into a test case that will be named as the complete path's trail. A test case will follow the actions specified in the path, and assertions will be generated from the invariants of every interaction. The actions specified on navigations will be translated into sentences in the test, for example typing text into a text field or clicking buttons. Reaching an interaction will require that we check its invariant (if any), by generating assertions on the test. As different interactions may alter the variables bound to an invariant, it may be necessary to repeat the updated assertions after navigating to the same interaction more than once.

For example, the discount personalization diagram (see Fig. 2) is derived into the following interaction test (in Selenium [21]). Line 1 opens de application. Lines 2-11 add 2 books to the cart and assert that the discount is not present yet. Lines 12-14 navigate to the book detail page and validate that the discount is present (because the user has already bought 2 books).

```
(01) selenium.open(
       "http://localhost:8080/bookstore");
(02) selenium.click("id=bookList");
(03) selenium.click("id=book1");
(04) assertFalse(selenium.isElementPresent(
       "id=discount"));
(05) selenium.click("id=buy");
(06) selenium.click("id=home");
(07) selenium.click("id=bookList");
(08) selenium.click("id=book2");
(09) assertFalse(selenium.isElementPresent(
       "id=discount"));
(10) selenium.click("id=buy");
(11) selenium.click("id=home");
(12) selenium.click("id=bookList");
(13) selenium.click("id=book3");
(14) assertTrue(selenium.isElementPresent(
       "id=discount"));
```

After the test derivation process is completed we can run the tests to ensure that the application does not satisfy the requirement yet (Step 3); the same tests will be run again when the requirements have been implemented. The personalization model (Step 4) will be automatically derived from the WebSpec diagrams as shown in the following section.

IV. AUTOMATIC GENERATION OF THE PERSONALIZATION MODEL

Once the personalization requirements have been specified and the tests have been generated, we focus on how to automatically derive concrete software artefacts that implement the personalization functionality from the personalization requirements. In this way, the mismatch between requirements and the developed application is avoided. The generation of such software artefacts leads to an application that satisfies the personalization requirements expressed in the WebSpec diagrams.

In this case, the software artefacts generated from the personalization requirements are personalization rules. We have chosen to specify these rules using the PRML (Personalization Rules Modelling Language) language [7]. PRML is a rule-based high level language devised to specify personalization in an orthogonal way upon Web applications, independently of the underlying methodology. PRML has been successfully used in several Web methodologies and applied to several Web systems and an engine to perform and validate these rules has been implemented [7].

In the following subsections we present how to derive the PRML rules from the WebSpec specifications in a formal way. We also show an intuitive example of PRML rule generation, and finally we explain how to build the UM from the personalization rules.

A. *Deriving PRML rules*

By automatically generating the personalization model, we provide the designer a first set of personalization rules that he can refine or modify later. This helps avoiding many manual errors and inconsistencies. In order to transform WebSpec diagrams into PRML rules, we use the MOF 2.0 Query/View/Transformation language (QVT) [15] which is a standard transformation language in the context of the MDA (Model Driven Architecture) initiative. QVT is the means for defining formal and automatic transformations between models. Defining transformations by specifying QVT relations has several advantages: (i) transformations are formally established, easy to understand, reuse and maintain, (ii) they do not have to be manually performed by an expert,

which is a tedious and time-consuming task, and (iii) relations can be easily integrated into an MDA approach.

The objective of QVT is to define a formal mapping of the elements of a source metamodel (e.g. WebSpec) into a target metamodel (e.g. PRML). The PRML metamodel can be checked at [7] and the WebSpec metamodel is shown in Fig.4.

The generation of a PRML rule from a WebSpec diagram is defined by a sequence of transformations (QVT relations). A PRML rule is derived from a set of actions specified in WebSpec diagrams. As PRML rules are event-condition-action rules, each of these three parts should be derived from WebSpec specifications:

Depending on the type of WebSpec interaction performed by the user (e.g. navigation, diagram setup actions, etc), we can generate the different PRML events.

- PRML conditions are automatically translated from WebSpec conditions.
- The *actions* of PRML rules are derived by taking into account the different expressions specified in each of the actions of a WebSpec diagram. For instance, we can derive a PRML setContent action (which updates the user information in the UM) from an assignment expression in WebSpec. We can derive actions which filter the attributes to be shown or the links (e.g. selectAttribute and hideLink in PRML) by checking the "visible" attribute of the WebSpec *WidgetReference* element of the metamodel.

Due to space limitations, we cannot show all the QVT rules we have defined. In Fig. 4, the QVT rule for deriving the PRML SetContent action is shown as an illustration. This relation checks that there is a set of elements in the WebSpec action that represents an assignment expression according to the WebSpec metamodel (see Fig. 5). These elements are: an assignment class together with the corresponding variable to assign the value, and the value (e.g. an expression) to be assigned. The relation enforces that the corresponding PRML expression has the following elements: a setContent class, and an expression that expresses the assignment of a value to a UM variable.

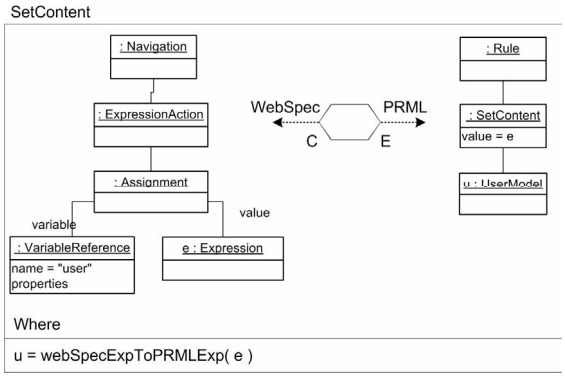

Figure 4. SetContent QVT transformation

To intuitively illustrate the rule generation process, let us consider the discount requirement example explained before (Fig. 2). As aforementioned (Sect. III), this requirement is derived into two PRML rules. The first one (i.e. acquisition rule) acquires/updates the number of books bought by the user in the UM. The second one (i.e. personalization rule) shows/hides the discount attribute to the user based on the previously acquired information (i.e. books bought).

The acquisition rule determines the moment (navigation), condition (always) and the action (increase the value of the variable in the UM). Then, from the navigation in the WebSpec diagram (see Fig. 2) we derive the following PRML rule:

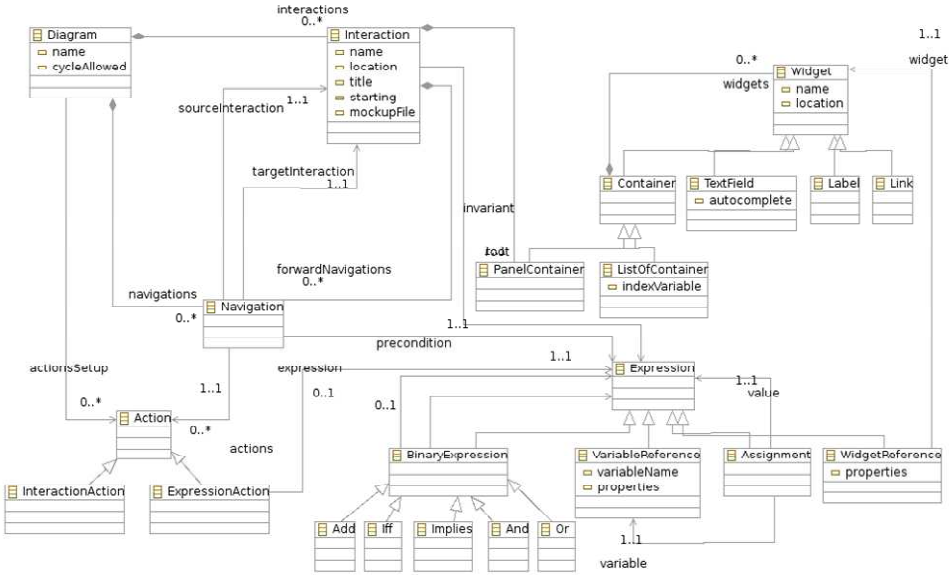

Figure 5. WebSpec's metamodel

```
When Navigation.BookDetailBuy(NM.Book book) do
        setContent(UM.User.booksBought,
        UM.User.booksBought + 1)
endWhen
```

In a similar way, we derive the personalization rule from the BookDetail invariant (see Sect. III). Since personalization takes place every time the node is loaded, the PRML event derived is LoadElement. The condition corresponds with the right part of the WebSpec iff Boolean expression, and the selectAttribute action matches the left part of the iff because it references the visible property of a label. The PRML rule derived is shown next:

```
When LoadElement.BookDetail(NM.Book book) do
If (UM.User.booksBought >= 2) then
        book.Attributes.selectAttribute(discount)
endIf
endWhen
```

In the following section we show how we incrementally implement the UM using the derived PRML rules as a starting point.

B. *Incremental Implementation of the UM*

In the previous section we showed how a set of personalization rules in the PRML language are derived. These rules express the Event-Condition-Actions that have to occur to personalize the application. Since we are deriving these rules from the requirements following a top down process, the UM may not reflect yet the functionality expressed on them. For instance, the first time we derive the rules, the User class may not even exist. Additionally, when the application has been deployed the UM may not reflect a new attribute that has been added by a new requirement. All these problems are detected by the PRML engine [7] when it validates the generated rules. The validation process will fail showing which parts of the UM do not exist yet.

Using the same philosophy of TDD, we create/enhance the UM in an incremental way by trying to validate the derived rules. The validation process will show which information is not yet present in the UM. For each attribute or class that does not exist in the UM, we create it manually and run the validation process again until the validation succeeds. In this way, we drive the development of the UM using the rules that were automatically generated in the previous step making it a straightforward process.

As an example, let us consider the first rule of the previous subsection. Assuming that the User class already exists, we run the PRML rule validation which fails because the booksBought attribute does not exist in the User class. To make the validation pass, we go to the class and add the instance variable of type number. Then, we run the validation again and finally the validation will succeed.

V. IMPLEMENTATION

WebSpec has been implemented as an Eclipse plugin (Fig. 6) using EMF and GMF technologies. It supports the specification of personalization requirements by means of diagrams that the user can create within the environment and using the palette on the left side of the diagram editor, the user can create concepts like Interactions and Navigations and complete the diagram with the personalization specification.

The automatic derivation of interaction tests is performed using a JUnit class writer that satisfies the syntax needed by the Selenium framework. Also, during test derivation, expressions are optimized for better readability. For

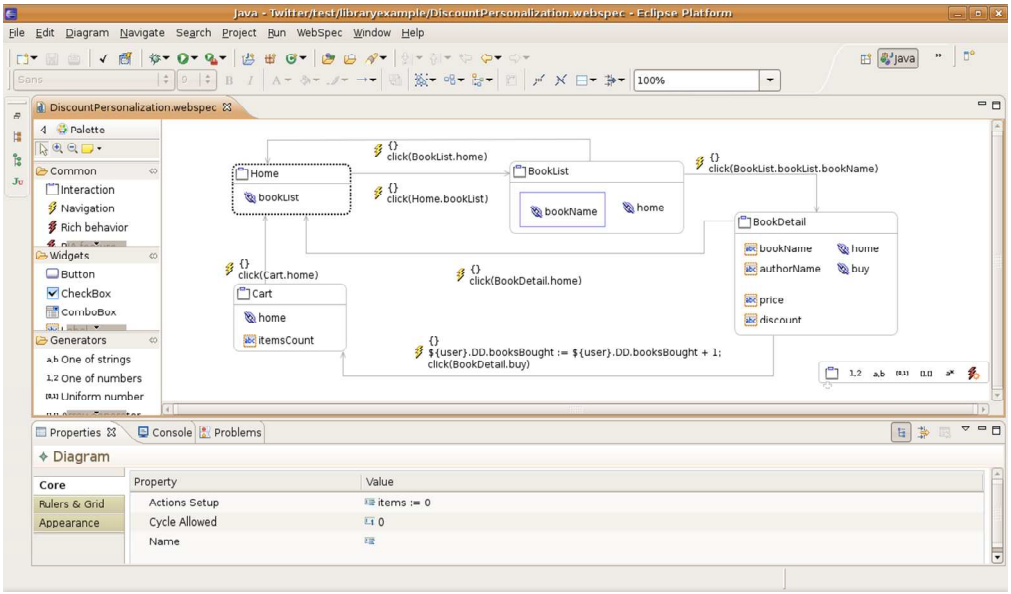

Figure 6. WebSpec's Eclipse plugin

example, an expression like: ${long} -> Home.username = ”John” where the long variable has the value false (-> means the implies relationship) is automatically optimized to true using Boolean equivalencies. We have chosen JUnit and Selenium because they are easily integrated in Eclipse though other web testing framework such as Watir [22] can also be used.

The automatic derivation of PRML rules is easily performed as PRML is also implemented as an Eclipse plugin thus allowing to seamless integrate both approaches. The WebSpec menu has options to allow the derivation of PRML rules that are automatically imported in the PRML prototype tool. PRML rules are plain text files thus the generation of such rules is easily performed by a model to model transformation from the WebSpec´s metamodel to the PRML metamodel. Then we reuse the transformation process of the PRML tool to use the model to text transformation.

We have used the WebSpec plugin with the PRML tool to implement a personalization version of the E-commerce application. Several personalization requirements have been specified and validated using the derived tests in the context of the WebTDD approach. We have used interaction tests to drive the development of the personalization and functional requirements. Tests were used to check that the new requirements have been correctly implemented and that we have not been unintentionally broken existing functionality. The personalization model was derived from the specification thus avoiding the mismatch between requirements and the implementation. However, as previously mentioned, we have to follow a short TDD cycle to complete the derivation as it only covers some structural aspects of classes in PRML. We expect to improve the derivation process in future work.

VI. Related Work

In the context of Web engineering, few approaches have focused on defining an explicit requirement analysis stage to model the user expectations. Some approaches consider the modelling of personalization to some extent [3], [8], [9], [12], [20]. In general those approaches ignore how personalization requirements are captured.

A-OOH [8] is a model-driven approach which allows the specification of personalization requirements. It uses the i* framework in order to specify a goal-oriented requirements model. From this specification, the conceptual models (e.g. domain and navigation models) are generated by means of QVT transformations. However, A-OOH does not allow the derivation of the personalization model as done in WebSpec.

In [13], in the context of OOHDM [20], personalized UIDs are used to capture a personalized version of the interactions that users have with the application. The difference with traditional UIDs is that they may have many initial interactions one for each different type of user. Webspec and personalized UIDs share the same terminology as WebSpec is based on UIDs; however, personalized UIDs do not provide automatic transformations to software artefacts, so there may be a mismatch between requirements and the final application.

Adaptative OOWS [19] extends OOWS [16] to support adaptation. It propose two artifacts to specify adaptative requirements: an enhanced of Activity Diagrams called User Stereotype Diagrams and their corresponding User and Data Specifications descriptions which capture the adaptive part of the requirements by means of intuitive and easy-to-understand schemas. Afterwards, the requirements models serve as a basis to derive the conceptual specifications of users and adaptive features in the OOWS Conceptual Modeling phase. This approach share some common features with the work presented in this paper such as: specification of requirements and derivation of the User model. However, the approach does not provide automatic ways to validate that the adaptative requirements are correctly implemented in the application. Requirements validation is extremely important to ensure that the behavior of the application is preserved, e.g. when maintainability needs to be improved by means of model refactorings.

In [6], Escalona and Koch have proposed a metamodel based on WebRE profiles to specify web requirements. Its main advantage is the automatic generation of conceptual models (content and navigation models) which automatically satisfy the requirements. Also, some tests are derived from the profiles to validate that the functionality has been correctly implemented. However, some requirements such as detailed composition of the user interface and specifically personalization requirements can not be specified thus requirements cannot be validated and the personalization models can not be derived using this notation.

In summary, the described approaches are, as far as the authors are concerned, the only that allow specifying personalization requirements, however they have the following drawbacks:

- They do not allow the automatic derivation of the personalization artefacts (personalization and UM). Doing so we avoid many manual errors and we assure that the defined model is aligned with the previously specified requirements.
- They do not provide a way to validate personalization requirements. Automatic validation using tests helps not only to validate the correct implementation of the personalization requirements, but it also helps to detect unintended errors when the application grows.

WebSpec supports the specification of Personalization requirements and can be used in different development processes to implement the personalization functionality. To the authors’ knowledge, the work presented in this paper is the first to provide test derivation and partial UM derivation from a requirement artefact specifically for Personalization. In addition to the advantages shown in this work, we can use WebSpec in conjunction with mockups to improve the communication between stakeholders while capturing the personalization requirements as shown in [17].

VII. Conclusions and Further Work

In this paper we have presented an approach for dealing with personalization requirements in Web applications. Requirements are captured in WebSpec diagrams which

allow us to derive a set of tests to validate requirements, and to automatically derive the personalization rules in the PRML language. In addition, we have shown how the UM can be incrementally implemented by validating the generated rules in the PRML engine. The idea has been presented in the context of WebTDD, an agile approach for developing Web applications, but it can be applied to any other Web methodology.

We are currently working on the automatic derivation of the UM which is, until now, done manually (as shown in Sect. IV B). Furthermore, we are working on some field experiences with the usage of mockups to help on developing the look and feel of the personalization functionality. Finally, we are analyzing how personalization requirements evolve and how we handle this evolution along the development cycle.

ACKNOWLEDGEMENTS

This work has been partially supported by the MANTRA project (GRE09-17) from the University of Alicante, and by the MESOLAP (TIN2010-14860) from the Spanish Ministry of Education and Science.

REFERENCES

[1] Beck, K.: Test Driven Development: by Example, Addison-Wesley, 2003.

[2] Casteleyn, S., Garrigós, I., Troyer, O.D.: Automatic runtime validation and correction of the navigational design of web sites. In: APWeb. (2005) 453–463.

[3] Ceri, S., Daniel, F., Matera, M., and Facca, F. M. 2007. Model-driven development of context-aware Web applications. ACM Trans. Internet Technol. 7, 1 (Feb. 2007), 2.

[4] Ceri, S., Manolescu, I.: Constructing and integrating data-centric web applications: Methods, tools, and techniques. In: VLDB. (2003) 1151.

[5] Escalona, M.J., Koch, N.: Requirements engineering for web applications – a comparative study. J. Web Eng. 2(3) (2004) 193–212.

[6] Escalona, M.J., Koch, N. Metamodeling Requirements of Web Systems. In Proc. International Conference on Web Information System and Technologies (WEBIST 2006), INSTICC, 310--317, Setúbal, Portugal. 2006.

[7] Garrigós, I.: A-OOH: Extending Web Application Design with Dynamic Personalization. PhD thesis, University of Alicante, Spain (2008)

[8] Garrigós, I., Mazón, J.N., Trujillo, J.: A Requirement Analysis Approach for Using i* in Web Engineering. In: ICWE. (2009), LNCS, 5648, 151-165.

[9] Houben, G.-J., Frasincar, F., Barna, P. and Vdovjak, R. (2004). Engineering the presentation layer of adaptable web information systems. In Web Engineering 4th International Conference, ICWE 2004, volume 3140 of Lecture Notes in Computer Science, pages 60-73, Springer, ISBN 3-540-22511-0.

[10] Jacobson, I., Object-Oriented Software Engineering: A Use Case Driven Approach, ACM Press/Addison-Wesley, 1992.

[11] Kim, K: Personalization: Definition, Status, and Challenges Ahead. Journal of Object Technology, 1, (2002) 29-40.

[12] Koch, N.: Reference model, modeling techniques and development process software engineering for adaptive hypermedia systems. KI 16(3) (2002) 40–41.

[13] Martin, A. Cechich, A: A Model-Driven Reengineering Approach to Web Site Personalization. In Proceedings of the Third Latin American Web Congress (October 31 - November 02, 2005). LA-WEB. IEEE Computer Society, Washington, DC, 14.

[14] Noble, J., Taivalsaari, A., Moore, I. (eds.): Prototype-Based Programming: Concepts, Languages and Applications. Springer-Verlag. ISBN 981-4021-25-3. 1999.

[15] QVT Language: http://www.omg.org/cgi-bin/doc?ptc/2005-11-01.

[16] Pastor, O., Abrahão, S., Fons, J.: An Object-Oriented Approach to Automate Web Applications Development. In: Bauknecht, K., Madria, S.K., Pernul, G. (eds.) ECWeb 2001. LNCS, vol. 2115, pp. 16–28. Springer, Heidelberg (2001).

[17] Robles Luna E., Garrigos I., Grigera J., Winckler M. Capture and Evolution of Web requirements using WebSpec. To be published in the Proceedings of 10th International Conference on Web Engineering (ICWE 2010).

[18] Robles Luna, E., Grigera, J., Rossi, G.: Bridging Test and Model-Driven Approaches in Web Engineering, Web Engineering, Lecture Notes in Computer Science, pp. 136-150, Springer, Heidelberg, June 2009.

[19] Rojas, G., Valderas, P., and Pelechano, V. 2006. Describing Adaptive Navigation Requirements of Web Applications. In Proc. of the 4th International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems (AH 2006), Dublin, Ireland. LNCS 4018. 318–322.

[20] Schwabe, D., Rossi, G.: An object oriented approach to web-based applications design. TAPOS 4(4) (1998) 207–225

[21] Selenium, a Web application testing system, http://seleniumhq.org/

[22] Watir, http://watir.com/