Managing Volatile Requirements in Web Applications

Matias Urbieta
LIFIA, Facultad de Informática, UNLP and CONICET
La Plata, Argentina
matias.urbieta@lifia.info.unlp.edu.ar

Damiano Distante
Unitelma Sapienza University
Rome, Italy
damiano.distante@unitelma.it

Gustavo Rossi
LIFIA, Facultad de Informática, UNLP and CONICET
La Plata, Argentina
gustavo@lifia.info.unlp.edu.ar

Wieland Schwinger
Department of Cooperative Information Systems,
Johannes Kepler University Linz
wieland.schwinger@jku.ac.at

***Abstract*— Web applications allow business to offer services or products to numerous users with different culture, context, and needs. There are situations where applications must adapt to unforeseen and temporary business requirements, such as a one-off market campaign to launch a new product, beta features for engaging users, or disaster solidarity features that remain in the application for a period of time. In this paper, we summarize an approach for dealing with this sort of volatile requirements and present challenges in the research field that must be addressed.**

Keywords—Web applications; volatile requirements, business process evolution, Web systems evolution

I. INTRODUCTION

Nowadays business must adapt to global trends in order to keep users engaged; unplanned marketing campaigns, season promotions (final season sales), crisis management, among others business requirements are examples of unexpected requirements that stress the whole applications' infrastructure. When unforeseen requirements arrive and remain in the application for a period of time until a given business or time event occurs, it is called Volatile Functionality.

Volatile functionalities demand business processes to change according to active market campaigns, catastrophes, etc. The system may modify the underlying workflow model; this may imply executing a slightly different workflow version which supports new requirements like discounts and free-shipping, or introduces new workflow steps like new forms to be filled, etc. In Web applications these changes may compromise several tiers of the applications, including content, behavior, navigation, and presentation. When the underlying workflow changes, user interfaces may, for example, introduce a new form that will demand a new view controller that orchestrates data validation and workflow navigation, and finally the business model must be modified to support new form entities and fields.

As a reference example, we use the checkout process in an e-commerce site; in order to buy some items, the user must follow a simple workflow comprising a number of steps such as selecting a product, choosing the type of wrapping (regular or special for birthday), selecting the shipping address, choosing the payment method, etc. Suppose an unforeseen event, such as a catastrophe, happens that leads to a donation campaign. We may require the introduction of a new donation step in the purchase workflow, where users can choose between different pre-set amounts of money to donate. This change will require at least a set of modifications:

(i) implement a page that presents a donation form with its corresponding fields;

(ii) add a corresponding step in the workflow and modify the workflow coherently;

(iii) add persistence machinery for the new data to be stored; and

(iv) upgrade navigation functionality, for example, to let users navigate to their donations.

In this case, the set of changes must be present only when the catastrophe campaign is active, otherwise they make no sense. In the mid-term we have a volatile requirement (the existing of a catastrophe and the donation campaign) which leads to a "context-aware" workflow behavior.

Regularly, e–commerce sites promote sales campaigns such as Back To School which is available for a period of time. In this case, the Back to School page which in turn points to products designed for scholars will be available to give quick access to scholar products. Some days before the summer holiday ends, customers can access to certain offers and, more specifically, can benefit from free "super-shipping" with certain constraints up to academic activities start. After that, these features are removed due to every New Year Back to School features defers from year before.

Additionally, the impact of the adaptation in the application may not be simple; that is, the introduction of this volatile requirement may cross other workflows such as ticket booking for a recital, product pre-order, etc. Therefore, the way in which the volatile requirements are modeled is critical to assure that they are correctly implemented.

Unfortunately, applications are not modeled to support this kind of situations and needs for adaptations usually arrive once the application has been already released.

When new requirements are unpredictable and temporary like volatile requirements [5], they are usually introduced in an

ad-hoc way. The inadequate implementation of the associated changes may lead to a decay of the software quality compromising application maintenance, stability, and complexity, and finally the application's budget.

In this paper we present the state of the art of our approach for handling volatile functionalities regarding business processes. This paper summaries our previous work [4][6][9][10] and introduces challenges that must be addressed for further work in this topic.

The rest of the paper is organized as follows: in Section 2 we present some background themes; in Section 3 we present our model-driven approach to welcome volatile requirements and implement corresponding volatile functionalities that impact on the different layers of a Web application; finally, we describe some related work in Section 4 and conclude the paper announcing feature work in Section 5.

II. BACKGROUND

In our approach to deal with volatile requirements in Web applications we use WebSpec [12] as the language for modeling workflow requirements and Pattern Specifications [3] as the technique for specifying the association between requirements belonging to different concerns.

A. *Abstract Data Views*

Abstract Data Views (ADVs) [11] allows describing user interface by means of an object-oriented model for interface objects. An ADV is defined for each node class to indicate how each node attribute or sub-node (if it is a composite node) will be presented to the user. An ADV can be seen as an Observer of the node expressing its perception properties, in general, as nested ADVs or primitive types (e.g. buttons). Using a configuration diagram we express how these properties relate with the node attributes and operations.

ADVs are also used to indicate how interaction will proceed and which interface effects take place as the result of user-generated events. These behavioral aspects are specified using ADV-charts, a kind of statecharts representing states and state transitions for a given ADV. ADV-charts are useful when we need to model rich interface behaviors such as that of Rich Internet Applications (RIA).

B. *WebSpec*

WebSpec [12] is a visual domain specific language for representing Web applications requirements; 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 (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).

In Figure 1, the checkout process in a Web application is depicted as a set of interactions where the user is able to select a product for start setting out its purchase (interaction Products); next she is able to choose whether a simple or gift wrap should be used; next, delivery information must be introduced such as address and city; and finally the list of current orders is shown.

WebSpec has a supporting tool with features that allow, in the early phases of requirement gathering, realizing simulations of application interaction against mock interfaces and generating independent Web tests for testing the final development result.

C. *Pattern specification*

Pattern Specifications (PSs) [3] is a technique for formalizing the reuse of models. Originally, the notation for PSs was presented using the Unified Modeling Language (UML) as a base, but in this work we will instead use the concept of patterns in the WebSpec realm. A PS describes a pattern of structure defined over the roles played by pattern participants. Role names are preceded by a vertical bar ("|") and a PS can be instantiated by assigning concrete elements to play these roles.

III. OUR APPROACH IN A NUTSHELL

In most mature Web design approaches, such as UWE, WebML, Hera, OOWS or OOHDM (see [7] for description and examples of each approach), a Web application is designed with an iterative process comprising at least conceptual and navigational modeling. According to the state-of-the-art of model-driven Web engineering techniques, most of these design methods produce an implementation-independent model that can be later mapped to different run time platforms. For the sake of clarity we will concentrate on the conceptual,

Fig. 1 Simple checkout process modeled using WebSpec

navigational and interface models as they are rather similar in different design approaches.

As most of the problems discussed so far apply to all development approaches, we will first describe the philosophy underlying our technical solutions in such a way that it can be reused; next, we will concentrate on the OOHDM design models and will briefly discuss how each part of our approach could be adapted to other methods. A detailed discussion on how to incorporate volatile functionalities in any other specific design method and/or model-driven framework is, of course, outside the scope of this paper.

Our approach is based on the idea that even the simplest volatile functionality (e.g., a video available for a period of time a website) should be considered as a first-class

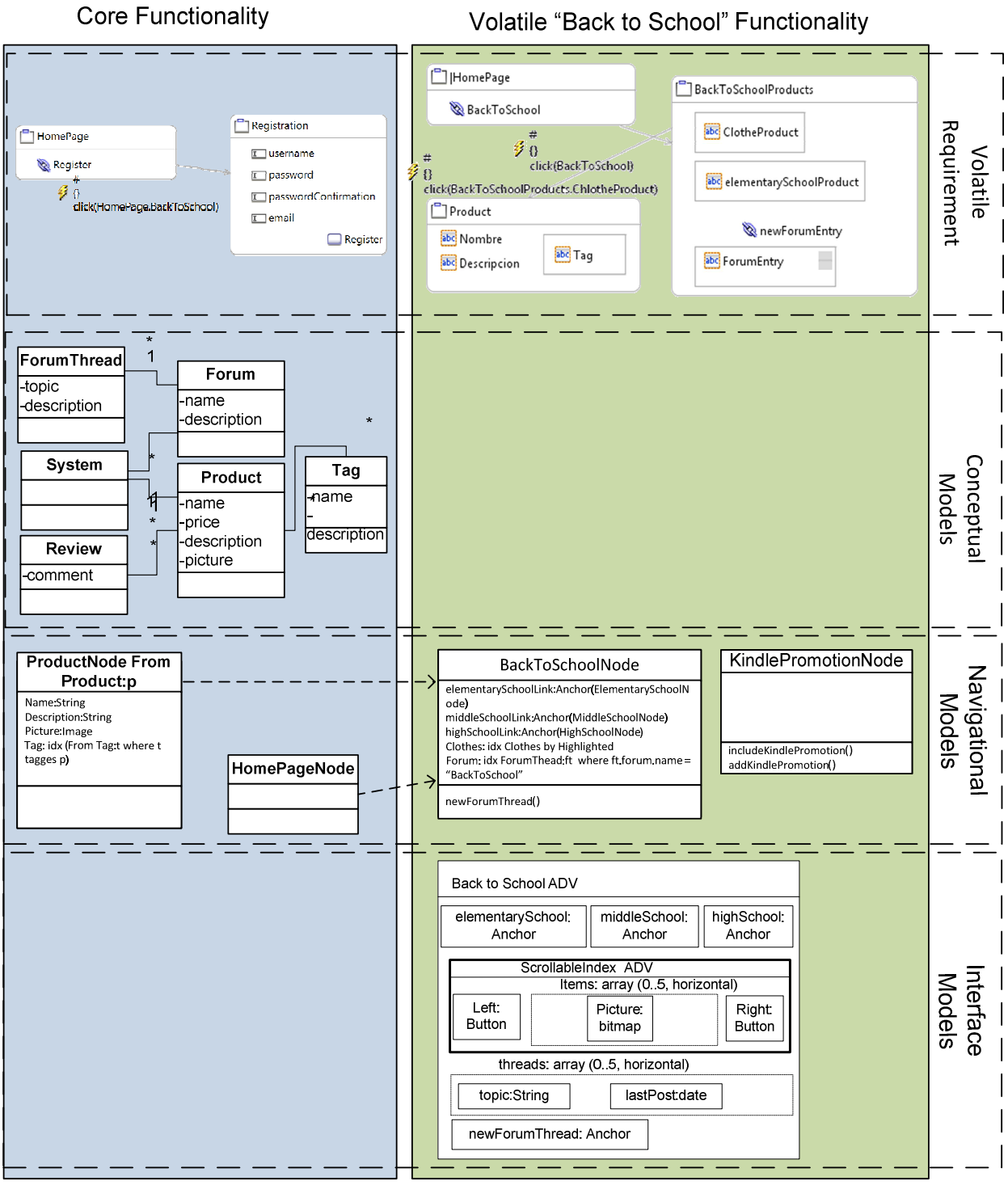

Fig. 2. Back to school core and volatile models

functionality and, as such, designed accordingly. At same time, their design and implementation have to be taken separated and as much as possible decoupled from that of core and stable functionalities.

Building on the above ideas, our approach can be summarized with the following design guidelines:

- We identify volatile functionalities in the early phase of requirement gathering and use WebSpec language to describe them. In Section III.A this step is described thoroughly.
- We decouple volatile from core functionalities by introducing a design layer for volatile functionalities (called Volatile Layer) which comprises a requirements model, a conceptual model, a navigational model, and an interface model.
- Volatile requirements are modeled using the same notation used to model core requirements (e.g., use cases, class diagrams, user interaction diagrams, etc.) and separately mapped onto the following models using the heuristics defined by the design approach. Notice that volatile requirements are not integrated into the core requirements model, therefore leaving their integration to further design activities.
- New behaviors, i.e. those which belong to the volatile functionality layer, are modeled as first class objects in the volatile conceptual model; they are considered as a combination of Commands and Decorators of the core classes. This strategy applies also to slight variants of business rules (such as adding a price discount to a product). In this case, the decoration is applied at the method level more than at the class level. Notice that this strategy can be applied to any object-oriented design method, i.e., any method using a UML-like specification approach. In methods based on data modeling constructs, such as WebML, adding new (volatile) information is straightforward given that a precise composition language for entity types is defined.
- Inversion of control is used to achieve obliviousness; i.e., instead of making core conceptual classes aware of their new features, the knowledge relationship is inverted. New classes know the base classes on top of which they are built. Core classes, therefore, have no knowledge about the additions. This also stands for aspect-oriented approaches.
- Nodes and links belonging to the volatile navigational model may or may not have links to the core navigational model. The core navigational model is also oblivious to the volatile navigational classes, i.e., there are no links or other references from the core to the volatile layer. This principle can be applied in any Web design approach.
- Separate integration specification is used to specify the connection between core and volatile nodes. As we show later in the paper, the integration is achieved at run time. In other model-driven approaches, the integration can be performed during model transformation by implementing the corresponding transformations.
- The interfaces corresponding to each concern (core and volatile) is designed (and implemented) separately; the interface design of the core classes (described in OOHDM using Abstract Data Views [11]) are oblivious with respect to the interface of volatile concerns. As in the navigational layer this principle is independent of the design approach.
- Core and volatile interfaces (at the ADV and implementation layers) are woven by executing an integration specification, which is realized using XSL transformations. Again, the idea of model weaving is generic and therefore the same result can be obtained using other technical solution.

In Fig. 2, we present both the Core and the Back to School volatile concerns showing, in different layers, the corresponding models. For the sake of simplicity, we have limited the core concern's models to those classes in the conceptual, navigational and interface diagrams which are "affected" by some Back to School feature.

A. *Workflow requirements modelling*

When a given application implements different workflows as nodes and links, the introduction of volatile functionalities may implicate ambiguity and inconsistencies with other requirements. In order to avoid such inconsistences, the approach supporting volatile functionalities helps identifying, modeling, and validating requirements. The approach is based on the idea that any volatile requirement must be treated as a first–class citizen; we consider these requirements as belonging to a separate concern [8] allowing us to isolate, model and later compose both core application workflow and volatile requirements. To make this presentation thorough, we first describe the general approach to model Web workflow requirements using WebSpec.

Step 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 [2].

Step 2: Requirement modeling. Web application requirements are formalized using a requirement Domain Specific Language (DSL). This formalization is essential during the requirement gathering process with stakeholders. Using a requirement DSL, tasks such as tests derivation and scenarios simulations can be automated easily. In this work, we selected WebSpec as the requirement DSL.

Step 3: Requirement generalizations modeling. Base workflow changes (e.g. adaptations) are modeled using the Pattern Specification extension for the requirement DSL; in this paper we exemplify with the WebSpec extension.

Step 4: Consistency validation. Syntactic and semantic analysis is performed over requirements. By means of an algebraic comparison of models, candidate structural and navigational conflicts are detected. These 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. Semantic equivalences between requirements are

detected for warning requirement analysts. For more information on this process see [9]. In order to check consistency of volatile functionalities, base requirements are composed with volatile requirements (following Pattern Specification semantics) giving as result a complete model that is validated. Since a given volatile requirement can be generic and so it can have several points of instantiation into the base diagram, the consistency validation procedure will only use specified binding configurations between base and adaptation model's elements.

Step 5: Test derivation. In this step, tests for the composition of the traditional WebSpec diagram and the WebSpec PS extension are generated producing tests that allow validating the final Web Application. Generated code is based on the Selenium tool which allows automating Web browsing task based on WebSpec requirements.

This also allows assessing the set of requirements with users by using simulations in the early stages of UI mocking. The same tests are used later in the testing phase of the software development process. When deriving tests for volatile functionalities, the same binding configurations used in Step 4 are taken into account. That is, the test derivation process uses an internal model where the base model was only enhanced on those points that specify a binding configuration.

For more details on the approach to address volatile requirements that involve adapting the workflows of the application see [7].

IV. CHALLENGES

We have presented an approach that provides support for volatile requirements during the whole development process of a Web application. Nonetheless, a number of challenges must be addressed in this field:

- *Models@Runtime*: In previous work that we have conducted on the topic of this paper, we have realized that introducing volatile functionalities in Web applications may be challenging when dealing with static underlying technology such as Java. Once the application is running it is not straightforward to introduce changes for a period of time. A trending research field called Models@Runtime [1] aims at providing guidelines, tools, and approaches for designing and implementing runtime adaptive applications. Introducing Models@Runtime concepts in software development process may help systems adapting to any future volatile functionality.
- *Agile development*: The presented model-driven approach is formal and well-structured but doesn't fit very well with agile approaches because this last relies on less structured tools for requirement gathering such as Mockups. Mockups have shown to be useful for effective requirement gathering. Although agile development reduces artifacts release cycle, applications are still prone to introduce volatile functionality. Tools for identifying and formalizing volatile functionalities in agile methods should be developed.
- *Volatile workflows*. Workflows are first-class citizen in Web applications because they address business goals. Volatile functionality may rise suddenly at any already defined workflow as it was depicted in the introduction section and may affect already stable and productive components. Conceptual tools for modeling volatile functionality that compromises workflow may ensure consistency in order to handle different versions (that run in parallel) of a given workflow: compromised instances of a given workflow, and not compromised ones.

Based on these themes, next we present a roadmap of further work.

V. FURTHER WORK

The proposed approach provides solutions to cover the whole life-cycle of volatile functionalities in Web applications, from design to implementation, deployment, and run-time state management (activation/deactivation, according to their volatility pattern). It addresses volatility at the different application layers it can impact, including conceptual, navigational and user interface, and makes it possible to seamlessly integrate and manage volatile functionalities without the need for any modification to the application's core (stable) components.

Our proposal makes it possible to introduce new volatile functionalities in Web applications "on the fly", and enables non-technical people to control their activation rules at runtime, thus providing business agility to the application.

We also pointed out different related topics that we are undertaking for integrating our approach in agile development, supporting business process as first-class citizen and rethinking applications for having Models@Runtime. First we are studying how to implement Volatile functionality in static typed language such as Java where changes in runtime are not simple.

We are also working on the integration of our approach in model-driven Web engineering methods other than OOHDM and particularly analyzing the integration at the meta-model level. By analyzing existing ideas to bridge and/or unify methods [5] we can find a way to express volatility in a higher abstraction level.

We plan to perform assessments to validate our ideas and measure benefits of its application exploiting WebSpec features such as test generations and simulations.

UML class diagrams and business process models can be sketched from WebSpec diagrams. Heuristics must be studied in order to produce accurate design models. Obtained UML and business process modes can be used also for producing prototype applications.

We plan to compare the outcome obtained from the requirement gathering tasks using our approach (based on Web requirement models) against traditional lexical software requirement specification. We also plan to analyze the advantages in traceability between model elements since our model for formalizing requirements also allows deriving design models following a model driven approach. Finally, we are studying how Web workflow requirements can ease agile development by inferring required story points [2] for a given requirement.

Finally, we are constantly analyzing and assessing Web applications in order to obtain additional feedback for our

conceptual framework related to this kind of Web application evolution.

ACKNOWLEDGMENT

The authors also want to thank Silvia Gordillo, Werner Retschitzegger, and Esteban Robles Luna who have helped reviewing and providing an interesting point of view on some aspects of our approach.

REFERENCES

[1] Bencomo, N., Bennaceur, A., Grace, P., Blair, G., Issarny, V. The role of models@run.time in supporting on-the-fly interoperability. In Computing, 95(3), pp. 167-190. Spinger-Verlag Wien (2013).

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

[3] France, R., Kim, D., Ghosh, S., Song, E. A UML-Based Pattern Specification Technique. In IEEE Transactions on Software Engineering, 30(3). IEEE Computer Sociaty (2004).

[4] Ginzburg, J., Distante, D., Rossi, G., Ubierta, M. "Oblivious Integration of Volatile Functionality in Web Application Interfaces", Journal of Web Engineering, Special Issue on Design of Sophisticated Web-based Systems, Vol. 8, No.1 pp. 25-47, Rinton Press. 2009.

[5] Moreira, A., Araújo, J., Whittle, J. Modeling Volatile Concerns as Aspects. In Proc. of the 18th International Conference on Advanced Information Systems Engineering (CAiSE 2006), Lecture Notes in Computer Science pp. 544-558, Springer (2006).

[6] Rossi, G., Ginzburg, J., Urbieta, M., Distante, D. "Transparent Interface Composition in Web Applications." 7th International Conference on Web Engineering (ICWE2007). Lecture Notes in Computer Science Volume 4607, 2007, pp 152-166, Springer (2007).

[7] Rossi, G., Pastor, O., Schwabe, D., Olsina, L., Web Engineering: Modelling and Implementing Web Applications. Human-Computer Interaction Series. Springer, London (2008).

[8] Sutton, S., Rouvellou, I. Modeling of Software Concerns in Cosmos. In Proc. of the 1st ACM international conference on Aspect-oriented software development (AOSD 2002), pp. 127-133, ACM Press (2002).

[9] Urbieta, M., Retschitzegger, W., Rossi, G., Schwinger, W., Gordillo, S.E., Roble Luna, E. Modelling adaptations requirements in web workflows. In Proc. of the 14th Int. Conference on Information Integration and Web-based Applications & Services (iiWAS 2012): 72-81, ACM Press (2012).

[10] Urbieta, M., Rossi, G., Distante, D., Ginzburg, J. Modeling, Deploying, and Controlling Volatile Functionalities in Web Applications. International Journal of Software Engineering and Knowledge Engineering 22(1), pp. 129-155, World Scientific Publishing Co. (2012).

[11] Urbieta, M., Rossi, G., Ginzburg, J., and Schwabe, D. Designing the Interface of Rich Internet Applications, in Proc. of LA-WEB 07, Chile, IEEE Press, 2007.

[12] WebSpec Language, http://code.google.com/p/webspec-language/. Access August 2012.