Current Research on the Design of Web 2.0 Applications Based on Model-Driven Approaches

Alessandro Bozzon, Sara Comai, Piero Fraternali, Massimo Tisi
Dipartimento di Elettronica ed Informazione, Politecnico di Milano, Milano – Italy
{bozzon, comai, fraterna, tisi}@elet.polimi.it

Abstract

This paper presents our research activities on the design of Web 2.0 applications currently ongoing at Politecnico di Milano. Our approaches are based on model-driven development *techniques: in particular, they extend Web 1.0 models to cope with the technological characteristics of the new applications and exploit design patterns to capture the desired behaviors. Two main research activities are presented, considering two orthogonal issues of Web 2.0 applications: the social and the technological aspects. Moreover, other ongoing research directions are outlined.*

1 Introduction

The advent of the so-called Web 2.0 has shifted the focus of Web application development towards a more prominent role of the *end-users*, now considered as the critical success factor. Among the many facets of Web 2.0 in our research at Politecnico di Milano we have considered both the social and the technological aspects.

From a social point of view, we are witnessing a shift towards collaborative and community-based applications. Community-driven applications can be defined as Web applications targeted to a set of users (the community), purposely designed for encouraging the social interaction of community members: content production and dissemination, content processing (rating, categorization, transformation), and inter-user relationship development. The success of Web 2.0 community-based applications depends on several factors, some of which are immaterial: quality of content, cohesion of interests, suitable mechanism for emergence of outstanding contributions and contributors, which add up to the well-known "traditional" criteria of usability of the interface and global quality of the user's experience. Many tools and algorithms have been developed for addressing specific topics related to social network analysis [13] and several frameworks, patterns and models have been proposed to address particular issues of a social application design (e.g., [2, 4, 1]).

From a technological point of view richer interfaces are needed to improve user experience. Indeed, traditional HTML interfaces are showing their limits, compared to desktop applications, both in terms of content presentation and manipulation (HTML was designed for documents, not GUIs, and multimedia support is limited) as well as in terms of interaction (server-side computation implies full page refresh at each user-generated event). Web 2.0 applications demand a novel development paradigm [12], to overcome such limitations: the technological answer is represented by Rich Internet Applications (RIAs) [7, 6]. RIAs extend traditional Web architectures by allowing computation to be reliably partitioned between the client and the server; they are an essential ingredient of the Web 2.0, because they blend the best of Web-enabled and desktop architectures and address core Web 2.0 requirements, like real-time collaboration among users, sophisticated presentation and manipulation of multimedia content, and flexible human-machine interaction (synchronous and asynchronous, connected and disconnected). As RIA adoption is growing, a multitude of programming frameworks have been proposed to ease their development (e.g., Flex, OpenLaszlo, Google Gears, Silverlight, AIR, Java Web Start, JavaFX, just to name a few). These increase productivity, but are bound to a specific technology and therefore not easily portable across different platforms.

In this paper we show how conceptual and platform-independent models can be used in the design and development of Web 2.0 applications (considering both social and technological aspects), in the tradition of *Model Driven Development* (MDD). Compared to several frameworks addressing social issues and supporting specific RIA platforms, our approach is technology-neutral and can be automatically converted into implementations in all the most popular RIA technologies and frameworks.

The envisioned development paradigm is based on the *extension of MDD approaches conceived for traditional Web 1.0* applications and on the usage of *design patterns*.

For convenience, we will refer to the WebML [14] notation, but the considerations discussed in this paper apply to other Web engineering notations and methodologies that allow the specification of the interface composition and navigation.

The paper is organized as follows: Section 2 explains how social aspects have been integrated in our methodology. Section 3 gives an overview on the extensions of the WebML model to capture the technological features. Section 4 briefly describes other current research directions related to Web 2.0. Finally, Section 5 presents future trends and draws the conclusions.

2 Capturing Web 2.0 Social Features in WebML

Our research on Web 2.0 social features aims to integrate the social perspective typical of emerging Web 2.0 applications within a structured Web Engineering approach, based on model-driven development. The result is a seamless development paradigm that is capable of capturing both conventional Web development issues (data storage, publication, management, Web service publication and invocation, user profile management, etc.) and the essential design patterns that appear in state of the art social Web 2.0 applications. Our current results comprise a set of classified model-driven design patterns for community-based Web applications, validated by means of an analysis of pattern occurrence in top-ranking Web 2.0 social applications and the extension of well-known Web development processes (i.e., RUP [9] or WebML [14]) to incorporate activities specific to community-driven development.

To identify best practices in social Web applications we analyzed ten of the most popular Web 2.0 community applications and distilled a number of recurring *design patterns*. The analysis of the pattern set lead to the individuation of nine core concepts that are the main focus of the social activities in Web 2.0 applications. Table 1 shows the design patterns grouped by their underlying social concept. Patterns are further distinguished into front-end patterns and back-end patterns: front-end patterns relate to the interface for the community members to express their activity, back-end patterns reflect the system responses to member-generated interaction.

The complete list of social features and relative patterns can be found in [10]. As a simple example, Item Clustering is the feature that allows to group in a Container Element items that share some common properties. Depending on the type of items, the Item Clustering feature is usually implemented allowing the user to create and manage photo sets, content hierarchies, playlists, and so on. Item Clustering answers a twofold purpose. First of all, it provides a structured access to content items: the browsing of shared content and the individuation of interesting items can be eased by providing the user with a group hierarchy. Secondarily, Item Clustering can be used by the community members, like any metadata, as further information to understand the properties of a given item. While traditional Web applications usually provide users with a standard classification (e.g., a hierarchy) for content items, most social application prefer to implement a collaborative approach to Item Clustering.

Concept	Front-end Design Pattern	Back-end Design Pattern
Item Clustering	Organization Browse by Tag	-
User Clustering	Group Creation Group Participation	-
Item Relevance	Rating Flagging	Rel. Adjustment
User Reputation	Social Visualization	Rep. Adjustment
Connections	Relationship Setting Browse by Connection	-
Scoring	-	Payment Reward
Communication	Talk Recommendation Invitation	Notification
Permissions	Permission Setting	Permission Check
Interoperability	Exportation	Syndication

Table 1. Community-driven design patterns.

A general solution for collaborative Item Clustering is the Organization front-end pattern [10]. The pattern is proposed together with several variants that describe different implementation mechanisms. One of the more common variants of the Organization pattern is the content tagging variant. A tagging mechanism gives to the users the possibility to associate freely chosen words with the shared items and can be considered as an instance of the Organization pattern where any user determines the inclusion of each item into an implicit container associated with the tag. In general, tag containers are public and they can be freely created by any community member. Figure 1 shows the Organization pattern variant based on content tagging, expressed using the WebML notation.

Design patterns are not applied in isolation, but within the framework of analysis and design activities forming the development process of a certain class of artifacts. Several process schemes have been tailored to Web applications, starting from the more general notion of software life-cycle model. We extended the WebML process [14] to cope with community-based Model-Driven development. In essence, the focus on community features affects both the requirements analysis phase, in which ad hoc functional requirements stem from the goal of fostering community life, and design, where the data model must reflect the members pro-

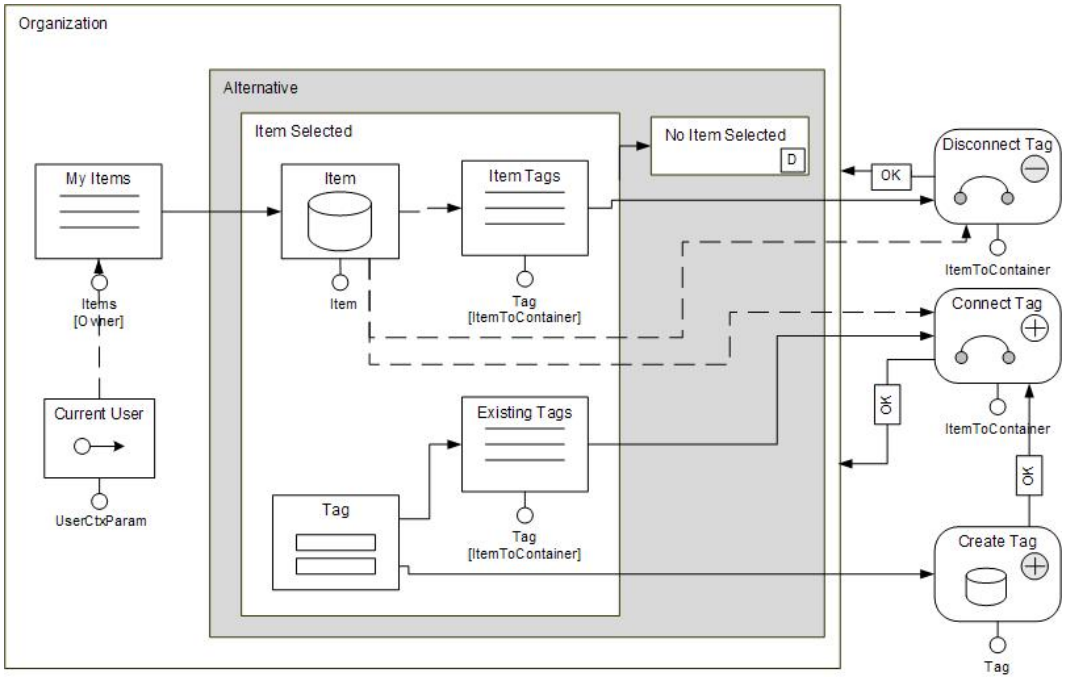

Figure 1. Organization design pattern (Tagging variant).

file meta-data and the application design must incorporate the appropriate community patterns in the front-end and in the back-end.

In the analysis of requirements of social Web applications, community governance and social processes emerge with a prominent role. Communication experts should identify the social processes needed to foster user participation: which user activities are critical for the community, which strategies are due for community monitoring (e.g., moderation, codes of conduct), and what reward and reputation mechanism to install; social network experts, instead, define the abstract models that represent member relationships and meaningful indicators for monitoring them, which will allow community administrators to trace the community's trends and govern its evolution. Furthermore, the analysis of requirements about users is expanded, to address multiple perspectives:

- Content management: as in conventional Web applications, functional roles must be identified, e.g., by classifying users into stakeholders, administrators, editors, etc. The question is "who can read/update what?".
- Governance: social roles must be elicited. The question is: "who is controlled by whom?". Different governance models may require alternative social roles: democratic moderation relies on a shared code of conduct whereby every member can monitor others and draw attention to violations. In such a system, the users' hierarchy consists simply of moderators and contributors. Alternatively, in more structured communities, the governance model could reflect some existing formal organization (e.g., a company's hierarchy).
- Social behavior: another classification of users can be obtained by the observation of their participation: behaviors such as pioneers, killers, lurkers can be defined and help monitor the community's status and plan reinforcement or corrective activities.

Besides users' roles, the analyst should pinpoint the relationships that members can set-up (e.g., group creation, friend-of-a-friend linking, etc) and the degree of collaboration they can establish (e.g., application sharing, invitation, etc). Once user roles are identified under all the relevant perspectives and the allowed relationships are determined, roles can be mapped into user types and associated to the activities that each type is entitled to perform, including relationship-setting activities.

In the design of Web applications, several tasks are affected by a community-driven focus: 1) the design of the *data model* of the application should encompass the meta-data needed to reflect the members roles, relationships, profile data, and reward policies; 2) the design of the *hypertext front-end* should integrate the selection of the navigation, contribution and social interaction pattern; 3) the design of the *hypertext back-end* should comprise the back-end de-

sign patterns needed to support the supported social community governance and processes.

Once the main design patterns are identified, the model-driven design of the Web application is guided by the pattern instantiation process and by the gradual integration of the pattern instances. When WebML is used as the modeling language, the WebML models for the front-end and back-end can be directly obtained by instantiation and composition of the proposed patterns.

As an example, the instantiation procedure of the Organization pattern follows these subsequent steps:

1. The units specified in the pattern and their connections are first instantiated in the page (e.g., into a page called My Videos). The instantiation requires the use of a particular entity (e.g. a Video entity) to perform the Item role in the pattern. The role instantiation is usually reflected by the data model where the role entity and instantiated entity are both included in the model, linked by the means of a specialization relationship (e.g., the Video entity is a specialization of an abstract Item entity).

2. The pattern is enriched by selecting the suitable data elements to show (e.g. the information attributes of the current video in the Video DataUnit, typically Title, Description and the Video itself).

3. The pattern is integrated with other patterns and model elements to obtain the desired functionalities. For example, by design-time decision, the upload functionality can be added to the My Videos page eventually reusing suitable units of the pattern. Finally the page is integrated with the rest of the application with incoming and outgoing hypertext links.

The last phase is particularly delicate, as it can involve a high level of complexity, especially when the design requires merging several patterns in the same hypertext page. The definition of a formal procedure to address the WebML pattern composition issues is one of our current research topics.

3 Capturing RIA Technological Features in WebML

Our works on the technological aspects of Web2.0 applications focused on the definition of a conceptual model that supports RIA application design, by abstracting from specific implementation technologies [3]; this model captures the essential features offered by RIAs such as: distribution of computation and logic across client and server, temporal and persistent data storage at the client-side, and asynchronous, possibly bidirectional client-server communication. The proposed RIA model extends the WebML notation conceived for traditional data-intensive applications: the same notation can be used both for Web 1.0 and Web 2.0 applications.

A RIA application can be described by its structure and behavior. The former comprises a *data model*, which specifies the content objects underlying the applications, and an *interface model*, which describes the front-end exposed to the user. The latter is represented by a *dynamic model* that describes what happens when the user or other sources of events interact with the application.

Data distribution among client and server can be reflected by refining the data model in order to provide a designer with entities and relationships to represent distributed query execution: in RIAs, contents reside both on the server and in the clients, in the form of main memory objects associated with the server/client applications both in terms of visibility and duration; persistent storage is provided, on the server, by means of standard mechanism (e.g., database, file systems etc.), while, on the client, some technologies also allow the permanent storage of data for local (or offline) application access. Data of RIA applications are therefore characterized by two different dimensions: (1) the architectural tier of existence, which can be the server or the client, and (2) the level of persistence, which can be permanent or temporary. Figure 2 depicts an example of a RIA data model: it represents a simplified version of a multimedia, collaborative platform for the publication, sharing and discovery of contents produced by user's communities. Graphically, we mark client entities and relationships with a "C" icon, and server elements with a "S" symbol; a filled icon denotes temporary persistence and a non-filled icon permanent persistence.

The entities/relationships on the left-hand side represent data persistently stored on the server, like in traditional Web applications. Content items are the main interaction objects for the application: the *MMItem* entity represents content published on the platform (e.g., photos, videos or audio tracks), while the *Member* entity represents all application users. A self-relationship connects each user with his friend list; members manage one or more *Playlist*, each one aggregating a set of content items. A *MMItem* belongs to a given member and it is associated to one or more *Tag* and *Comment* instances, created by registered users. Data about content items, playlists, comments and tags are *persistently* stored also on the *client*, to allow *disconnected* usage of the application: while off-line, the user can process new items to upload on the platform, update his playlists, and manage associated tags and comments. Reconciliation with the server can be triggered when he goes online.

As data model elements are partitioned considering the architectural tier of existence, a similar approach can be

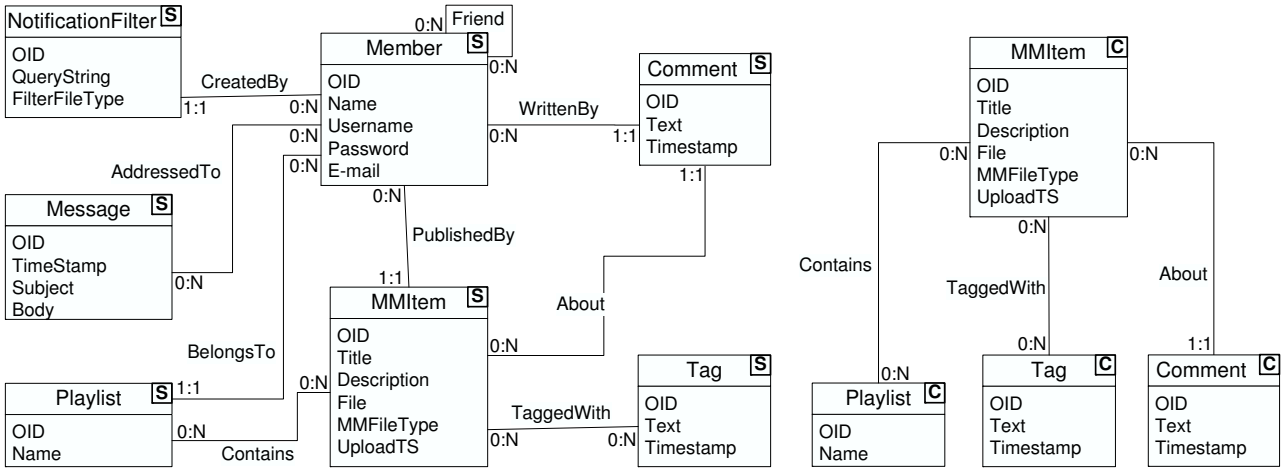

Figure 2. Data model with permanent client and server Entities/Relationships.

used to mark *interface model* elements in order to distribute computation competencies across client and server. In particular, the designer should be able to specify how the computation of the page and of its content is distributed between the client and the server, how distributed data are managed (to minimize data transmissions), how and when replicated data are synchronized, etc. We therefore distinguish three different aspects of interface modeling, each one related to a specific feature to represent: structural composition, content publication, and content management.

From the technological standpoint, RIAs have a different physical *structure* than traditional Web 1.0 applications: the latter consist of multiple independent templates, processed by the server and simply rendered by the client. RIAs, instead, typically consist of a single application "client-container" (e.g., a Java applet or a FLASH movie), which loads different data and components based on the user's interaction, Moreover, the structure of the interface consists of a topmost page (eventually contained into a traditional, server-computed HTML page) partitioned into peer-level sub-pages, independently calculated and rendered by the client, possibly in collaboration with the server. As a consequence, we design the structural composition of RIAs by distinguish between two types of pages, classified into *server* pages (marked with a circled "S", and representing traditional Web 1.0 pages, where content and presentation are calculated by the server, whereas rendering and event detection are handled by the client) and *client* pages (marked with a circled "C", and representing pages incorporating content or logics managed – at least in part – by the client). To reflect the complex, single-application shell structure of RIA applications, client pages can contain other client sub-pages.

For each page (or sub-page) the interface model specifies the data to be shown, the available interaction mechanisms, and the operations that may be triggered by the user using the provided interaction mechanisms. To this aim, we refined all the WebML concepts related to *content publication* and *management* with the explicit specification of distribution between the server and the client: content and operation units, selectors, and ordering clauses can be defined either as server or as client, with some constraints on the possible combinations. Units contained in a server page are computed by the server and are defined as *server units*, while units contained in a client page are computed by the client (possibly invoking the server) and are defined as *client units*. For a client unit it is possible to: 1) publish or manipulate content locally stored at the client-side or (by invoking the server) at the server-side (i.e., the reference entity/relationship of the unit can be either a server or a client one, persistent or temporary, as seen in the data model); 2) have client-side selector conditions and/or server-side selector conditions; the former are computed locally at the client, whereas the latter are executed at the server-side. 3) have client-side or server-side ordering clauses; the former are computed locally at the client, whereas the latter are executed at the server-side. Instead, server units are entirely computed by the server and therefore cannot use client-side entity/relationships and cannot comprise client-side selectors and ordering clauses.

As in Web 1.0 applications all the computations performed by the server must rely only on data and operations computable at the server side to cope with the asymmetric nature of the Web (where the client calls the server and not vice versa), in RIAs, instead, Web applications are allowed to manage bidirectional communications in order to provide advanced features like distributed event management. Previous works [8] explored the implications and possibilities stemming from such capabilities but, due to space reasons, we invite the reader to refer to the original paper.

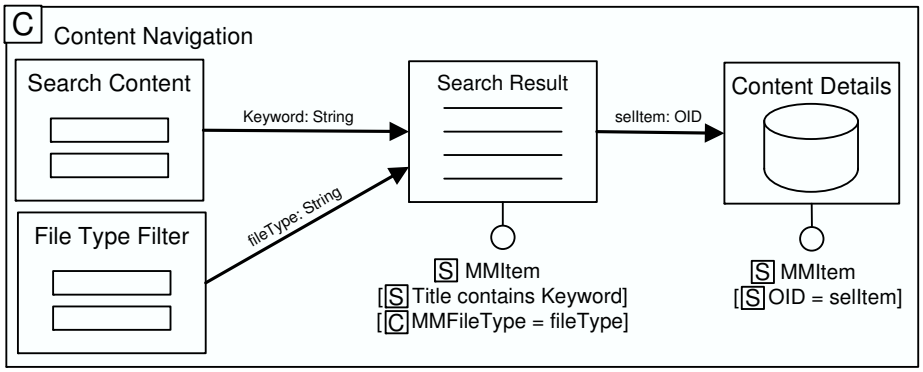

Figure 3. Example of RIA interface model.

Figure 3 shows an examples of RIAs interface models based on the data model for the multimedia, collaborative platform of Figure 2. The *Content Navigation* client page contains two entry units: *Search Content* accepts a keyword to be matched in the item's title, while *File Type Filter* accepts its file type (e.g., audio, video). Both entry units are linked to the *Search Result* index unit, defined over the server entity *MMItem* and provide the parameters to be used in the parametric conditions of the index unit (e.g., parameter `Keyword:string` is used by the selector condition `[Title contains Keyword]`). Notice that the computation of the page is managed by the client: however, being the *Search Result* defined over a *server* entity, the data of the searched items are retrieved from the database server. The query is then refined by the client, through the application of the second, client selector condition, which filters the retrieved instance without the need for a further server request. The selection of a result from the list triggers the calculation of the *Content Details* data unit, which retrieves the data about the selected item from the server.

This example shows how the computation of the selectors of the client index unit can be partitioned between the server to comply with the trade-off between efficiency and usability of the designed functionality: server-side selectors allow reducing the data to be transmitted to the client, client-side selectors avoid to invoke the server for their computation. Similar considerations apply also to the partitioning of ordering clauses.

Further examples combining units defined over client and server entities/relationships to exploit the client storage capacities are available in [3].

While the *data* and *interface* model allow to represent how content and data management can be distributed between the client and the server, the *dynamic model* explains what happens upon the interaction of the user or, possibly, of other events (like, for example, Web service calls). Rich Internet applications offer a very flexible runtime behavior: they allow one to selectively (re)compute or refresh only a portion of the interface of the application and to maintain unchanged all the pieces of information that are not affected by the interaction, so that unneeded re-computation can be avoided. Dually, the interaction may cause some pieces of content, which were previously displayed, to be deactivated or invalidated because they are no longer consistent with the rest of the page. The interface model presented in the previous subsection has been extended in [5] with the specification of the behavior required to support the possible effects of user's interaction, expressed as computation sequences activated in response to interaction events. The proposed dynamic model associates each occurrence of interaction with the application (e.g., link navigation, Web service calls, temporal events, and data-driven events like data updates) with an ordered sequence of operators that allow to (re)compute or invalidate the content and the parameters (used in parametric selectors) of the units of the application.

4 Other Research Directions

Currently, we are working also on other research projects, which can benefit from the Web 2.0 features.

Within the European PHAROS project[1] we are developing an audio-visual Web search portal, permitting users to pose advanced queries to multi-media materials, access results of queries using multi-modal and multi-channel interfaces, and personalize the search experience by saving queries in a personal profile, so that they can be exploited for asynchronous notification of new relevant audiovisual information. This ongoing research demonstrates that the model-driven approach can help the design of such a complex application and the generation of code of sophisticated Rich Internet Application front-ends, typical of the multimedia portals of the future.

[1] Platform for searcHing of Audiovisual Resources across Online Spaces – http://www.pharos-audiovisual-search.eu/

We are addressing also the problem of defining a Web design process suitable for the *multi-cultural* audience of a Web application in the globalization era [11]. Cultural markers are defined as Web design elements and patterns that meet the preferences of a specific cultural group, due to some influential cultural factor. We are studying a methodology for identifying cultural markers, validating them by user testing and turning them into design guidelines.

5 Future Trends and Conclusions

In this paper we have presented some research directions ongoing at Politecnico di Milano on Web 2.0 applications, mainly focussing on the social and the technological aspects of Web 2.0. The two aspects are *orthogonal* and in the design of an application including both of them, they can be considered in different steps: for example, first the design of the application including the patterns for the social aspects can be defined, then it can be refined with the extensions proposed for implementing the application in a RIA platform.

We believe that in the future social Web applications will be more and more integrated in conventional Web applications: we foresee that in the next years companies will start integrating also social features in their B2C portals, organizations will foster user generated content and peer to peer interaction in their Intranets, and so on. As far as the technological aspects are concerned, the current trend is already showing an increasing supply of new tools and several new Web applications are adopting the new technologies. However, the focus of the available tools is on the implementation of the applications for a specific framework/platform.

We believe that the *integration of traditional model-driven Web Engineering methods* supporting conventional Web development issues (data storage, publication, management, Web service publication and invocation, user profile management, etc.) *with RIAs and social Web 2.0 characteristics* is an essential factor for supporting the development of well-crafted, maintainable, usable, and socially effective Web 2.0 applications.

References

[1] Welie Interaction Pattern Library. http://www.welie.com/patterns/.

[2] J. Bishop. Increasing participation in online communities: A framework for human-computer interaction. *Comput. Hum. Behav.*, 23(4):1881–1893, 2007.

[3] A. Bozzon, S. Comai, P. Fraternali, and G. Toffetti Carughi. Conceptual modeling and code generation for Rich Internet Applications. In *ICWE*, pages 353–360, 2006.

[4] R. Cheng and J. Vassileva. User- and community-adaptive rewards mechanism for sustainable online community. pages 332–336. 2005.

[5] S. Comai and G. T. Carughi. A behavioral model for rich internet applications. In *ICWE 2007*, 2007.

[6] M. Driver, R. Valdes, and G. Phifer. Rich Internet Applications Are the Next Evolution of the Web. Technical report, Gartner, May 2005.

[7] J. Duhl. White paper: Rich Internet Applications. Technical report, IDC, November 2003.

[8] G. Toffetti Carughi et al. Modeling distributed events in data-intensive rich internet applications. In *WISE 2007*, pages 593–602, 2007.

[9] I. Jacobson, G. Booch, and J. Rumbaugh. *The unified software development process*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[10] P. Fraternali and M. Tisi. *Building community-based Web applications with a Model-Driven approach and design patterns [Submitted]*. IGI Global, 2008.

[11] P. Fraternali and M. Tisi. Identifying Cultural Markers for Web Application Design Targeted to a Multi-Cultural Audience. In *ICWE2008*, 2008.

[12] Preciado, J.C. et al. Necessity of methodologies to model Rich Internet Applications. In *WSE 2005*, pages 7–13, September 2005.

[13] J. Preece. *Online Communities: Designing Usability and Supporting Socialbilty*. John Wiley & Sons, Inc., New York, NY, USA, 2000.

[14] S. Ceri et al. *Designing Data-Intensive Web Applications*. Morgan Kaufmann Publishers Inc., 2002.