Detecting Conflicts and Inconsistencies in Web Application Requirements

Matias Urbieta[1,3], Maria Jose Escalona[2], Esteban Robles Luna[1], and Gustavo Rossi[1,3]

[1] LIFIA, Facultad de Informática, UNLP, La Plata, Argentina
{murbieta,esteban.robles,gustavo}@lifia.info.unlp.edu.ar
[2] IWT2 Group. University of Seville, Spain
mjescalona@us.es
[3] Conicet

Abstract. Web applications evolve fast. One of the main reasons for this evolution is that new requirements emerge and change constantly. These new requirements are posed either by customers or they are the consequence of users' feedback about the application. One of the main problems when dealing with new requirements is their consistency in relationship with the current version of the application. In this paper we present an effective approach for detecting and solving inconsistencies and conflicts in web software requirements. We first characterize the kind of inconsistencies arising in web applications requirements and then show how to isolate them using a model-driven approach. With a set of examples we illustrate our approach.

1 Introduction

Eliciting web application requirements implies understanding the needs of different stakeholders, those that are related with the same underlying enterprise business. Most of the times, requirements are agreed by stakeholders in such a way that the semantics and meanings of each used term is well understood; however when different points of view [11] of the same business concept exist, ambiguities and/or inconsistencies may arise, being them detrimental to the Software Requirement Specification (SRS). Traditionally, conciliation tasks are performed using meeting-based tools, in order to eliminate requirements ambiguity and inconsistence. When requirement inconsistencies are not detected on time -being this one of the most severe reason of project cost overrun [12][17]-, they may become defects in the web software. In this context, the effort to correct the faults is several orders of magnitude higher than correcting requirements at the early stages [12].

Inconsistencies may also arise from new requirements, which introduce new functionality or enhancements to the application or, even, for existing requirements that change during the development process. For example, an online e-commerce site may plan a promotion for Christmas, where some products have free shipping for a period of time; meanwhile other products keep the usual shipping cost. This new

requirement introduces changes that are perceived by the user because he can see promotional banners in different pages. It is noteworthy that the existing "shipping" requirement is overridden (and contradicted) with the shipping cost exception, introducing ambiguities: what products have the free shipping promotion? In which way users are notified? How long will the promotion be available?

In this paper we present a model-based validation and inconsistency detection technique for web application requirements, particularly for those that reflect themselves during navigation and interaction, two aspects are the key features of web applications. Though we exemplify our technique with WebSpec[15], the same ideas can be easily applied to other similar approaches such as WebRE[8] or Molic[6]. By using this technique we reduce the risk of errors and costs caused by inconsistencies detected in the final stages of software development.

The main contributions of this paper are threefold: a characterization of web application requirement inconsistencies depending on a taxonomy for conflicts; a modular approach for detecting inconsistencies that can easily complement any web application engineering process no matter its style: agile or unified; and a set of running examples to illustrate our approach.

The rest of this paper is structured as follows. Section 2 presents some related work in requirements validation. Section 3 introduces the background for the paper. Section 4 presents our characterization of web requirement conflicts. Section 5 describes our approach to detect and deal with inconsistencies. Section 6 presents a tool which provides support for conflict detection analysis. Finally Section 7 concludes this work discussing the lessons learned, our main conclusions and some further work on this subject.

2 Related Works

The analysis and detection of conflicts in the requirements phase are one of the most critical tasks in requirements engineering [15]. A global view presented in [7] divides this phase in three main tasks: requirements capture, requirements definition and requirements validation. The detection of conflicts is normally executed in the last one. In [7] the authors surveyed the way in which web engineering approaches dealt with these three phases and conclude that requirements validation is one of the less treated. Besides, none of these techniques offers a systematic detection of conflicts in requirements. Approaches studied in this survey support four main techniques for requirements validation: reviews, audits, traceability matrix and prototypes. In [16] this set is enriched adding requirements test. It consists in the generation of early test cases derived from requirements, which enables the early validation with users.

Recently, some web design approaches, such as WebML[5], support this idea using the model-driven paradigm. However, even offering systematic (or even automatic) support for early testing, the detection of inconsistencies in the requirements specification continues being "too artisanal" and depends on the analyst's experience and his/her capability for supporting the review with customers and users.

Focusing only on the detection of conflicts, in [3], an approach to detect conflict in concerns is presented. In this approach, the authors propose the use of a Multiple Criteria Decision Making method to support aspectual conflict management in aspect oriented requirements. The main limitation of this approach it that it is oriented to aspect-oriented requirements treatment and it only deals with concern conflicts.

In other phases of the life cycle, the conflict detection process has been researched intensively by the model-driven community mainly focused to UML model conflicts. In [1] the author proposes detecting conflict in a twofold process: analyzing syntactic differences raising candidate conflicts and understanding these differences from a semantic view.

3 Background

In this work we focus on detecting conflicts in web applications requirements which are modeled using WebSpec, a web requirement meta-model describing interactions, navigations and interface aspects.

WebSpec[15] is a visual language; its main artifact for specifying requirements is the WebSpec diagram which can contain *interactions*, *navigations* and *rich behaviors*.

A WebSpec diagram defines a set of scenarios that the web application must satisfy. An *interaction* (denoted with a rounded rectangle) represents a point where the user can interact with the application by using its interface objects (widgets). *Interactions* have a name (unique per diagram) and may have widgets such as labels, list boxes, etc. In WebSpec, a *transition (*either *navigation* or *rich behavior)* is graphically represented with arrows between *interactions* while its name, precondition and triggering actions are displayed as labels over them. In particular, its name appears with a prefix of the character '#', the precondition between { } and the actions in the following lines.

The scenarios specified by a WebSpec diagram are obtained by traversing the diagram using the depth-first search algorithm. The algorithm starts from a set of special nodes called "starting" nodes (*interactions* bordered with dashed lines) and following the edges (*transitions*) of the graph (diagram).

As an example of WebSpec's concepts we present in Fig. 1 the specification for the user story: "As a customer, I would like to search products by name and see their details" in an e-commerce application. *Home* represents the starting point of the specification and it contains 2 widgets: *searchField* text field and *search* button (see [15] for further details).

4 Characterizing Requirements Conflicts in Web Applications

During requirement specification, there may be cases where two or more scenarios that reflect the same business logic differ subtly from each other producing an inconsistency. When these inconsistencies are based on contradictory behaviors, we are facing a conflict of requirements [10]. Conflicts are characterized by differences of objects' features, logical (what is expected) or temporal (when is expected) conflicts between actions, or even difference of terminology that creates ambiguity.

In this analysis, we will emphasize on web application navigation, as well as user interaction peculiarities that are not covered in the traditional characterization of requirement conflicts [10]. Consequently, we provide an interpretation of each conflict type in the web application realm, using simple but illustrative examples. We use WebSpec terminology to specify the requirements.

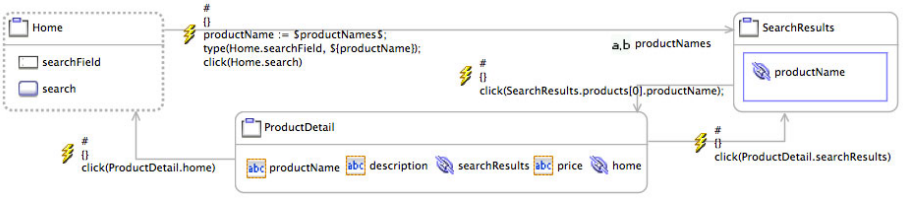

Fig. 1. WebSpec diagram of the *Search by name* scenario

Structural conflicts stand for a difference in the data expected to be presented in one web page by different stakeholders. A stakeholder may demand a data to be shown in a web page that contradicts other stakeholder requirement. For example, a stakeholder expects a product content description just as a read-only label, while another one may expect the content as a list of packaged items with an overall description contradicting the first requirement.

Two web application requirements may contradict the way in which links are traversed producing navigational conflicts, e.g. having a single source node but two targets. The target nodes are different, but the event that triggers the navigation and the condition guards are the same, producing an ambiguity of such requirement. In WebSpec terms, for a given navigation sequence (or path) composed with *interactions* and *navigations*, there are two navigation alternatives triggered by the same event. For instance, a WebSpec *navigation* can define that after clicking the "Buy" button at the Product *interaction*, a shopping cart is presented. On the other hand, the same *navigation* has as target the PaymentMethod *interaction*, which allows selecting a payment method instead of presenting the Shopping cart.

A semantic conflict occurs when the same real world object is described with different terms. This situation may generate a false negative in the conflict detection process, since a conflict may not be detected and new terms are introduced into the system space thus increasing its complexity. As a consequence the same domain object is modeled in two entities having different terminology. For instance, an e-commerce site can wrongly define two entities that stand for the same concept: Good and Product.

5 Detecting and Correcting Conflicts

Next we present our approach that helps detecting conflicts checking the existences of false positives and false negatives conflicts. The approach comprises the following steps, depicted in Fig. 2 (notice that steps 1 and 2 are already part of any development process; therefore the novel contribution begins in step 3):

1. Requirement gathering: Using well-known requirement elicitation techniques such as meetings, surveys, Joint Application Development (JAD), etc. a Software Requirement Specification (usually in natural language) is produced. In the case of an agile underlying development process, a briefer description is usually produced with user stories [4]; use cases are often used in a unified process style.
2. Requirement modeling: Web application requirements are formalized using a requirement domain specific language (DSL) (e.g. WebSpec, WebRE or Molic). This formalization is essential during the validation process with stakeholders. By means of using a requirement DSL, the validation process can automated.
3. Structural analysis of the web requirements model: by means of an algebraic comparison of models, candidate structural and navigational conflicts are detected. Additionally, navigation paths are evaluated for checking their consistency.
4. Semantic analysis: candidate conflicts are analyzed and semantic equivalences are detected. For each candidate conflict, both the new requirement and the compromised requirement are translated from a high abstraction level (the requirements DSL) to a minimal form, using an atomic constructor in order to detect semantic differences.
5. Conciliation process: once the existence of a conflict is confirmed, we must start conciliating requirements. This process demands the establishment of a communication channel among those stakeholders concerned to the conflict.
6. Refinement: When a conflict is confirmed some adjustment and tuning must be done in order to remove the detected conflict and reach a consistent state.

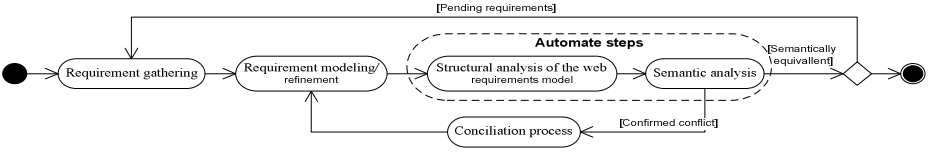

Fig. 2. The overall process for detecting requirement conflicts

The process is applied iteratively each time a new set of requirement rises. The new incoming set of requirements is checked with each one of the already consolidated requirements of the system space. In Fig. 2, those steps that can be implemented to be automated are grouped with a dashed box and those steps outside the dashed box are manually elaborated.

Show product information
As a customer **I want** to be able to see quickly product's information from a list of products **So that** i can see a detailed view of product features

Show product summary
As a customer **I want** to be able to see quickly product's summary when listed **So that** i can see product's features

Fig. 3. User stories for gathered requirements

5.1 Requirement Gathering and Requirement Modeling (Steps 1 and 2)

In order to describe clearly and accurately the aforementioned process, we use as a running example the development and extension of an e-commerce site. In Fig. 3, user stories [4] derived from gathered requirements are shown. Instead of including in this section the corresponding WebSpec diagrams, we show them in each of the subsequent steps.

5.2 Detecting Syntactic Differences (Step 3)

A candidate conflict arises when the set of syntactic differences between requirement models is not empty. These differences may be a consequence of the absence of an element in one model but present in the other, the usage of two different widgets for describing the same information, and finally a configuration difference in an element such as the properties values of a widget. This situation may arise when two different stakeholders have different views of a single functionality, or when an evolution requirement contradicts an original one. As the result of having a formal tool for describing requirements, the detection task can be implemented by reasoning over the specification. In this case using the WebSpec support tool [15], this task can be performed using OCL [14] sentences or RDF [9] queries.

Structural conflicts detection can be implemented by a comparison operation between interactions, in order to detect the absence of elements or elements constructions differences. Since WebSpec *interactions* are containers of widgets, we can apply set's difference operations in order to detect inconsistencies. For example, a Product interaction version called Product[1] have *Name*, *Valorization* and *Content* Labels, and an *addToShopping* Button and, on the other hand, a different version called Product[2] comprises a *Name*, and *Description* Label, and a list of *PackageItem* Labels. After applying the symmtric difference, following widgets differs: *Valorization*, *Content*, *addToShoppingCart*, *Description*, and a list of *PackageItem*.

Notice that for the comparison operation, two elements are equal if and only if they have the same identifier and have the same widget type and compatible configuration.

To detect navigational conflicts, outgoing navigations from a given node with identical triggering events but different targets must be detected. The task is pretty straightforward; since *navigations* are described by a guard and a set of actions that trigger them, the *navigations* for a given *interaction* must be compared to each other taking into account their guards and set of actions. The main challenge of this procedure is to check whether or not the sets of actions that correspond to navigations are semantically equivalent considering that the actions can be syntactically different.

Next we introduce an analysis process that helps avoiding false positives.

5.3 Semantic Analysis (Step 4)

As the result of the structural analysis of models, a list of candidate conflicts is reported; this list must be verified in order to detect false positives, i.e. conflicts that actually are not conflicts since the compromised specifications describe the same requirement. This issue has been already studied in [1][13] where models are analyzed in order to expose their underlying goals. When the underlying goals are different, we are facing a confirmed conflict.

We use an approach proposed in [1] and based on having an additional semantic view of requirements that complements the existing syntactic view. For achieving this, requirements models are downgraded in terms of abstraction, obtaining a refined model formed only with semantically simple elements. The resultant model is larger than the source diagram but has the same semantics.

This approach is twofold: a meta-model called semantic view, defined as a reduced subset of the web application requirement DSL is specified, and a transformation is specified that takes elements from the source model to the "semantic view".

The compromised models (the new and the stable one) are transformed into a semantic view where the derived models are finally compared syntactically. For each conflict detected in step 3, this approach helps detecting false positives because the semantically equivalent constructions imply that different models specify the same requirement. In the other hand, models are compared when no conflict is detected to expose false negative cases.

We will use as semantic view a simplified WebSpec meta-model where the Transition's hierarchy and Container widgets are removed. The transition hierarchy is formed by two specializations - Navigation and RichBehavior - that are removed in order to focus on determining what is the intent of the *interaction*, independently of the used interaction pattern: traditional navigation or RIA interaction. When containers do not have a name, they are removed in order to reduce composition complexity and avoid unnecessary object aggregations.

Finally a model transformation must turn a WebSpec model into a semantic one in order to provide a simpler understanding.

In the transformation, a set of rules closely related to the Web requirement meta-model used are applied over the input model obtaining the semantic view. These rules are based on heuristics defined by the requirement engineer and the available set of rules must be improved iteratively by means of lessons learned of its application.

If other Web requirement meta-model is used such as WebRE, a different set of rules must be defined where each one must increase the abstraction level in such a way the intent of the model is emphasized.

Some of the rules for WebSpec meta-model comprised by the transformation are:

- Disabled TextFields are translated to Labels. As disabled TextFields do not allows user inputs these are replaced by simple Labels.
- Links are translated to buttons. Links and Buttons are usually used for describing an action triggering. Therefore, links are normalized to buttons.
- Navigations and RichBehavior are simplified into a single transition abstraction. This rule makes the diagram focus more on the data itself instead of the way in which it is accessed. Finally, Navigation´s and RichBehavior ´s actions are removed.

In order to detect if the syntactic conflict is in fact a conflict, the semantic transformation is applied over both requirement specifications. Both transformations produce the same model that is formed by Labels and a Button. Thus, as both semantic views are equal, there is not conflict at all.

The following example aims at illustrating how semantic conflicts are detected; in particular a false negative case. In Fig. 4 two requirements, namely "show product

information" and "show product summary" represent the same interaction idea but use two different interaction patterns: traditional web navigation and RIA´s mouse hover pattern.

The left-hand image specifies that after clicking the name of a product, the link is traversed and a product detail is shown. On the other hand, in the picture at the right, when the mouse´s pointer is place over the product´s name, a product detail is popped-up. It is remarkable that both requirements´ models have the same intent but are described with distinct WebSpec constructors.

The resultant of applying the transformation to both conflicted WebSpec is a pair of normalized diagrams that must be syntactically compared in order to detect differences. Fig. 5.a and Figure 5.b show the result of applying the transformation to the examples presented in Fig. 4.a and 4.b respectively where Navigations and RichBehavior were normalized into the more abstract Transitions, and the Home link was removed because it is not referenced anymore.

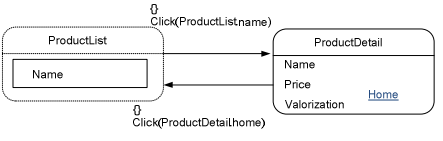

Fig. 4a. Specification of conventional navigation requirement.

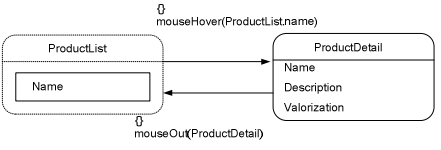

Fig. 4b. Interaction based on a RIA feature.

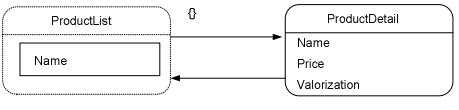

Fig. 5a. Normalized conventional navigation model into Semantic view.

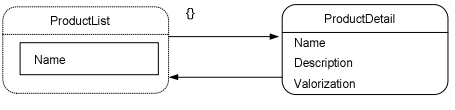

Fig. 5b. Normalized RRIA feature model into Semantic view.

Then a semantic conflict is detected because both models are not syntactically equal in the semantic view because Price and Description Labels are not present in both ProductDetail interactions (Fig. 5.a and Fig. 5.b).

There are cases were both traditional navigation and RIA features are required, in this case the raised warning for a false negative conflict must be omitted.

5.4 Conciliation Process (Step 5)

So far, we have shown how to detect conflicts that must be resolved in order to keep the SRS sound and complete. Next we will introduce a set of heuristics that helps resolving structural and navigation conflicts and that have been implemented as suggested refactorings in our tool support.

In the case of structural conflicts, the absence of a given widget in a model but present in the other, we can take an optimistic position understanding that the best

solution is to include the construction as an improvement when it is not present. This idea comes from the fact that new requirements may improve others requirement´s functionality; therefore the new requirement widget may enrich an existing interaction.On the other hand, the widget type incompatibility demands a deeper analysis understanding the context of the difference.

Navigational conflicts express ambiguity in the way in which the web application is browsed, having two targets (WebSpec *interactions*) in a *navigation* triggered by the same event. This situation is naturally resolved enriching the scenario in such a way that the conflict is dissolved because the scenario detail is increased. Since we are using WebSpec as a requirement modeling tool, there are two strategies available for disambiguating: adding precondition clauses or extending the scenario path; both increase scenario detail.

As we have previously introduced, different stakeholders may provide slightly different specification for the same application goal. Nonetheless, there are scenarios where it is more prone to face inconsistencies such as the presence of business objects' hierarchies. At the requirement elicitation stage, hierarchies of business objects may not be clearly detected and defined, and as a consequence several business objects structurally different are referenced with the same name.

6 Tool Support

We have extended the WebSpec tool [15] with a reasoning support that helps detecting inconsistencies in the requirement modeling process. The tool provides a consistency checker engine based on the Eclipse EMF OCL[14] query system. By means of executing OCL queries over diagrams both structural and navigational inconsistencies are detected. The tool automates the structural analysis of web requirement models, transformation of requirements into semantic view and the syntactic analysis discussed in Section 5. Its main intent of use is during the requirement gathering and requirement modeling steps of the process, as it aids analysts in the requirement modeling, requirement management, and consistency checking activities. The tool provides a consistency report is generated showing detected conflicts and compromised widgets. Finally, when inconsistencies are detected, candidates list of automatic and semiautomatic (those that require an input parameter) refactorings that correct inconsistencies are presented. Since conflicts can not be trivially resolved, the tool provides a list of refactorings that could be applied to resolve the problem. The analyst should decide which option is the best to be applied, and afterwards the tool will perform automatically the refactoring over the WebSpec diagrams.

7 Concluding Remarks and Further Work

We have presented a novel approach for detecting conflict and inconsistencies in web application requirements in the early stages of software development. The presented approach leans on a web requirement meta-model used for specifying, in a formal

way, the application requirements. Any new requirement is checked against the consolidated requirement set in order to detect conflicts. By means of syntactic and semantic analysis inconsistencies are detected. The approach is modular so it can be plugged in any software engineering approach to ensure application consistency, validate requirements, and save time and effort to detect and solve error in latest software development steps. Our support tool helps to automate the analysis and correction of these inconsistencies.

We have presented some simple examples that illustrate the approach feasibility but it still requires further work. We are currently working on the following issues: complete the approach with a set of ontology matching algorithms in order to improve semantic conflicts detection; extend the available heuristics for resolve detected conflicts in order to provide automated conflict detection and solving solution; and carry out an experiment instantiating the approach in order to provide evidence and to measure the time and effort effectively saved.

References

[1] Altmanninger, K.: Models in Conflict - Towards a Semantically Enhanced Version Control System for Models. In: MoDELS Workshops 2007, pp. 293–304 (2007)

[2] Boehm, B.W., Grünbacher, P., Briggs, R.O.: Developing Groupware for Requirements Negotiation: Lessons Learned. IEEE Software 18(3) (2001)

[3] Brito, I.S., Vieira, F., Moreira, A., Ribeiro, R.A.: Handling Conflicts in Aspectual Requirements Compositions. In: Rashid, A., Aksit, M. (eds.) Transactions on AOSD III. LNCS, vol. 4620, pp. 144–166. Springer, Heidelberg (2007)

[4] Cohn, M.: Succeeding with Agile: Software Development Using Scrum, 1st edn. Addison-Wesley Professional (2009)

[5] Ceri, S., Fraternali, P., Bongio, A., Brambilla, M., Comai, S., Matera, M.: Designing Data-Intensive Web Applications. Morgan Kaufmann Publishers Inc., San Francisco (2002)

[6] de Paula, M.G., da Silva, B.S., Barbosa, S.D.: Using an interaction model as a resource for communication in design. In: CHI 2005 Extended Abstracts on Human Factors in Computing Systems, Portland, USA, April 02-07, pp. 1713–1716 (2005)

[7] Escalona, M.J., Koch, N.: Requirements Engineering for Web Applications: A Survey. Journal of Web Engineering II(2), 193–212 (2004)

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

[9] Euzenat, J., Shvaiko, P.: Ontology Matching, 1st edn. Springer, Heidelberg (2007) ISBN: 978-3540496113

[10] IEEE Recommended Practice for Software Requirements Specifications. IEEE Std 830-1998 (1998)

[11] Kotonya, G., Sommerville, I.: Requirements engineering with viewpoints. Software Engineering Journal 11(1), 5–18 (1996)

[12] Leffingwell, D.: Calculating the Return on Investment From More Effective Requirements Managament. American Programmer 10(4), 13–16 (1997)

[13] Li, C., Ling, T.W.: OWL-Based Semantic Conflicts Detection and Resolution for Data Interoperability. In: ER (Workshops) 2004, pp. 266–277 (2004)

[14] Object Management Group, Object Constraint Language, Version 2.2, http://www.omg.org/spec/OCL/2.2/
[15] Luna, E.R., Garrigós, I., Grigera, J., Winckler, M.: Capture and Evolution of Web Requirements Using WebSpec. In: Benatallah, B., Casati, F., Kappel, G., Rossi, G. (eds.) ICWE 2010. LNCS, vol. 6189, pp. 173–188. Springer, Heidelberg (2010)
[16] Sommerville, I.: Software Engineering. Addisson Wesley (2002); Van Der Straeten, R., Mens, T., Simmonds, J., Jonckers, V.: Using Description Logic to Maintain Consistency Between UML Models. In: Stevens, P., Whittle, J., Booch, G. (eds.) UML 2003. LNCS, vol. 2863, pp. 326–340. Springer, Heidelberg (2003)
[17] Yang, D., Wang, Q., Li, M., Yang, Y., Ye, K., Du, J.: A survey on software cost estimation in the chinese software industry. In: ESEM 2008, pp. 253–262 (2008)