BUILDING COMMUNITY-BASED WEB APPLICATIONS WITH A MODEL-DRIVEN APPROACH AND DESIGN PATTERNS

Piero Fraternali, Massimo Tisi, Matteo Silva, Lorenzo Frattini

Politecnico di Milano

Abstract

This chapter addresses the design of community-based Web applications using the model-driven approach and design patterns. A set of best practices, identified by reviewing a number of top-rank Web 2.0 Web applications is illustrated and turned into a set of design patterns, expressed with a Model-Driven approach. A suitable development process, with specific focus on community-based application design, is defined, as an extension of traditional Web Engineering approaches. The defined patterns and process tasks are then implemented in a commercial Web development tool suite, enabling the visual specification and rapid generation of code for social Web 2.0 applications.

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. The goal is that of transforming end-users from passive recipients of content and communication into active contributors, by fostering their will to produce content, evaluate it, and interact with other users. This idea is not new, but reflects the long-standing concept of virtual community (Rheingold, 1993 [10]), now revitalized by the pervasive reach of the Web. Community-driven Web applications are the technical means for achieving Web 2.0 successful interaction. They 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.

As for any vertical application, 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. However, by looking at the most popular social applications (e.g., Del.icio.us, Facebook, LinkedIn), one easily recognizes a set of best practices that seem to be the backbone of successful Web 2.0 applications.

The goal of this chapter is to identify such best practices in the development of community-based Web applications, and capture them into a design paradigm that can be consistently applied to develop well-crafted Web 2.0 social applications. The envisioned development paradigm is based on two fundamental principles of modern Web engineering: the Model-Driven approach to application development and the usage of design patterns. The original contribution of the chapter comprises: 1) 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; 2) the extension of well-known Web development processes (e.g., RUP [1] or WebML [2]) to incorporate activities specific to community-driven development; 3) the implementation of the identified patterns in a commercial tool for model-driven Web applications, so to support the visual specification of community features and the automatic generation of code; 4) the evaluation of the proposed approach in the implementation of a demonstrative application.
The rest of the chapter is organized as follows: Section 2 introduces a running example; Section 3 gives the background on virtual communities and Model-Driven Web development; Section 4 presents the design patterns distilled by a review of popular Web 2.0 social applications; Section 5 contextualizes the usage of such patterns within a development process scheme tailored to community-based Web applications; Section 5.1 and 5.2 respectively zoom into the requirement analysis and design activity, showing the influence of community-related concerns; they revisit the running example, briefly highlighting a few instances of the illustrated design patterns and practices; Section 6 discusses possible future trends and, finally, Section 7 draws the conclusions.

2. Running Example

The chapter exploits a running example consisting of a simple video sharing application called MiniYouTube. The example, obviously inspired to the well-known video sharing community and similar systems, presents the typical features of a Web 2.0 application for the creation, sharing and evalutation of User Generated Content.
The features of this kind of application typically allow users to:

- publish new videos;
- classify videos and browse them by means of the social tagging mechanism;
- establish friendship relationships with other users and navigate the user base by friendship;
- evaluate videos proposed by others through a rating system;
- post messages to the profile page of other users;
- recommend videos to other users.

The next sections will show that a simple Web 2.0 social application can be easily modeled by composing and enriching a set of basic design patterns.

3. Background

The addressed topic is at the intersection of two main research areas: Model Driven Web development and Social Web application development. In the following sections, we briefly sketch the background of both fields.

From Virtual Communities to Social Web 2.0 applications

Many tools and algorithms have been developed for addressing specific topics related to social network analysis. Significant contributions have been made by Preece [3], showing how a community should have to be developed under a social-behavioral approach.
Preece characterizes virtual communities as composed of:

- *People*: actors of social interactions, they perform tasks and play roles.
- *Shared purpose*: the focus of social interactions; it could be a matter of interest, a job or a service.
- *Policies*: the rituals, protocols, laws. etc. that govern member interaction.

- *Social software*: the infrastructure that mediates communication, to let people share tasks and “feel together".

Community-support applications revolve around a common set of concepts, independent of the community's purpose:

- *Members*. Members act, communicate and set relationships, changing the state of the application by their behavior. By acting within the community each member builds up her/his own reputation.
- *Items*. Generic elements of interest. A community may base its existence on the mere sharing of items (e.g., bookmarks).
- *Activities*. An action relevant for the development of the community. Some activities could have a positive impact: e.g., item contributions, invitations, tagging, and rating. Other ones may hinder the community or dispel the interest of members (e.g., spamming).
- *Messages*. Unlike items, messages do not add value to the community directly, but support relationship development.

The activity of members can be described according to the Pay-Perform-Reward (PPR) behavioral model, which consists of the following steps.

- *Permission check*. Before a member starts an activity, his rights to act can be evaluated, e.g., based on privacy, etiquette, relationships, ownership, etc.
- *Affordability check & payment*. Some activities may be limited, e.g., in the number of times or frequency of performance. Limitation can be represented by means of an abstract notion of *credit* charged prior to activity execution.
- *Activity performance*. The actual execution of an action.
- *Evaluation*. An activity may yield added value for the community and therefore its utility must be evaluated, either explicitly, e.g., with a scoring system, or implicitly, e.g., on the basis the further activity it generates.
- *Reward*. To foster participation, proper rewards are given to users' contributions. Reward can range from fixed credit's updates to more complex rewards taking into account member's reputation, evaluator's reputation, and so on.
- *Reputation adjustment*. Finally, the community system can adjust the performer's reputation by evaluating the impact of his actions.

PPR is at the base of some patterns, in the following called *back-end patterns*, which describe the workflows that the system needs to activate in response to user's activity.

Many researchers have proposed frameworks, patterns and models to address particular issues of a social application design. Bishop [7] and Cheng, Vassileva [8] show how to foster members' participation proposing specific human-computer interaction patterns. Other notable works in this area are [11][12][14][15][16][17]. Mao et al. [13] and Chin and Chignell [18] study modelling approaches to social network analysis. Lee et al. [19] show patterns for recommendation systems. Furthermore there are many contributions to this research coming from the HCI field: tools for online communities have quite consolidated interaction patterns, as accurately shown by Welie [9].

Model-Driven Web Development with WebML

WebML is a Domain Specific Language, born for specifying at the conceptual level a Web application publishing or manipulating data. Content is modeled using Entity-Relationship (E-R) or UML class diagrams. Upon the same data model, it is possible to define different hypertexts (called *site views*), targeted to different types of users or to different access devices. A site view is a graph of *pages*, possibly clustered into *areas* dealing with an

homogeneous subject (e.g., the Facebook application comprises such areas as Profile, Friends, Networks, and Inbox) and hierarchically organized into sub-pages. Pages comprise *content units*, representing components for content publishing: the content displayed in a unit typically comes from an *entity* of the data model, and can be determined by means of a *selector*, which is a logical condition filtering the entity instances to be published. Instances selected for display can be sorted according to *ordering clauses*. Units are connected to each other through *links*, which carry parameters and allow the user to navigate the hypertext. WebML also allows specifying *operations* implementing arbitrary business logic; in particular, a set of data update operations is predefined, whereby one can create/delete/modify the instances of an entity, and create or delete the instances of a relationship. Examples of WebML design patterns will be provided and explained throughout the chapter. A complete illustration of the language can be found in [2]. We stress that we have adopted WebML as an illustrative language only for convenience[1]. Any comparable visual notation for expressing design patterns (e.g., UML[6]) would have served the same purpose.

Design Patterns

A *design pattern* [4] describes a recurrent problem in software design and a general reusable core solution to it. A pattern can be applied to several specific cases by detailing and enriching the provided core solution in a process that is called *pattern instantiation*. The pattern is usually presented by the means of a visual modelling language (e.g., UML[5]) together with possible variants and a textual specification in order to express relevant semantic aspects of the pattern. The specification includes the pattern name and aliases, a description of the addressed task, the relationship with other patterns and some guidelines about the pattern usage.

A WebML hypertext design pattern is a WebML model that specifies a set of core hypertext elements and their organization in order to fulfil a defined task. To instantiate a pattern inside a WebML model, some requirements have to be fulfilled. The rules for the application of the pattern are divided into two areas: data-level instantiation and hypertext-level instantiation.

Data-level instantiation rules require considering the pattern elements related to data specification (e.g., entities, attributes, relationships) as *roles* that have to be performed by some real application data element. For example, the *Item* role is required by several patterns and is performed in our running case by the *Video* entity.

The second category, hypertext pattern instantiation rules, enforce some formal requirements when instantiating the pattern:

- all the units (content units and operation units) that are specified in the pattern have to be present in the pattern instance;
- the pattern instance can be enriched by inserting new units anywhere in the model, even breaking links in sequences of units;
- the activation semantic of a link in the pattern have to be kept unchanged in the pattern instance, even when the link has been broken in a sequence of units (e.g., if the link in the design pattern requires an explicit click by the user to be activated, the same should be true for the link, or the sequence of links, of the pattern instantiation);
- the specified parameter coupling between the output of a source unit and the input of a connected unit in the pattern has to be kept unchanged in the pattern instance, even through sequences of units;
- container hypertext elements in the pattern, such as pages and areas, can be changed without constraints in the pattern instance, given that the links activation semantic is kept unchanged.

These instantiation rules are illustrated in Section 5 showing their application to the running case.

4. Design Patterns for Community-Driven Web applications

We analyzed ten of the most popular Web 2.0 community applications, as ranked by the alexia.com directory, 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.

Table 1: Community-Based web design patterns

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	• Relevance Adjustment
User Reputation	• Social Visualization	• Reputation Adjustment
Connections	• Relationship Setting • Browse by Connection	
Scoring		• Payment • Reward
Communication	• Talk • Recommendation • Invitation	• Notification
Permissions	• Permission Setting	• Permission Check
Interoperability	• Exportation	• Syndication

In the rest of this section we sketch the main features of the design patterns mentioned in Table 1, illustrating some of the pattern models and then showing how patterns are used in the reviewed social applications.

Item Clustering

Most virtual communities evolve around a particular category of items, e.g. bookmarks, videos, news. By Item Clustering, items that share some common properties can be grouped together in a Container element. This activity answers a twofold purpose. First of all, it provides a structured access to content items: for instance 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 (usually a hierarchy) for content items, most social application prefer to implement a collaborative approach to Item Clustering. A general solution is shown in the Organization front-end pattern.

When Item Clustering relies on a tagging mechanism, a peculiar front-end pattern is used to navigate the tag structure: the Browse by Tag pattern.

Organization. One of the most common ways members contribute to the growth of a virtual community is by publishing and sharing content with others; each user manages personal catalog of owned (favorite, suggested, etc) items, which grows over time. Because of this, and not secondarily in order to improve metadata accuracy, users should be entitled to organize contents into clusters that reflect their very personal mind-map of contents. The solution is to allow users organize their objects with a simple, yet flexible, mechanism that links in a many-to-many relationship items with containers. The container manipulation strategy can be implemented in several ways: containers can be personal or shared and collaboratively updated, created by users or picked up from a list of available ones, and so on.

A particular implementation of the Organization Pattern is the tagging mechanism that gives to the users the possibility to associate freely chosen words with the shared items. Tagging 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 an implementation of the Organization pattern using content tagging. A GetUnit named *Current User* retrieves the user identifier from the application session and passes this identifier to the *My Items* unit by means of a transport link (i.e. the dashed arrow). *My Items* is an IndexUnit that shows a list of Items using the current user identifier to display only the Items that are owned by the user (i.e. the ones that are connected to the user by the *Owner* relationship). The rest of the *Organization* page is occupied by an *Alternative* section that by default (D) shows an empty subpage named *No item selected*. At the moment of the selection of an item the subpage *Item selected* is displayed together with the units it contains: the *Item* DataUnit shows the details of the selected Item, and the *Item Tags* index unit displays the tags related to the selected item. The subpage allows the user to remove a tag association selecting a link exiting from *Item Tags* and activating a DisconnectUnit that removes the connection. On the other hand the user can also create a new tag association writing a tag name in the *Tag* EntryUnit. It is possible to display, for example in a dropdown menu, the list of the *Existing Tags* that contain the inserted keyword or to create a new tag by invoking the CreateUnit *Create Tag*. In both cases a ConnectUnit is activated to perform the final connection of the Tag with the Item. Transport (i.e. dashed) links perform the communication of the required parameters among the units. After the execution of the tag creation and connection sequence, the same Organization page is redisplayed.

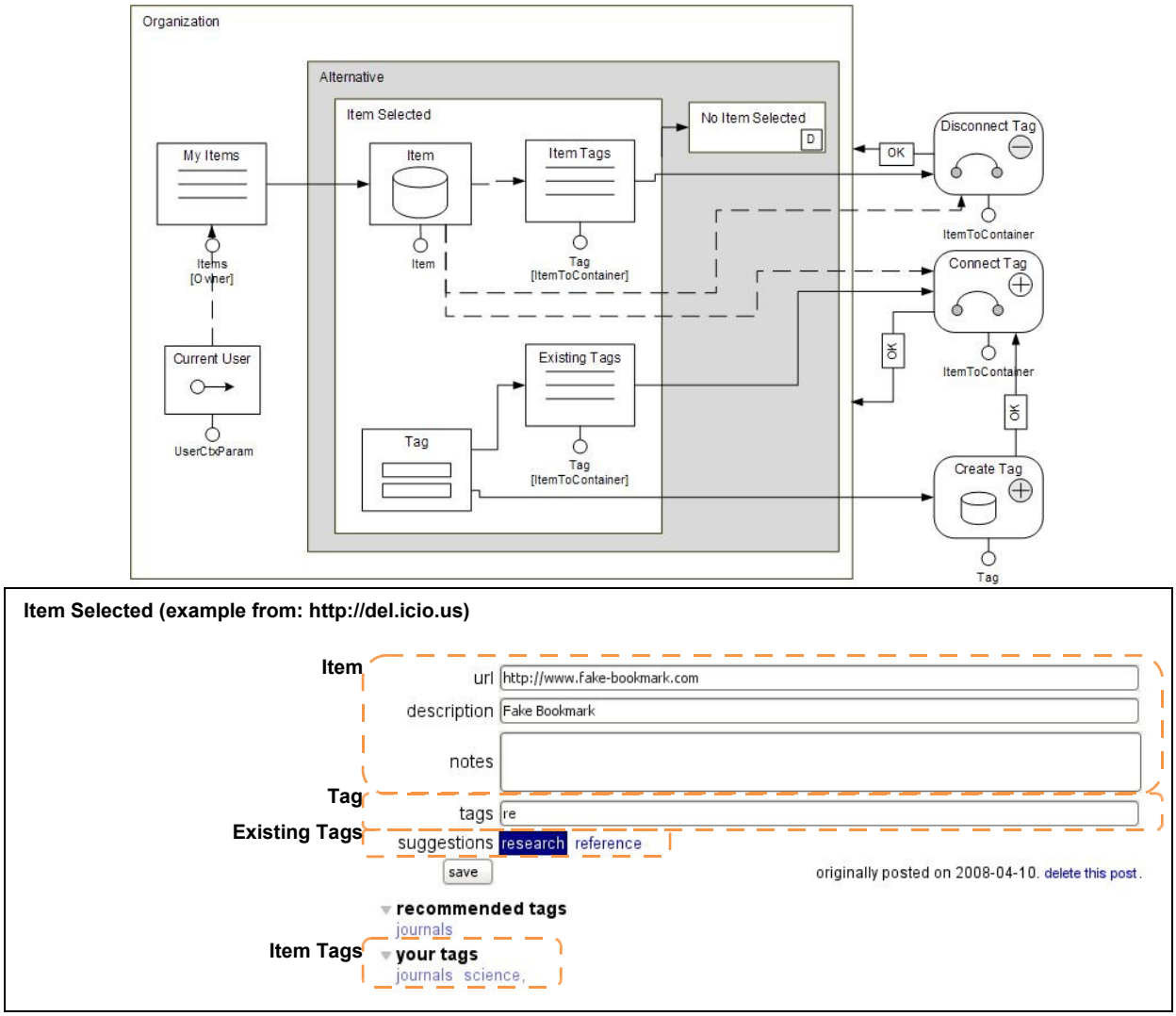

Figure 1:"Organization" pattern implemented by content tagging (WebML and HTML rendition)

Browse by Tag. Especially when dealing with large sets of items, users may prefer to locate content according to their mental organization, rather than based on a predetermined classification. One of the Web 2.0 best practices is to give users a mechanism for "tagging" contents with significant labels, so to create clusters of objects associated with specific keywords. More important tags should be easily recognizable by users in order to help them to predict the relevance of descriptions; this could be achieved by sorting tags, by indicating the number of items contained within each label, or by highlighting each tag with a different graphic appearance, depending on the number of occurrences it has. From the navigation perspective, tags can be seen as overlapping subsets of items; they define dynamic containers of objects that derive directly from the users' classification. Figure 2 shows an instance of the "Browse by Tag", expressed in WebML.

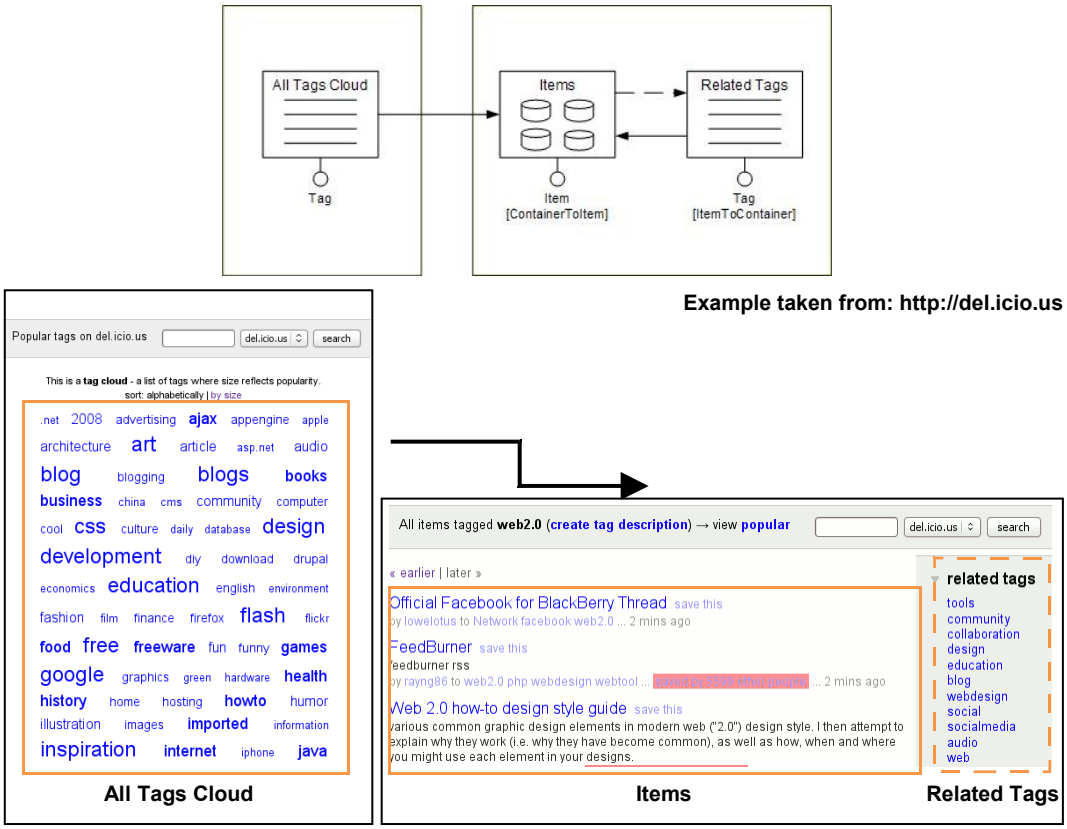

Figure 2:"Browse by Tag" pattern in WebML and its rendition in HTML

User Clustering

Several social applications create groups of interest in order to set a shared environment for communications and joint activities. In general, groups are characterized by a join policy (determining how members can participate to the group) and roles (determining who is in charge to do what).

Group Creation. This front-end pattern allows users to freely create new groups, and to set the group policies and roles. This collaborative way to address User Clustering is very common in Web 2.0 applications.

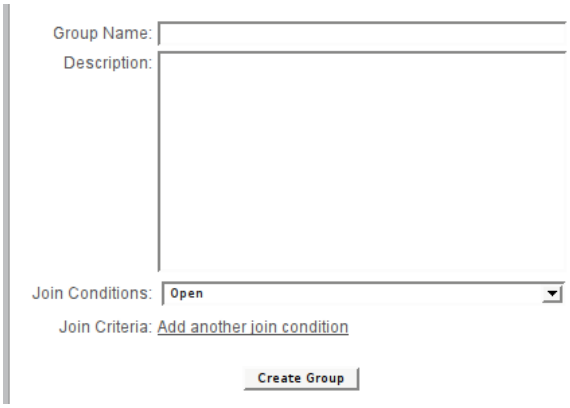

Figure 3: Group creation on Last.fm

Group Participation. Joining a group entails a number of back-end activities that need to be taken into account. A group participation could be autonomously and asynchronously performed by a member, but it could also be performed against the acceptation of an

invitation. The possibility of a member to join a particular group needs be evaluated on the basis of permissions and join policy; this could be achieved by the *permission check* pattern. Then, in case of success, the system may inform the group founder, and eventually all other subscribers through the use of a *notification* pattern.

Item Relevance

The free contribution policy of most virtual communities makes an Item Relevance mechanism necessary to help the user distinguish outstanding items. Community feedback needs to be taken into consideration for evaluating the quality of published items.

Rating. This pattern provides an explicit rating mechanism that can allow users to explicitly declare their liking about items. Ratings should be weighted based on the reputation of the user to balance them on the authority that the community recognizes to the member that expresses the rating.

Figure 4 shows the front-end patter for rating content. In the View Items page the user can see the details on an item and use a form to submit its rating (be it a numerical score or a more complex annotation). Submitting the rating information triggers a chain of components that create the persistent record of the rating, associate it with the item and the user, and call the Relevance Adjustment back-end pattern (implemented as a Web Service) that adjusts the score based on the user's reputation.

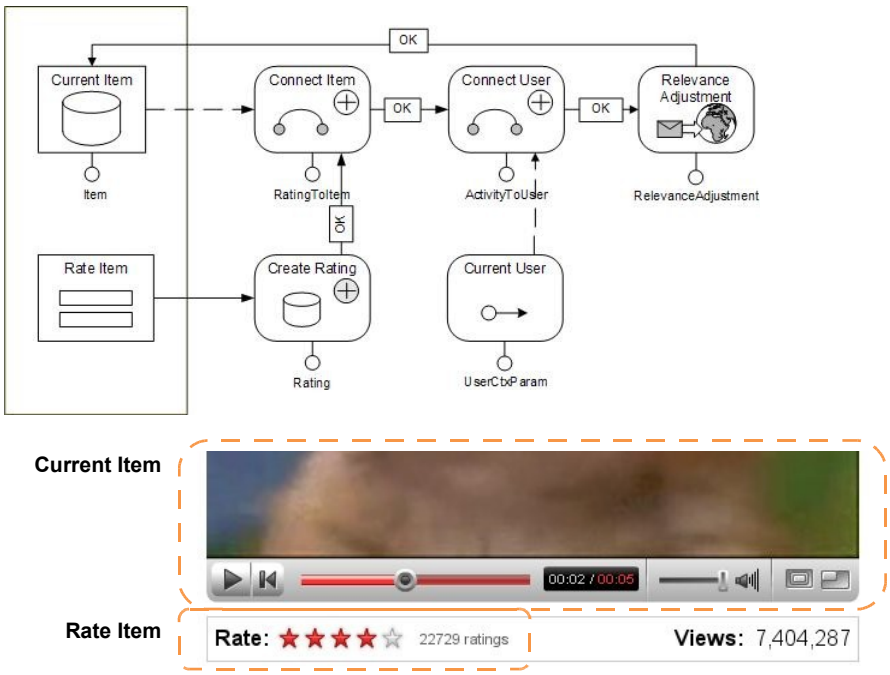

Figure 4:"Rating" pattern - WebML and HTML rendition of the interface for voting (example taken from http://www.youtube.com)

Flagging. The flagging front-end pattern is a quick method to let users mark an item or draw attention to it. The pattern allows people to easily point out a particular element that distinguishes itself according to a certain criteria. This solution is widely adopted, for instance, for signaling offensive contents.

Relevance Adjustment. This back-end pattern allows using any activity performed on an item to evaluate the quality of that item as perceived by the community. Usually social networks adopt mechanisms that combine both explicit (like ratings) and implicit (inferred by members' activities) evaluations to judge the value of an item and, consequently, the

reputation of the user who published it. Not all the activities impact the same on items' value: “viewing", for example, is a weaker indicator of quality than the “adding to favorites" activity; in this case, the two activities are said to have different *intensities*. Moreover not every activity corresponds to a positive evaluation: tagging an item as “offensive", for example, can be considered as a negative evaluation and so impact negatively both on the item’s value and the owner’s reputation.
The Relevance Adjustment back-end pattern is presented in Figure 5 in the form of a service that can be executed by other patterns whenever needed. The service caller needs to specify as parameters at least two identifiers that are used respectively to retrieve the item to evaluate and the triggering activity. In the WebML representation the Solicit Unit is the starting point of the service and the chain of OK-links shows the sequence of operations. The relevant Item and Activity are first retrieved and eventually used in the following operations as shown by the transport links (i.e. the dashed connections). A Switch unit selects a different path for each activity type, such as Rating, Flagging or Recommending. The computation of the New Relevance is then performed taking into account the appropriate formula and the previous relevance value (read from the selected Item). Finally the Item is updated with the new relevance value.

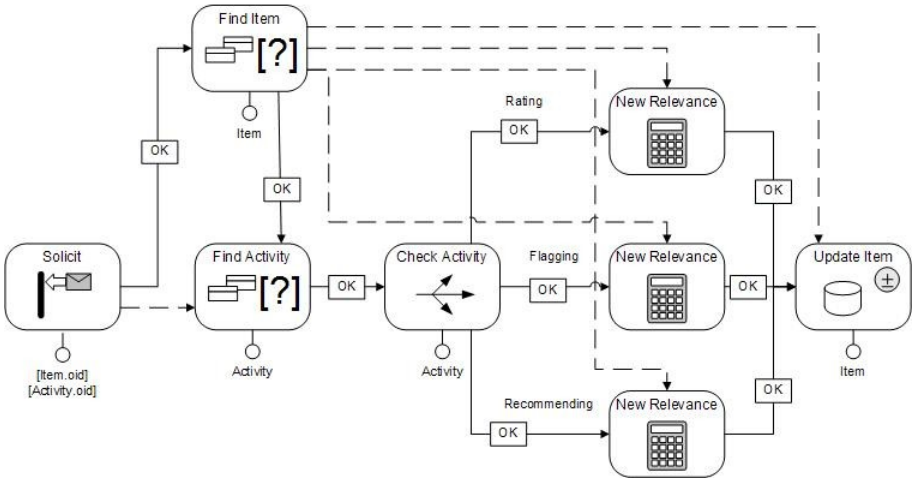

Figure 5: an instance of the Relevance Adjustment back-end pattern, triggered by the user’s activity on an item.

User Reputation

The definition of a User Reputation score in the virtual community is important for a twofold reason. First of all, members feel more likely to contribute if participation can help them to improve their status and visibility within the virtual community. As a second reason, when a member of the community contributes with an item, the relevance of such items should depend in general on the publisher's reputation.
Social visualization. This pattern can be an effective solution for fostering participation. By graphically visualizing levels of participation of community members, users can be stimulated to engage in responsible and reciprocal behavior. Social visualization builds upon one or more state variables denoting each member, like the number of contributions, the intensity of social relationships, and so on.

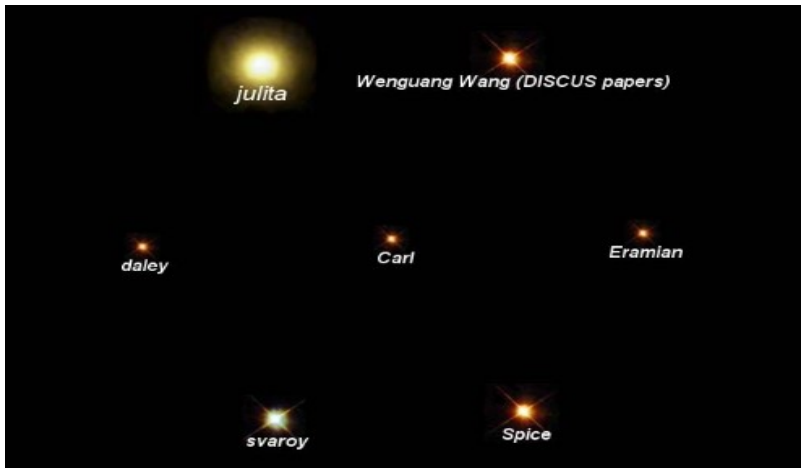

Figure 6: Social visualization in Comtella

Reputation Adjustment. This back-end pattern requires the definition of a reputation score and of a back-end mechanism to adjust reputation when necessary. The reputation of a member depends on several factors, e.g., *sociability*, the extent to which other members within the community establish social relationships with the user, and *feedback*, a measure of the quality that the community recognizes to the items contributed by the user.

Connections

Virtual communities are made of connections. Connections can be represented as graphs, where every member corresponds to a node, and edges between nodes denote member (or item) relationships.

Browse by connection. Connections can be exploited for proposing interesting candidate relationships to members, by letting them discover other potentially interesting members or items, reachable following a path of connection edges.

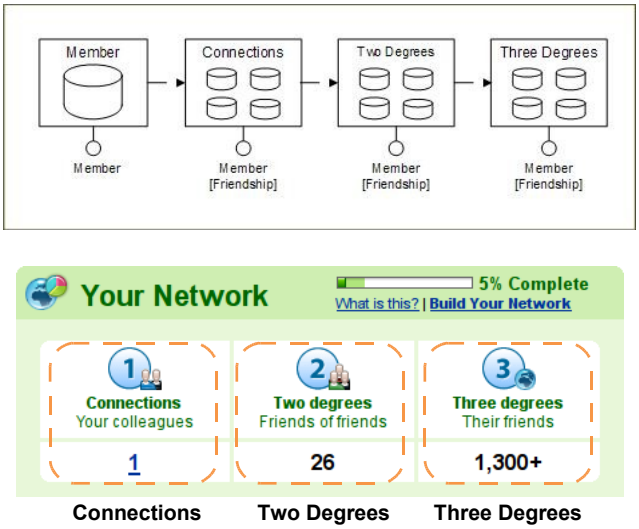

Figure 7: "Browse by Connection" pattern in WebML and its rendition in a Web page (example taken from http://www.linkedin.com).

Relationship Setting. In social networks members can set several kinds of relationships like friendship, nearness, familiarity, or they can block other users adding them to personal blacklists, and so on. Any member, regardless her group/role within the community, can

establish relationships with others. The pattern of relationship setting provides a generalized front-end to the management of the user's personal connections.

Scoring

Successful virtual communities are those in which members are happy to participate. By participation we mean activities (e.g., frequent logins, message checks, publications, rating, etc) which benefit the community and demonstrate involvement.

Reward. In order to foster participation, a proper reward mechanism is necessary: a certain amount of "credit" is paid to a member that performs a "constructive" activity. This credit can be manifested or hidden to the user until a certain threshold is reached, after which the credit is revealed.

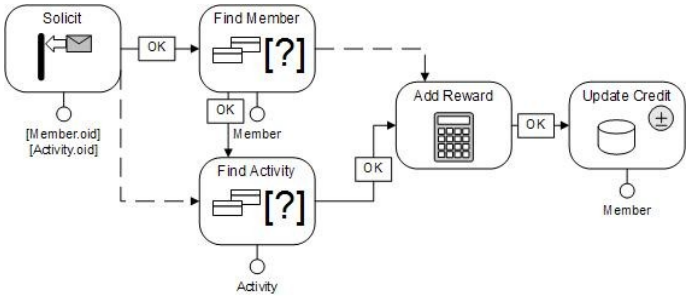

Figure 8: an instance of the back-end Reward pattern, triggered by the user's activity

Payment. Some activities within the community need to be subject to a policy of moderation, meaning that members can only perform them against the payment of a certain amount of credit, that should be limited. Credit and cost of an activity do not necessarily correspond to money (that is only a particular case). The cost rather corresponds to the extent to which the activity should be "pondered" by the performer.

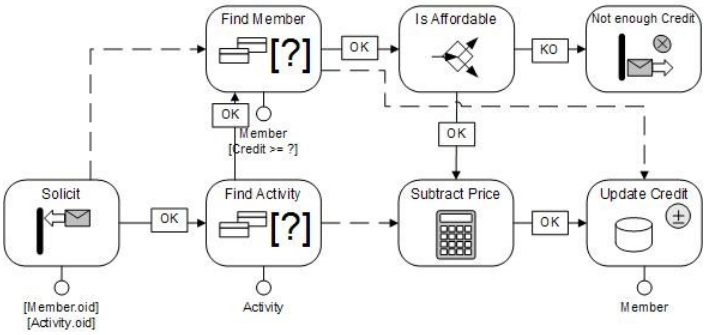

Figure 9: an instance of the back-end Payment pattern, triggered by the user's activity

Communication

Members of a community communicate to establish social relationships and to develop sense of belonging. To this end, free-will communication between users should not be limited to contributions, but should be promoted independent of their specific activity on items, e.g., by allowing the spontaneous exchange of messages.

Talk. Plain communication among community members is mostly intended as point- to-point communication; multicast and broadcast patterns are in general more critical because they expose the community to the risk of spamming, so their usage should be limited. A common practice when implementing a Talk pattern, is to forward messages also to third party

communication services (like mailboxes, instant messaging, mobile phones, etc) in order to support multi-channel communication. Figure 10 shows an instance of the Talk pattern, implemented as a page fragment for posting on the "wall" of a user.

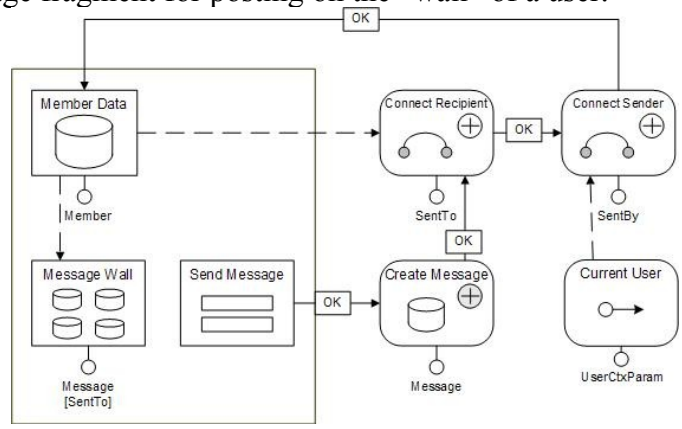

Figure 10: The Talk pattern, implemented as a Wall Posting mechanism, and its rendition in FaceBook

Invitation. The pattern implements a communication mechanism for letting members invite other users of the community to join groups of interest. A possible variant of this patter is applicable for inviting external people to join the social system, typically using traditional electronic communication means such as e-mail.

Recommendation. Members could wish to recommend an item to others (members or non-members). Although a recommendation may consist of, in practice, the plain forwarding of a certain message, it could also be figured as a promotion effort performed by members which, depending on community policies, could be treated as a reinforcing activity and rewarded. The suggestion message can be forwarded to another member of the community, through the notification pattern.

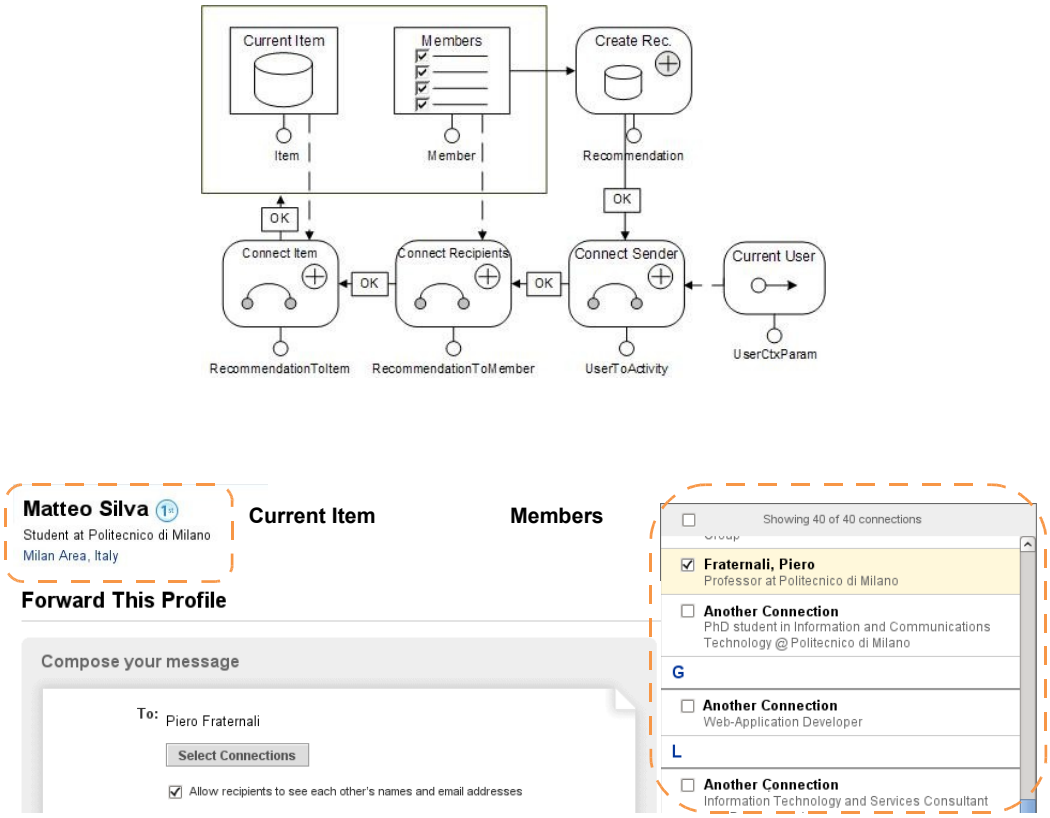

Figure 11:"Recommendation" pattern - WebML and HTML rendition (example taken from http://www.linkedin.com).

Notification. The system needs to notify members upon the performance of some activities performed by others (for example being added to somebody's list of contacts), the appearance of an issue (for example a low credit balance), etc. The pattern addresses the design of a built-in messaging system within the application. Usually such system is reinforced by a traditional delivery system, i.e. standard mailboxes or instant messaging systems.

Permissions

Not every activity available within the community can be performed by any member. Besides the usual statically defined access control, a more complex permission model is required to take into account the dynamically created groups, the connections by relationships, and the user responsibilities over the management of permissions.

Permission Setting. A member can set permission for granting/forbidding the execution of some activities to other members on items he published. The pattern implements a mechanism that is capable of setting permissions related to items, activities and dynamic aggregations of users, such as user-defined groups.

An example can be seen in the Flickr photo sharing community where every activity that members are generally allowed to perform on other's photos is subject to the permissions that the owner has established. This means, for instance, that it is possible for a member to grant the permission to view or comment a specific photo to familiars, but not to friends or public.

Permission Check. The pattern designs a permission checking process, to verify the possibility to perform any activity against every member request.

Interoperability

The Web 2.0 personal space comprises a set of different Web applications, each one with a specific perspective over the user information. Several of these applications provide the user with some kind of interoperability feature, to allow communication among these systems and data sharing. The communication can be initiated by a front-end pattern as shown in the Exportation pattern, or the data can be simply exposed by the back-end with a standard machine readable notation, as described by the Syndication pattern.

Exportation. The social application front-end needs to provide specific functionalities aimed at the exportation of the visualized content to be used by external applications, usually by the means of a Web services API exposed by the target systems.

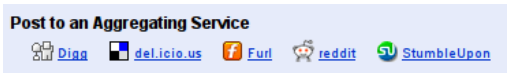

Figure 12: Exportation of a video in YouTube

Syndication. Formal description of contents with syndication systems and microformats has become an issue of growing interest for improving the degree of interoperability of applications. This pattern is used to provide a machine-readable description of items reflecting the logic organization of contents. Description can be implemented, e.g., according to the Atom/RSS specification. This pattern should be actuated against every modification and/or reorganization of contents.

Pattern occurrence in Web 2.0 applications

Table 1 summarizes the coverage of front-end patterns in the public areas of some of the most popular social Web 2.0 applications.

Table 2: Summary of the occurrence of the identifies design patterns in exemplary web 2.0 applications

	Browse by connection	Browse by tag	Exportation	Flagging	Organization	Rating	Recommendation	Group creation	Group participation	Invitation	Permissions setting	Relationship setting	Social visualization	Talk
Del.icio.us		X	X		X		X				X			
Facebook	X	X	X		X		X	X	X	X	X	X		X
Flickr	X	X		X	X	X		X	X	X	X	X	X	X
Hi5	X	X		X	X	X	X	X	X	X	X	X		X
Last.fm	X	X			X		X	X	X	X		X	X	X
LinkedIn	X					X				X		X	X	X
Ma.gnolia		X	X		X	X				X	X	X		
MySpace	X			X	X		X	X	X	X		X		X
Twitter	X		X	X		X				X		X		X

	Browse by connection	Browse by tag	Exportation	Flagging	Organization	Rating	Recommendation	Group creation	Group participation	Invitation	Permissions setting	Relationship setting	Social visualization	Talk
Del.icio.us		X	X		X		X				X			
YouTube	X		X	X	X	X	X			X		X	X	X

5. Community-driven development process

Design patters 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. Figure 13 shows how one of such schemes, the WebML process [2], extended 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 member's profile meta-data and the application design must incorporate the appropriate community patterns in the front-end and in the back-end.

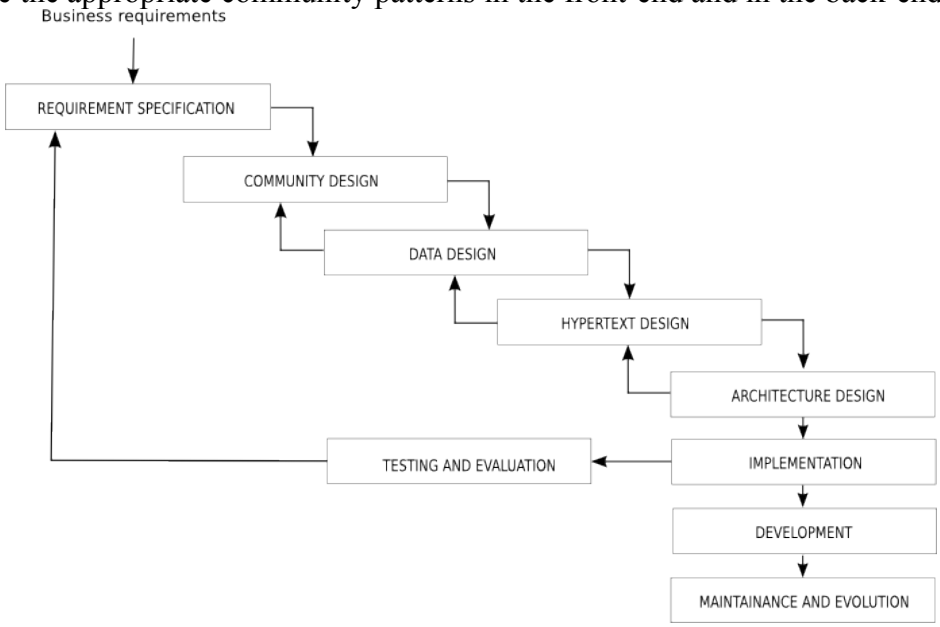

Figure 13: Community-aware development process

In the rest of this section, we briefly sketch the objectives, inputs and outputs of the development activities affected by the community-driven focus.

Community-driven requirement analysis

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.
The next figures show the community-driven requirement analysis activities applied to the running case. Figure 14 shows the classification of users in the running case stemming from the content-management, social and, governance perspectives. Figure 15 shows an example of specification of user's roles under the above-mentioned perspectives.

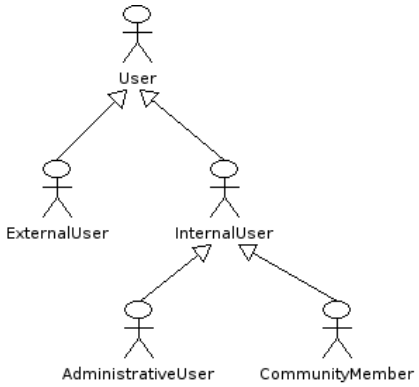

Figure 14: User classification in the running case

Group name	**ExternalUser**
Description	Unregistered users interested in accessing content published by other members.
Profile data	None

Super-group	User
Relevant use cases	• browse videos by means of the tagging mechanism; • read ratings proposed by other members; • read messages published on registered members' message walls.
Objects accessed in read mode	Item, Tag, Activity, Member, Message
Objects accessed in content management mode	None
Relationships with other users	None

Group name	**CommunityMember**
Description	Registered members of the community interested in sharing their videos and accessing content published by other members.
Profile data	First Name Last Name E-mail Login Password
Super-group	Internal User
Relevant use cases	• publish new videos; • classify videos and browse them by means of the social tagging mechanism; • instaurate friendship relations with other users and navigate the user base by friendship; • evaluate videos proposed by others through a rating system; • post messages to the profile page of other users; • recommend videos to other users.
Objects accessed in read mode	Item, Tag, Activity, Member, Message
Objects accessed in content management mode	Item [only the owned ones], Tag [only creation], Message [only creation]

Relationships with other users	Friendship [with other CommunityMember]

Figure 15: Community-aware requirement specification: eliciting member's roles, relationships, and policies

Community-driven design

In Web application design, several tasks are affected by a community-driven focus:

- The design of the data model of the application should encompass the meta-data needed to reflect the member's roles, relationships, profile data, and reward policies.
- The design of the hypertext front-end should integrate the selection of the illustrated navigation, contribution and social interaction pattern.
- The design of the hypertext back-end should comprise the back-end design patterns needed to support the supported social community governance and processes.

In the rest of this section, we will exemplify these concepts on the design of the running example. Figure 16 and Figure 17 show respectivelly the data model and WebML hypertext model of the MiniYouTube application. Only the *Community Siteview* of the hypertext model is reported, i.e. only the part of the model accessible to the non-administrative users.
Within the hypertext model the instances of the described patterns are highlighted.

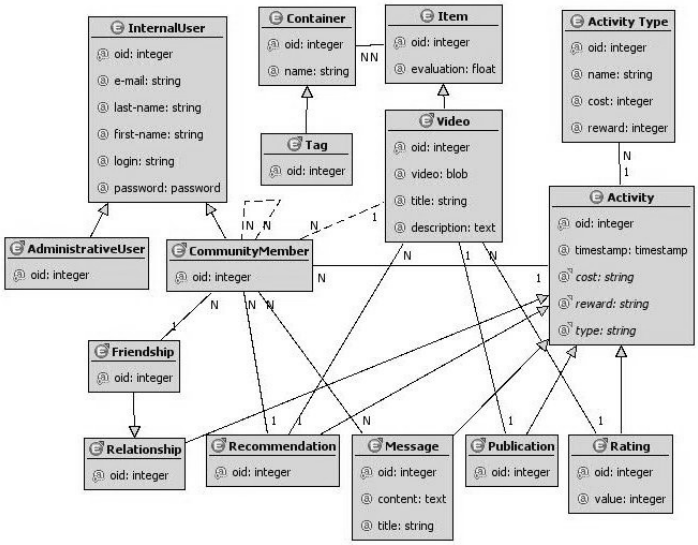

Figure 16: MiniYouTube Data Model

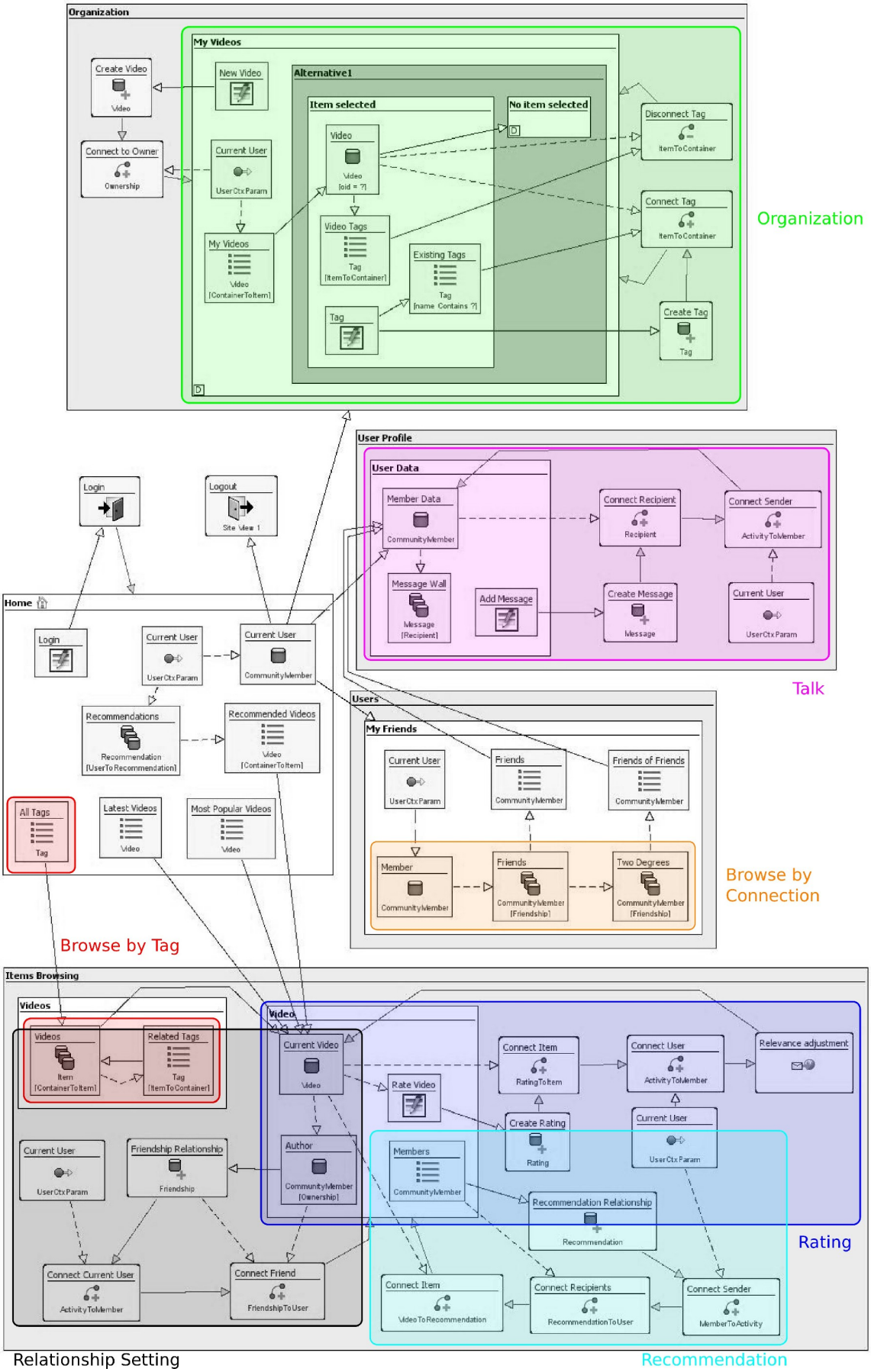

Figure 17: MiniYouTube WebML Hypertext Model – Community Siteview

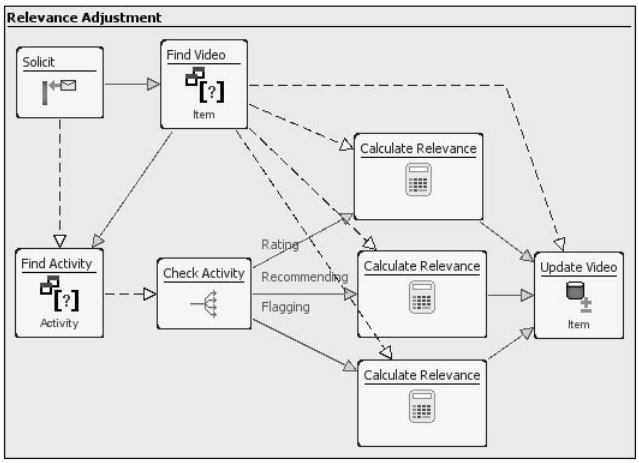

Figure 18: MiniYouTube WebML Web Service Model (Relevance Adjustment Design Pattern)

The use cases elicited during the community-driven requirement analysis of the previous section are easily associated to design patterns. For instance the use case "classify videos and browse them by means of the social tagging mechanism" will require the instantiation of the *Organization* pattern in its tagging variant and of the *Browse by Tag* design pattern, the use case "post messages to the profile page of other users" maps directly to the *Talk* design pattern.
The complete list of design patterns involved in the MiniYouTube example is:

- Organization
- Browse By Connection
- Relationship Setting
- Browse By Tag
- Relevance Adjustment
- Rating
- Recommendation
- Talk

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

1. The units specified in the pattern and their connections are first instantiated in the page *My Videos*. The instantiation requires the use of the entity *Video* 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 (in Figure 16 the *Video* entity is a specialization of the abstract *Item* entity).
2. The pattern is afterwards enriched by selecting, e.g., the suitable data elements to show, like 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, functionalities and model elements to obtain the desired level of functionality. In the example, by design-time decision, the *upload* functionality is added to the *My Videos* page. The unit *New Video* is inserted

to the page to allow the publication of new items by an ad-hoc operation chain. The new functionality eventually reuses suitable units of the pattern (in this case, the *Current User* unit). 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. For instance in the MiniYouTube application the Video page is obtained by overlapping three design patterns (i.e., Rating, Recommending, Relationship Setting) that share the content units of the page.

6. Future Trends

In the future social Web applications will be more and more integrated with "conventional" Web applications. It is possible to foresee that the success of the Web 2.0 will have an impact on many traditional software systems. Companies will start integrating social features in their B2C portals, organizations will foster user generated content and peer to peer interaction in their intranets, and so on.
Therefore, the integration of modern Web Engineering methods with Web 2.0 characteristics will become an essential factor for supporting the development of well-crafted, maintainable and socially effective Web applications.

7. Conclusion

In this chapter we have shown a way 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. The identified patterns have been implemented using a commercial Web Engineering tool (Webratio [20]) and are in use in the development of socially enriched B2C applications.

8. References

1. I Jacobson, G Booch, J Rumbaugh, The unified software development process - Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 1999
2. Ceri S., Fraternali P., Bongio A., Brambilla M., Comai S., and Matera M., Designing Data-Intensive Web Applications, Morgan Kauffmann, San Francisco (USA), 2002
3. J. Preece. "Online communities: design usability, supporting sociability". Introduction. Wiley & Sons (ISBN 0-471-8059998) (2001)
4. E Gamma, R Helm, R Johnson, J Vlissides, Design patterns: elements of reusable object-oriented software, Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 1995
5. OMG, UML Superstructure and Infrastructure Specification v 2.1.2, www.omg.org, 2007
6. Conallen, J. Building Web applications with UML, 2nd edition. Addison Wesley, 2002.
7. J. Bishop, "Increasing participation in online communities: A framework for human-computer interaction", Computers in Human Behavior 23 (2007), Pages: 1881-1893

8. J. Vassileva, R. Cheng, “User- and Community-Adaptive Rewards Mechanism for Sustainable Online Community ”, L. Ardissono, P. Brna, and A. Mitrovic (Eds.): UM 2005, LNAI 3538, pp. 342 - 346, 2005. © Springer-Verlag Berlin Heidelberg 2005
9. M. Van Welie, “Welie Interaction Pattern Library” – http://www.welie.com/patterns
10. H. Rheingold. “The Virtual Community: Homesteading on the Electronic Frontier”, London: MIT Press. (ISBN 0262681218) (2000)
11. P. Kolloch, M. Smith, “The Economies of Online Cooperation: Gifts and Public Goods in Cyberspace”, London: Routledge (1999)
12. Webster, J. Vassileva, “Visualizing Personal Relations in Online Communities”, V. Wade, H. Ashman, and B. Smyth (Eds.): AH 2006, LNCS 4018, pp. 223 - 233, 2006. © Springer-Verlag Berlin Heidelberg 2006
13. Y. Mao, J. Vassileva, W. Grassmann. “A System Dynamics Approach to Study Virtual Communities”. Proceedings of the 40th Annual Hawaii International Conference on System Sciences (HICSS'07)
14. N. Hoebel, S. Kaufmann, K. Tolle, R.V. Zicari. “”. Web Intelligence, 2006. WI 2006. IEEE/WIC/ACM International Conference on Volume, Issue, 18-22 Dec. 2006 Pages: 317 – 320.
15. J. Yu, Z. Jiang, H. Chan, “Knowledge contribution in problem solving virtual communities: the mediating role of individual motivations”, Proceedings of the 2007 ACM SIGMIS CPR conference - Session: Virtual community contributions Pages: 144 - 152
16. J. Han, R. Zheng, Y. Xu, “The Effect of Individual Needs, Trust and Identification in Explaining Participation Intentions in Virtual Communities ” - Proceedings of the 40th Annual Hawaii International Conference on System Sciences (HICSS'07)
17. J. Soto, A. Vizcaíno, J. Portillo-Rodríguez, M. Piattini - “Applying Trust, Reputation and Intuition Aspects to Support Virtual Communities of Practice”, Springer Berlin / Heidelberg, (ISBN 978-3-540-74826-7) (2007)
18. Chin, M. Chignell, “A Social Hypertext Model For Finding Community In Blogs”, HT’06, August 22–25, 2006, Odense, Denmark.
19. H.Y. Lee, H. Ahn, I. Han - “VCR: Virtual community recommender using the technology acceptance model and the user’s needs type”, Expert Systems with Applications, vol. 33, no. 4, November 2007, pp. 984-995
20. WebRatio. http://www.webratio.com.

9. Key Terms and Their Definitions

- Virtual Community.
  Social network made of people with a common interest, idea, task or goal that interact in a virtual society.
- Model-Driven Web Development.
  A branch of Web engineering which addresses the specific issues related to design and development of large-scale Web applications. In particular, it focuses on the design notations and visual languages that can be used for the realization of robust, well-structured, usable and maintainable Web applications.
- Design Pattern.
  A recurrent problem in software design and a general reusable core solution to it. The pattern is usually presented by the means of a visual notation with possible variants and a textual specification in order to express relevant semantic aspects.
- Pattern Instantiation.

Development activity for the application of a design pattern to a specific application. It requires the selection of real design elements to perform all the roles specified in the design pattern and the integration of the pattern model with the rest of the system.

- Social Web application.
  Web application that aims to build online social networks for communities of people who share interests and activities or who are interested in exploring the interests and activities of others.
- Tag.
  Term associated with a piece of information to describe the item and enable keyword-based classification and search.
- Community-driven requirement.
  Additional application requirements derived from a community-driven analysis, with the aim to foster user participation and to ease the development of social relationships.