From Requirements to Implementation of Ad-hoc Social Web Applications: 
an Empirical Pattern-Based Approach

MARCO BRAMBILLA
POLITECNICO DI MILANO, ITALY

Abstract

Social technologies are transforming the Web to a place where users actively contribute to content production and opinion making. At B2E level, enterprises look at social networking tools as possible means for improving their operations thanks to the unstructured interaction they foster among employees. This paper tackles the problem of defining a quick and pragmatic approach for specifying the requirements of ad-hoc social web applications and for going down to their implementation. The approach is motivated by the fact that in most cases the ad-hoc needs that the company foresees are just a peculiar subset of the large set of requirements that are addressed by current social platforms. The main contributions of this work include: the classification of the social interactions that can be observed in virtual communities in a comprehensive taxonomy of requirements (based on a broad analysis of the existing social networking platforms, considering both public and enterprise solutions); the definition of requirement specification as a selection process upon this taxonomy; and the mapping to design and implementation artifacts through automatic generation of design models and of running code. This is implemented through a set of pattern-based model transformations that transform the requirements (specified according to a goal-oriented methodology) to UML models and then to running Java code. A validation of the effectiveness of the approach is provided, in terms of percentage of hand-written code with respect to the parts that can be automatically obtained through the transformations.

1 INTRODUCTION

Social technologies are transforming the Web to a place where users actively contribute to content production and opinion making. The first-generation social tools (blogs,
wikis, and content sharing applications), have been followed by a second generation of solutions, which are known under the umbrella term of social networks, whose main aim is to allow people to interact and establish relations online. While the broad public is aware of only a bunch of world-spread applications (including Facebook, Gowalla, Foursquare, LinkedIn, Twitter), social networking requirements are becoming a core part of the needs of modern enterprises, at the B2C, B2B, and B2E levels.

Several examples of applications exist at B2C level, spanning from brand management and viral marketing (exploiting the dissemination power of social networks to test or promote products and services), to Customer Relationship Management (exploiting enterprise blogs and social network communication to install a multilateral dialogue with customers).

At B2E level, enterprises look at social networking tools as possible means for improving their operations thanks to the unstructured interaction they foster among employees. This complements more structured organizational approaches like BPM (Business Process Management). This evolution has been recently witnessed by business analysts [ORS10] who defined Social Business Process Management (Social BPM) as the effort of designing and executing business processes collaboratively.

This implies a broad adoption of web and social networking technologies within the enterprise, but also imposes new kinds of requirements to the developed software applications. In particular, while in some cases general public social networks can be adopted, in most situations the companies need to deal with specific needs and thus opt for ad hoc social solutions. Some general purpose enterprise social networking platforms exist, such as Tibco Tibbr [Tibco11] or SalesForce Chatter [SF11], which can accommodate these requirements in some scenarios, but for the other cases companies still need to build their own social networking platform.

This paper tackles the problem of defining a quick and pragmatic approach for specifying the requirements of ad-hoc social web applications and for going down to their implementation. The approach is motivated by the fact that in most cases the ad-hoc needs that the company foresees are just a peculiar subset of the large set of requirements that are addressed by current social platforms. In other words, a new social application is often required to implement a set of requirements that existing systems already provide, while completely new needs are rarely indentified. Thanks to
this tendency, the requirements specification can be made much more efficient by reducing it to a simple selection of requirements from a predefined taxonomy, rich enough to cover all the typical social behaviors. This also paves the way to a very efficient design and implementation of the applications, since the corresponding artifacts can be predefined according to the existing requirements taxonomy, and thus be integrated so as to compose the final application.

The solution we propose is therefore to study and classify the social interactions in a taxonomy of requirements (based on a broad analysis of the existing social networking platforms, considering both public and enterprise solutions), and then allow requirement specification as a selection process upon this taxonomy. Subsequently, quick design and implementation can be supported by providing automatic generation of design models and of running code, based on the selected requirements through pattern-based model transformations that transform the requirements (specified according to a goal-oriented methodology) to UML and WebML models and then to running Java code.

The selection-based requirement specification is not limiting the expressive power or the freedom of the analyst, since we adopt a general-purpose requirement specification approach and therefore we allow the analyst to expand the requirements at will, so as to cover aspects that are not described in the proposed taxonomy. In particular, in our analysis we do not consider: any non-functional requirements.

To demonstrate our approach, we present the requirement definition phase through the early requirement specification, based on a goal-oriented notation and method (namely, i* [Yu02]). On one side, this highlights the benefit of our solution by providing a straightforward direct path from early requirements down to the design; on the other side, this keeps the requirements specification phase at a very coarse level of detail. However, the same discussion and solution applies if more advanced requirement specification phases are covered, possibly based on Web-specific requirement specification methodology, such as WebSpec [RGGW10], MoLIC [PSB05], WebRE [EK04], or others.

The main contributions of this work therefore include: the classification of the social interactions that can be observed in virtual communities; the definition of requirement specification as a selection process upon this taxonomy; and the mapping to design and
implementation artifacts through pattern-based model transformations towards design models and of running code. Finally, a validation of the effectiveness of the approach is provided in terms of productivity gains for the developer. The productivity gain is obtained through the combination of three main ingredients: clear and comprehensive requirement taxonomy, a set of predefined design patterns already in place as devised by the proposers of this method, and the model transformations that map the requirements to the patterns.

Our approach grants the possibility of designing complex and yet completely customized social applications. This is a major advantage for enterprises, individuals and organizations that do not want to exploit public social networks; furthermore, our approach covers by far much more features than out-of-the-box enterprise social platforms like Tibbr, Saleforces, or event IBM Blueworks Live, which focus on some specific characteristics (e.g., microblogging, or collaborative development), disregarding the others.

Notice that in principle the proposed approach can be potentially generalized to any kind of Web applications. The focus on social aspects is motivated by four main reasons: 1) the need of limiting the field of coverage of the study and of the taxonomy; 2) the possibility of raising the level of abstraction in the specification of the requirements with respect to the general case, in which detailed specification of content, navigational and presentation requirement was needed; 3) the fact that the interactions between the different requirements is less critical than in the general case, and this facilitates the subsequent steps of the method, which instead would have been cumbersome if covering the entire spectrum of web requirements (in particular, notice that non-functional requirements are out of the scope of this work); and 4) the large number of requirements and broadness of the taxonomy gives is a good clue for asserting the possible generalization of the approach.

This paper is organized as follows: Section 2 offers an overview of our approach; Section 3 presents our analysis of the existing social networking platforms and provides a taxonomy of social requirements; Section 4 describes a running example and how to apply the requirements collection according to our approach; Section 5 presents the design and implementation phases following the requirements; Section 6 discusses the implementation issues; Section 7 describes our experimental findings; Section 8 discusses the related work; and Section 9 concludes.
2 OVERVIEW

Our approach to requirements specification and implementation of social networking applications follows a standard software engineering development process. The added value of the work is to provide a comprehensive taxonomy of requirements for social applications and to increase the productivity of each step thanks to appropriate model transformations and to some simplifying assumptions that apply to social application design.

Figure 1. Overview of the pattern-based development approach for social web applications.

Figure 1 shows the general flow of the development steps (according to the BPMN notation), that comprise:

1. **Requirements specification:** since most ad-hoc social applications can be defined in terms of as a subsets of all the features of existing social applications, the requirements specification phase can be implemented with a subtractive approach starting from a “universal” social specification design. Assuming that such a design exists (in terms of a taxonomy of requirements specified in a semi-formal notation, such as a goal-based model), the requirement specification is a simple selection of requirements from a set. If further requirements are needed, they can be specified as an extension of the selected social requirements.

2. **Design:** based on the set of requirements selected before, the design (specified with a general purpose language like UML and a domain-specific language like
WebML) is built through a pattern-based model transformation that associates to every requirement the corresponding design model fragment and integrates it with the other fragments into a unified and coherent model. This is made possible through the a-priori design of the static and navigation models (defined as patterns) for every macro-requirements, together with the definition of a set of model transformations that merge the patterns. Obviously, the part of the design associated to the requirements external to the specified taxonomy cannot be translated so easily to existing designs, but our finding showed that for social applications this part is unusually small.

3. **Implementation**: once the design is completed, the predefined design patterns may be directly available as running applications, including the UI aspects. The new pieces instead need the usual coding effort.

3 **Analysis of Social Networks Requirements**

For enabling the pattern-based, subtractive approach to social requirement specification, a preliminary study must be carried out to assess the state of the art of social networks and the corresponding features.

The first step of our work has been actually devoted to the massive analysis of social networks features, at the purpose to reverse engineer all the possible functional requirements associated to them and therefore build a broad, comprehensive requirement taxonomy for social applications.

In our analysis we considered more than 30 social networking applications and we struggled to extract coarse and detailed features; in turn, these features have been reverse engineered to requirements and a complete mapping has been performed between the applications and those requirements. Although we didn’t follow entirely a theoretical methodology for systematic reviews like the one presented in [K04], we tried to adhere to the basic guidelines and we formally clarified the objectives and criteria that are imposed by these methods. In particular, we defined the structure of the analysis in Table 1 as requested in [K04]. The requirements have been adapted to the focus of this work, which is not literature review but software applications review.
Table 1. Fragment of the feature mapping table for social networks.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Explanation</th>
<th>Analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td>Background</td>
<td>The rationale for the survey</td>
<td>Helping the analyst in defining the requirements for social Web applications</td>
</tr>
<tr>
<td>Research</td>
<td>The research questions that the review answers</td>
<td>Analyze all the popular social Web application for mining high-level functional features and requirements</td>
</tr>
<tr>
<td>Strategy</td>
<td>The strategy used to search for the features</td>
<td>1) Identify a set of important applications; 2) For each of them, identify core features that a user would perceive as a separate functionality; 3) Cluster together similar features from different platforms when their aim and interaction format can be considered the same</td>
</tr>
<tr>
<td>Selection criteria</td>
<td>The criteria for including in, or excluding a system from, the systematic review</td>
<td>Inclusion of systems is based on: 1) popularity of the social platform according to published statistics; 2) diversity of the platform in terms of target users, objectives and features</td>
</tr>
<tr>
<td>Quality assessment</td>
<td>The quality assessment checklists and procedures</td>
<td>Quality assessment of the analysis is based on cross-validation of the findings between two researchers. Disagreements are solved through direct discussion between the two</td>
</tr>
<tr>
<td>Synthesis</td>
<td>The synthesis of the extracted data</td>
<td>The synthesis consists of the clustering procedure for similar features from different platforms</td>
</tr>
<tr>
<td>Timetable</td>
<td>The definition of the review plan</td>
<td>To grant coherent and fair comparison of the social Web application, the reviews have been performed in a tight timeframe (1 month, December 2010). The synthesis phase took 1 subsequent week.</td>
</tr>
</tbody>
</table>

The set of analyzed social networks, grouped by category or focus, include:

- general-purpose networks: Facebook, Netlog, Google Buzz
- blogs and micro-blogging platforms: Twitter, Live Spaces, Tumblr, Jaiku, Xanga
- business networks: LinkedIn, Xing, Viadeo
• video and multimedia: YouTube, Flixster
• photography: Flickr, Fotolog, Faces, Sneppi
• music: MySpace, Last.fm, Blip.fm, Trig
• books and reading: Anobii, weRead, Shelfari, GoodReads
• art and images: DeviantArt
• travel: Wayn, TripAdvisor, Tripit
• social bookmarking: Delicious, Bibsonomy, Citeyoulike
• location-based: Foursquare, Gowalla, Brightkite

As an example, Table 2 shows the subset of the most known social networks that have been analyzed and the respective high level features. As the table shows, a wide set of high level features is recurrent in many social applications. However, every application assumes its flavor simply based on the features that are included or left out. For instance, friendship is featured by all the social networks (unsurprisingly, being at the basis of any social interaction); however, other popular features such as user groups, recommendations, notifications, favorites, and item rating are differently supported by the various networks. Notice that some relevant features (e.g., geo-location, checkin and badging mechanisms, and so on) have been left out of the table for space reasons. Furthermore, a similar analysis can be carried out at a lower level of detail, by considering more precisely how these features are supported. We don’t go into this now for space reasons.

Table 2. Fragment of the feature mapping table for social networks.

---

1 The analysis has been performed in December 2010.
Just at the coarse level of detail, we identified 52 core features. In turn, each of them splits into 5 to 15 sub-features. We converted these features into requirements and we represented them in a goal diagram. Figure 2 shows an extract of it, highlighting only the 52 main goals. For sake of readability, we classify them into 7 categories: user relations, organization, content interaction, user interaction, system actions, messages, and a catch-all category called extensions. Overall, the complete goal map would include more than 500 goals.

Despite its completeness and complexity, notice that the purpose of this goal model is not to represent all the goals associated to a Web application. Indeed, this would make it less useful to the designer than the current one. Our focus is to provide a taxonomy of functionalities that one can implement in a social system, so as to facilitate the selection and the subsequent development. Therefore, the highlighted relations between goals are not covering all the possible dependencies, but only the taxonomical ones, plus the
essential mandatory dependencies. On one side, this makes the model incomplete (for instance, keeping out all the non functional requirements, plus the general-purpose Web application requirements), but on the other side this lets the designer identify clearly the categories of requirements and decide which ones to implement. Once this is done, the model can be enriched at will with non functional goals, soft goals of any kind, general-purpose requirements for possible non-social parts of the application, and additional dependencies.

Figure 2. Goal model and classification of goals for social networks (high level of abstraction).

The category user relations includes all the requirements related to the interactions between users and groups of users. In particular, they deal with profile management, friends and groups management, subscriptions to (other users’) events, and
management of the various kinds of *user interactions* (described in the corresponding subcategory), which also include *message exchanges* in various formats. The category *organization* describes the basic system features that enable the management of access control to resources, customization, search capabilities, statistics, and notifications. The content *interaction* category includes all the requirements associated with the management and socialization of contents, such as uploading, rating, tagging, and so on. Finally, the *extension support* category describes the management of plugins, import/export features, shopping, and location-based services.

At the purpose of exemplifying the detail level that has been reached in the taxonomy of goals, Figure 3 shows the detailed diagram for one of the 52 macro-goals depicted in Figure 2, namely the *Profile management* goal. This goal comprises three main sub-goals aimed at the user login, and at publishing and editing of the user profile. The login is a mandatory requisite for allowing profile management and private or protected viewing. The public profile is visible also to non-registered users, while the private view is reserved to the profile owner and the protected one is reserved to his network of contacts. The editing of the profile includes the management of several profile parts. Their coverage is quite diverse depending on the social network of choice.

Due to space reasons, it’s impossible to report in this paper all the details of the taxonomy and the description of the patterns. A more analytical description and classification is available online at the following URL:

http://dbgroup.como.polimi.it/brambilla/socialrequirements.
4 DEVELOPMENT APPROACH

In this Section we detail the quick development approach we propose, with special attention to the requirements specification and design phases. To exemplify the approach, imagine the following running example: we want to build the PoliBook application, which is a simple social network dedicated to university students (namely, Politecnico di Milano students in our case) where members can refine their profile, select their friends and create or join groups. Students can send messages, get updates and notifications, and can rate materials. On the other side, we don't want to allow open registration of people (enrolled students are automatically registered by the university management system) or free posting of materials; we also require automatic group creation for university courses (and automatic subscription to these groups by students attending the respective courses).

4.1 REQUIREMENTS SPECIFICATION

The requirement specification phase exploits the social requirement analysis performed as described in Section 3 and consists in defining the flavor of the ad-hoc social network by selecting the subset of requirements that are deemed useful for the specific needs. Figure 4 shows a snapshot of the goal oriented tool we exploited: the analyst simply starts from the comprehensive goal model specified in Figure 2 (actually, the detailed version thereof) and removes all the requirements that are not relevant for his application, with a purely subtractive process.

If needed, the analyst can enrich the model with additional goals, including non-social functional goals and non-functional ones, and/or relations between them. The result is a goal model that closely describes the objectives of the application.
In our running example, we select as coarse goals for PoliBook the following ones: Profile, Friends, and Groups as social interactions; Posts, Guestbook, Comments, Inbox and DirectMessages as message management; Authorization management; Updates, Activities and Notifications as system actions; and rating.

4.2 DESIGN

Once the requirement phase is completed, the goal model is fed into a model transformation that produces a complete and coherent class diagram covering the design of all the predefined goals and their relations. This represents a major advancement in terms of design productivity, as the designer gets automatically a large share of the design. This has been made possible thanks to the a-priori specification of a set of design models (namely, UML class diagrams and WebML navigation models) for all the known social requirements in the taxonomy of Figure 2.

Concretely, in our work we manually devised a UML class diagram and a WebML navigation model (describing what we call a social pattern) for every category of requirements, plus some “bridge” class diagrams that connect them.

The UML models have been designed so as to grant that the goal-to-class transformation can act as follows: if at least one goal has been selected in a category, the respective UML...
social pattern is picked; if two UML social patterns are picked, also the “bridge” UML models are picked. Furthermore, since the pattern would support all the requirements of the category, the transformation takes care of:

- preserving the part of the pattern that is common to all the requirements of the category;
- stripping out all the parts of the pattern associated to non-selected goals.

To exemplify the kind of models that have been designed, Figure 5 shows a simplified version of the class diagrams designed to address the Messages and Contents Interaction categories of requirements. Analogous models have been designed for the other categories.

Once an actual application needs to be implemented, the transformation takes care of integrating the relevant models and dropping the irrelevant parts. Subsequently, based on this model, the designer can refine or extend the obtained models to fit the specific requirements.

Inconsistencies and alignment between the patterns are granted once again through a mapping approach: the transformation also includes a set of modifiers to be applied in case 2 or more patterns need to be merged. Thanks to the almost complete orthogonality of the requirements categories, the number of possible combinations do not explode, because only intra-category dependencies must be considered.

The mapping transformation assumes the following shape:

```
<category name= "x">
    <requirement id = "x.1">
        <staticmodel id = "j">
            <navigationmodel id= "k">
            </navigationmodel>
        </staticmodel>
    </requirement>
    ...
    <modifier id = "x.m1">
        <requirement id = "x.y">
            <requirement id = "x.z">
                <requirement id = "x.w">
                    <staticmodelchanges> //list of delete/create items
                        <navigationmodelchanges> //list of delete/create items
                    </staticmodelchanges>
                    <navigationmodelchanges>
                </requirement>
            </requirement>
        </requirement>
    </modifier>
    ...
</category>
<category name = "y"> ...
```
For every category of requirements, all the mappings for the requirements are listed, together with the modifiers that need to be applied to the patterns when merged with others. Besides our best attempt to cover these cases, there will always be the need of manual refinement.

For instance, in our running example a few refinements must be applied. One case is based on the fact that in the general design the Rating class is associated to the generic Content class; the “bridge” model (not shown here) in this case consists of a UML class diagram comprising a IS-A hierarchy between the Content superclass and the Message and Post subclasses. However, in our case we want only Posts and Comments to be rated, and these two classes have nothing in common. Therefore we manually drop the Content-Rating association and design the two associations from Rating-Post and Rating-Comment, as shown in the UML class diagram detail of Figure 6.

(a)
Figure 5. UML class diagram for the Messages (a) and Contents Interaction (b) categories of goals.

Figure 6. Fragment of UML class diagram finally obtained in the design phase for the Rating goal of the running example.

The navigation perspective of our patterns is represented through WebML diagrams. WebML is a Domain Specific Language for describing the interaction and business logic.
of Web applications through a visual notation. WebML covers composition (describing which contents make up a Web page) and navigation (described by links that users can navigate). The pages of the Web site are the containers units, atomic content elements used to publish information or perform business actions. Figure 7 shows an example of WebML model describing the navigation model for the Post requirement. The model shows the Posts page, where the current user ID is retrieved (GetCurrentUser unit), together with the list of his posts (PublishedPosts index unit). The user can click on a post and see the list of comments associated to it (Received Comments index unit); he can also delete a post or a comment, by clicking on the corresponding links. Through the New Post form, the user can create a new post. Upon submission, the post is created and connected to the current user.

Figure 7. Fragment of WebML navigation diagram describing the user interaction for the combination of Post and Comment requirements.

4.3 IMPLEMENTATION

The implementation phase follows the UML and WebML modeling one. For the parts that are not predefined in the requirement taxonomy (and hence not already mapped to a designed pattern), it consists in generating the class declarations in the implementation language of choice, by exploiting standard interface generation software
for UML. If one of the predefined patterns must be implemented instead, the entire class code can be automatically obtained, as it has been already prepared in a preliminary phase. Again, only the proper class implementations (and the proper methods) need to be selected for producing the final implementation, based on the classes (and the other design models) that build up the design. This aspect is taken care of by the model-to-code transformation provided by our approach: the transformation to the running code is once again consisting of a mapping from the goals to the implementation-level artifacts (classes and methods).

In our approach we opted for generating **MVC code (model-view-controller)** in the J2EE environment. Therefore, the class diagrams represent the **model** part of the application, while a set of JSPs implement the **view** part and a configuration file (namely the struts-config file, as we adopted the Struts framework) together with a set of servlet implement the **controller**.

For the navigation part, one can either opt for creating fixed templates and then editing them by hand, or for exploiting the WebML models. In the latter case, the transformation from WebML to the user view components (i.e., the JSP pages) can be achieved by exploiting the code generation facilities of WebRatio\(^2\), the official model-driven tool that supports WebML.

5 **IMPLEMENTATION**

To demonstrate, validate, and evaluate our proposal we have built a complete implementation of the ideas, by exploiting existing tools when possible. In particular, we assume usage of commercial or opensource tools for drawing requirements and design models. In this framework, our contribution has been to define the comprehensive social requirements models, the respective design models, and the associated default implementations in Java. Furthermore, a set of imperative model-to-model transformations have been specified for transforming requirements to design models and model-to-text transformations for generating the final Java code.

In our experimental setting, we used GR-Tool [GRT11] for modeling the goal-based requirements (other tools can be used, such as OpenOME [OOME11]). The outcome is fed into our goal-to-class transformation tool, implemented as a Java application, which

\(^2\) http://www.webratio.com
produces a set of UML diagrams. Figure 8 shows the dialog displaying an intermediate output of the transformation, in which, by analyzing the goal model, a set of classes (grouped in categories) are identified as relevant for the application. Afterwards, the corresponding classes are selected in the predefined designs and integrated within a coherent set of models.

![Figure 8. Intermediate outcome of the Goal-to-UML transformation.](image)

These models can be opened and refined with a UML editor (in our case, we used the ArgoUML tool), and finally the proper class implementations are chosen from the predefined ones.
Figure 9. Sample UIs of the generated social Web application: (a) user home page, (b) user friends page, and (c) group page.

The final outcome consists in a running prototype of the designed application, in which the user interfaces (defined as JSP files) present a basic menu of features (corresponding to the ones selected in the goal model) and for each of them provides a basic behavior. The visual part is implemented through a fixed CSS file, describing the graphical properties of the interface.

As an example, Figure 9 shows a set of snapshots corresponding to the running example application: (a) the user home page with the summary of the updates, notifications, and activities; (b) the page listing the friends of the user; and (c) the page describing a group of interest.

6 Experiments and Evaluation

Based on our implementation, we conducted a set of experiments to validate the approach and to check the advantages it brings to development productivity.

The analysis aimed at assessing the actual advantage provided by our approach with respect to manual application development. Indeed, we wish to evaluate the productivity of the developer. We use the development time and the number of lines of code as metrics instead of other standard production metrics like function points because we need a very fine-grained evaluation measure.

6.1 Manual Development

At the requirements level, manual development would require to specify the requirements from scratch. Analogously, the application design (according to UML or any other notation), imposes 100% manual design of classes.

At the implementation level, we don’t consider the case of 100% manual code development because this would not be fair: indeed, savvy developers would exploit the basic code generation techniques made available by development IDEs. Therefore, as a baseline we considered the case in which the class stubs were generated automatically by the off-the-shelf ArgoUML code generator. Table 3 shows the effort spent for the development of each feature of the running example application, split between the time spent for classes development and user interface development in case of completely
manual development. The analysis shows an overall effort of 55 hours: 32% of the time was dedicated to the implementation of the class managing the business logics and 68% to the web interface implementation. This is rather typical for Web applications, where the user interface assumes a crucial importance.

Table 3. Development effort (hours.minutes) for basic manual implementation of social web features.

<table>
<thead>
<tr>
<th>feature</th>
<th>classes</th>
<th>UI</th>
<th>total</th>
</tr>
</thead>
<tbody>
<tr>
<td>user</td>
<td>.30</td>
<td>.45</td>
<td>1.15</td>
</tr>
<tr>
<td>profile</td>
<td>.40</td>
<td>2.30</td>
<td>3.10</td>
</tr>
<tr>
<td>friends</td>
<td>1.45</td>
<td>4.00</td>
<td>5.45</td>
</tr>
<tr>
<td>groups</td>
<td>1.45</td>
<td>4.00</td>
<td>5.45</td>
</tr>
<tr>
<td>posts</td>
<td>1.30</td>
<td>4.00</td>
<td>5.30</td>
</tr>
<tr>
<td>guestbook</td>
<td>1.30</td>
<td>4.00</td>
<td>5.30</td>
</tr>
<tr>
<td>comments</td>
<td>.30</td>
<td>1.15</td>
<td>1.45</td>
</tr>
<tr>
<td>inbox</td>
<td>2.00</td>
<td>4.30</td>
<td>6.30</td>
</tr>
<tr>
<td>directmessages</td>
<td>.30</td>
<td>1.15</td>
<td>1.45</td>
</tr>
<tr>
<td>authorizations</td>
<td>1.20</td>
<td>1.45</td>
<td>3.05</td>
</tr>
<tr>
<td>updates</td>
<td>.45</td>
<td>1.45</td>
<td>2.30</td>
</tr>
<tr>
<td>activities</td>
<td>1.45</td>
<td>2.30</td>
<td>4.15</td>
</tr>
<tr>
<td>notifications</td>
<td>1.45</td>
<td>3.30</td>
<td>5.15</td>
</tr>
<tr>
<td>rating</td>
<td>1.20</td>
<td>2.00</td>
<td>3.20</td>
</tr>
<tr>
<td>total</td>
<td>17.35</td>
<td>37.45</td>
<td>55.20</td>
</tr>
</tbody>
</table>

Table 4 shows a more detailed analysis on the size of the implemented classes: each of them is described in terms of number of lines of code splitted in lines automatically generated by ArgoUML, lines manually edited or added, and lines of getters and setters (that can be generated by other code generation tools, like Eclipse). Hence, in the best case 34% of the code is actually developed by hand overall.

6.2 Pattern-based, Subtractive Development

Our approach saves effort and time in every phase of the development. To demonstrate this, we built four different social applications, based on very different requirements and context. The first application corresponds to the running example described in this paper, the second one is a social network based application for trouble ticket management within a company, the third is a simple community of customers for a company, and the fourth is a social platform for members of an association that
collaborate for organizing fund-raising events. We report here the general findings we deduced.

**Table 4. Size of classes (# lines of code), splitted in: automatically generated, manually edited or added, and code for getters and setters.**

<table>
<thead>
<tr>
<th>class</th>
<th>auto</th>
<th>manual</th>
<th>get/set</th>
<th>total</th>
</tr>
</thead>
<tbody>
<tr>
<td>user</td>
<td>34</td>
<td>28</td>
<td>162</td>
<td>224</td>
</tr>
<tr>
<td>profile</td>
<td>25</td>
<td>16</td>
<td>96</td>
<td>137</td>
</tr>
<tr>
<td>friends</td>
<td>29</td>
<td>110</td>
<td>58</td>
<td>197</td>
</tr>
<tr>
<td>group</td>
<td>34</td>
<td>23</td>
<td>100</td>
<td>157</td>
</tr>
<tr>
<td>groups</td>
<td>37</td>
<td>80</td>
<td>30</td>
<td>147</td>
</tr>
<tr>
<td>post</td>
<td>32</td>
<td>51</td>
<td>100</td>
<td>183</td>
</tr>
<tr>
<td>posts</td>
<td>23</td>
<td>73</td>
<td>30</td>
<td>126</td>
</tr>
<tr>
<td>guestbook</td>
<td>29</td>
<td>38</td>
<td>30</td>
<td>97</td>
</tr>
<tr>
<td>comment</td>
<td>25</td>
<td>18</td>
<td>29</td>
<td>72</td>
</tr>
<tr>
<td>inbox</td>
<td>34</td>
<td>87</td>
<td>72</td>
<td>193</td>
</tr>
<tr>
<td>directmessage</td>
<td>19</td>
<td>19</td>
<td>53</td>
<td>91</td>
</tr>
<tr>
<td>message</td>
<td>18</td>
<td>0</td>
<td>51</td>
<td>69</td>
</tr>
<tr>
<td>authorizations</td>
<td>17</td>
<td>19</td>
<td>14</td>
<td>50</td>
</tr>
<tr>
<td>updates</td>
<td>25</td>
<td>142</td>
<td>48</td>
<td>215</td>
</tr>
<tr>
<td>activity</td>
<td>19</td>
<td>14</td>
<td>49</td>
<td>82</td>
</tr>
<tr>
<td>notification</td>
<td>19</td>
<td>13</td>
<td>49</td>
<td>81</td>
</tr>
<tr>
<td>rating</td>
<td>29</td>
<td>35</td>
<td>59</td>
<td>123</td>
</tr>
<tr>
<td><strong>total</strong></td>
<td>448</td>
<td>766</td>
<td>1030</td>
<td>2244</td>
</tr>
</tbody>
</table>

Thanks to our approach, at requirements specification time the taxonomy of known social requirements is predefined. This dramatically reduces the specification effort. In particular, in our experimental settings we only needed to extend the requirements with the non-social aspects of the applications (and thus are outside the scope of this work). On the social side, 100% of the specs were already described and therefore the only task was to drop the non-significant part of the taxonomy. This implied an effort of less than 5% of the one needed for specifying the requirements from scratch.

At the design level, the most frequent need was to extend the specification of the classes (especially the content objects) with appropriate attributes and/or associations. Other examples included dropping some not used classes, or reducing/increasing the levels of inheritance hierarchies between classes. This required an average effort corresponding to 15% of the time required for designing the models from scratch.
Finally, also at implementation level we registered a good increase in performances: at the business logic implementation level, the 34% of manually written code in any other best practice dropped to less than 10% with our approach (from 10% to 1% depending on the application); at the UI implementation level, the 100% of manually implemented JSPs and Servlet fell down to less than 40% on average in case of template-based development. Almost only graphical refinements were accounted for this manual part still remaining. If the WebML models were exploited, the effort would have been reduced further, because only the graphical style templates needed to be developed by hand.

7 RELATED WORK

Several activities have tackled the problem of requirements engineering for software applications. Most notably, goal-oriented techniques such as i* [Yu02] and methods such as Tropos [GMS05] addressed the problem related to describing the requirements in terms of objectives and sub-objectives (with respective positive or negative influences), according to multiple stakeholders. Some works also addressed the problem of deriving software requirements from organizational and business aspects [MPME09], which in a sense is close to the need of mapping the social requirements to the software specification problem. The results of these foundational works have been exploited in this paper: the i* notation has been used for describing the requirements and the work on business requirements has inspired how to define the social specifications.

A few approaches address more precisely the requirement engineering problem within the Web context. Among them, we can mention the work [EK04], which discussed a taxonomy of Web requirements and a preliminary approach to their specification, according to the WebRE approach; the work proposed a mapping to a static model at the design phase (UML class diagrams) and also highlighted how the previous modeling and specification practices for the Web fell short when dealing with the requirement specification. The current paper tries to address this weakness of the field by proposing a method for a specific vertical field on the Web, which can however be extended and generalized to cover further categories and classes of Web applications. Being focused on a specific domain, our approach can apply more precise mapping rules to the design models. The work [GMT09] applied the i* approach to the Web field, by specifying a UML profile for i* for the Web, a mapping to UML models, and a method that grants
traceability. The work presented here applies i* in a similar manner and integrates explicitly defined rules for model and code generation. Analogously, the work [BP04] (also based on i*) proposed an early attempt to apply goal-driven specification to hypertextual Web applications, bridging the gap between requirements and Web design and reusing requirements for usability evaluation; however, the attention in this case was focused mainly on scenario-based techniques for allowing application evaluation in terms of usability. The WebSpec proposal [RGGW10] specifically addressing evolution of requirements, not covered in the current version of this work. The MoLIC approach [PSB05] instead provides a global view of the system’s apparent behavior, from the users’ point of view, with the aim of helping in the specification of the user interfaces and in the definition of the expected user interaction. MoLIC allows designers to build a blueprint of all the interactions that may take place when the application is used, with an interaction-as-conversation metaphor, with a radically different approach with respect to ours. Indeed, we focus more on the functional requirements and in the specification of the expected system behaviors. Finally, among the various Web engineering methods, we can mention OOWS as one of the most complete approaches with respect to requirement specification and social interaction coverage [VP+09]: requirement specification is explicitly supported, a set of design patterns is provided (covering also social aspects) [VPVP09], and a mapping to an executable model is specified too.

While these approaches precisely address Web requirements, their focus and aims are different than ours. They mainly address requirements elicitation, evolution, alignment verification, or they tackle coverage of precise Web issues, but they don’t provide much attention to software artifact reuse, which is the core of our work.

On the other side, a bunch of activities have been carried out in the scientific community for studying the structure and organization of social networking applications. The requirements elicitation phase can benefit also from more economic [TK10] and sociological studies [F10] that try to determine the social behaviour of the users and their motivations, especially within virtual communities [MV+07] [SR+08]. Indeed, requirements and patterns can be elicited also from studies in vertical domains like e-learning, where social interactions have been applied successfully [RL+07]. Some empirical studies have been conducted for analyzing the results of the requirement
collection phase within communities of designers too [YT05], which provided useful insights on the structure and aim of requirements in real settings.

Some works already tried to collect and classify design patterns for social web applications. Among them, we mention [FT+10], collecting WebML design patterns describing the most used social interactions within online platforms. With respect to [FT+10], in this paper we propose a more complete taxonomy of social requirements, collected according to the guidelines presented in [K04], and a mapping to UML designs, that can lead to implementation through mainstream IDE tools. Other preceding works focused on pattern-based development [KR+02] [RS+01] and have been inspirational for this work too.

Our ongoing research [BFV11] is now addressing the new problem of identifying design patterns that bridge the social user interfaces with the business/enterprise web interfaces with the aim of integrating structured business processes with informal social communication.

8 CONCLUSIONS

This paper analyzed the features implemented within existing social networks and has studied an empirical, pattern-based method to specify and design ad-hoc social applications. The upfront cost of requirement and design specification of the social features which is required by our approach is quickly repaid when various applications need to be implemented. Thanks to the broad reuse of artefacts at all the development stages, our approach allows for effort savings of more than 90% in requirement specification, 15% at design time, 14% at business logic implementation time, and 60% at UI implementation time. The productivity gain is obtained thanks to a clear and comprehensive requirement taxonomy, a set of predefined design patterns already in place, and the model transformations that map the requirements to the patterns.

We now plan to extend our approach with: coverage of more detailed requirement definition, also exploiting Web-specific requirement languages beyond goal-oriented specification, for better describing the Web application features and thus getting a higher amount of automatic implementation, both at the business logic and user interface level.
Acknowledgements. This work is part of the research conducted within the BPM4People project, funded within the EU 7FP Capacities program. I wish to thank all the project partners for the useful discussion and especially Davide Ripamonti for his help in the implementation of the approach.

9 REFERENCES


